

3RD EDITION

Principles of Data Science

A beginner's guide to essential math and coding skills
for data fluency and machine learning

A stylized orange logo consisting of several parallel lines forming a series of nested, angular shapes that resemble a stylized 'S' or a series of connected chevrons.

SINAN OZDEMIR



Principles of Data Science

Copyright © 2024 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Group Product Manager: Ali Abidi

Publishing Product Manager: Tejashwini R

Book Project Manager: Farheen Fathima

Content Development Editor: Priyanka Soam

Technical Editor: Kavyashree K S

Copy Editor: Safis Editing

Proofreader: Safis Editing

Indexer: Manju Arasan

Production Designer: Alishon Mendonca

DevRel Marketing Coordinator: Vinishka Kalra

First published: December 2016

Second edition: December 2018

Third edition: Jan 2024

Production reference: 1120124

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK

ISBN 978-1-83763-630-3

I have dedicated many books to many loved ones in the past, and for this edition, I want to dedicate this work to the people of Packt Publishing, who not only gave me my first chance at writing a book when I was early in my career but have stuck by me and continued to release editions with me since.

Thank you to everyone at Packt Publishing for all of your hard work, patience, and dedication to my work!

– Sinan Ozdemir

Contributor

About the author

Sinan Ozdemir is an active lecturer on large language models and a former lecturer of data science at Johns Hopkins University. He is the author of multiple textbooks on data science and machine learning, including *Quick Start Guide to LLMs*. Sinan is currently the founder of LoopGenius, which uses AI to help people and businesses boost their sales, and was previously the founder of the acquired Kylie.ai, an enterprise-grade conversational AI platform with RPA capabilities. He holds a master's degree in pure mathematics from Johns Hopkins University and is based in San Francisco.

About the reviewer

Jigyasa Grover, a 10-time award winner in AI and open source and the co-author of the book *Sculpting Data for ML*, is a powerhouse brimming with passion to make a dent in this world of technology and bridge the gaps. With years of machine learning engineering and data science experience in deploying large-scale systems for monetization on social networking and e-commerce platforms, she primarily focuses on ad prediction, sponsored content ranking, and recommendation. She is an avid proponent of open source and credits her access to opportunities and career growth to this sphere of community development. In her spirit to build a powerful community with a strong belief in the axiom, “We rise by lifting others,” she actively mentors developers and machine learning enthusiasts.

Table of Contents

Preface

1

Data Science Terminology

What is data science?

Understanding basic data science terminology

Why data science?

Example – predicting COVID-19 with machine learning

The data science Venn diagram

The math

Computer programming

Example – parsing a single tweet

Domain knowledge

Some more terminology

Data science case studies

Case study – automating government paper pushing

Case study – what's in a job description?

Summary

2

Types of Data

Structured versus unstructured data

Quantitative versus qualitative data

Digging deeper

The four levels of data

[The nominal level](#)

[Measures of center](#)

[The ordinal level](#)

[The interval level](#)

[The ratio level](#)

[Data is in the eye of the beholder](#)

[Summary](#)

[Questions and answers](#)

3

[The Five Steps of Data Science](#)

[Introduction to data science](#)

[Overview of the five steps](#)

[Exploring the data](#)

[Guiding questions for data exploration](#)

[DataFrames](#)

[Series](#)

[Exploration tips for qualitative data](#)

[Summary](#)

4

[Basic Mathematics](#)

[Basic symbols and terminology](#)

[Vectors and matrices](#)

[Arithmetic symbols](#)

[Summation](#)

[Logarithms/exponents](#)

[Set theory](#)

[Linear algebra](#)

[Matrix multiplication](#)

[How to multiply matrices together](#)

[Summary](#)

5

[Impossible or Improbable – A Gentle Introduction to Probability](#)

[Basic definitions](#)

[What do we mean by “probability”?](#)

[Bayesian versus frequentist](#)

[Frequentist approach](#)

[The law of large numbers](#)

[Compound events](#)

[Conditional probability](#)

[How to utilize the rules of probability](#)

[The addition rule](#)

[Mutual exclusivity](#)

[The multiplication rule](#)

[Independence](#)

[Complementary events](#)

[Introduction to binary classifiers](#)

[Summary](#)

6

[Advanced Probability](#)

[Bayesian ideas revisited](#)

[Bayes’ theorem](#)

More applications of Bayes' theorem

Random variables

Discrete random variables

Continuous random variables

Summary

7

What Are the Chances? An Introduction to Statistics

What are statistics?

How do we obtain and sample data?

Obtaining data

Observational

Experimental

Sampling data

How do we measure statistics?

Measures of center

Measures of variation

The coefficient of variation

Measures of relative standing

The insightful part – correlations in data

The empirical rule

Example – exam scores

Summary

8

Advanced Statistics

Understanding point estimates

Sampling distributions

[Confidence intervals](#)

[Hypothesis tests](#)

[Conducting a hypothesis test](#)

[One-sample t-tests](#)

[Type I and Type II errors](#)

[Hypothesis testing for categorical variables](#)

[Chi-square goodness of fit test](#)

[Chi-square test for association/independence](#)

[Summary](#)

[9](#)

[Communicating Data](#)

[Why does communication matter?](#)

[Identifying effective visualizations](#)

[Scatter plots](#)

[Line graphs](#)

[Bar charts](#)

[Histograms](#)

[Box plots](#)

[When graphs and statistics lie](#)

[Correlation versus causation](#)

[Simpson's paradox](#)

[If correlation doesn't imply causation, then what does?](#)

[Verbal communication](#)

[It's about telling a story](#)

[On the more formal side of things](#)

[The why/how/what strategy for presenting](#)

[Summary](#)

10

How to Tell if Your Toaster is Learning – Machine Learning Essentials

Introducing ML

Example – facial recognition

ML isn't perfect

How does ML work?

Types of ML

SL

UL

RL

Overview of the types of ML

ML paradigms – pros and cons

Predicting continuous variables with linear regression

Correlation versus causation

Causation

Adding more predictors

Regression metrics

Summary

11

Predictions Don't Grow on Trees, or Do They?

Performing naïve Bayes classification

Classification metrics

Understanding decision trees

Measuring purity

Exploring the Titanic dataset

Dummy variables

[Diving deep into UL](#)

[When to use UL](#)

[k-means clustering](#)

[The Silhouette Coefficient](#)

[Feature extraction and PCA](#)

[Summary](#)

12

[Introduction to Transfer Learning and Pre-Trained Models](#)

[Understanding pre-trained models](#)

[Benefits of using pre-trained models](#)

[Commonly used pre-trained models](#)

[Decoding BERT's pre-training](#)

[TL](#)

[Different types of TL](#)

[Inductive TL](#)

[Transductive TL](#)

[Unsupervised TL – feature extraction](#)

[TL with BERT and GPT](#)

[Examples of TL](#)

[Example – Fine-tuning a pre-trained model for text classification](#)

[Summary](#)

13

[Mitigating Algorithmic Bias and Tackling Model and Data Drift](#)

[Understanding algorithmic bias](#)

[Types of bias](#)

[Sources of algorithmic bias](#)

[Measuring bias](#)

[Consequences of unaddressed bias and the importance of fairness](#)

[Mitigating algorithmic bias](#)

[Mitigation during data preprocessing](#)

[Mitigation during model in-processing](#)

[Mitigation during model postprocessing](#)

[Bias in LLMs](#)

[Uncovering bias in GPT-2](#)

[Emerging techniques in bias and fairness in ML](#)

[Understanding model drift and decay](#)

[Model drift](#)

[Data drift](#)

[Mitigating drift](#)

[Understanding the context](#)

[Continuous monitoring](#)

[Regular model retraining](#)

[Implementing feedback systems](#)

[Model adaptation techniques](#)

[Summary](#)

[14](#)

[AI Governance](#)

[Mastering data governance](#)

[Current hurdles in data governance](#)

[Data management: crafting the bedrock](#)

[Data ingestion – the gateway to information](#)

[Data integration – from collection to delivery](#)

[Data warehouses and entity resolution](#)

[The quest for data quality](#)

[Documentation and cataloging – the unsung heroes of governance](#)

[Understanding the path of data](#)

[Regulatory compliance and audit preparedness](#)

[Change management and impact analysis](#)

[Upholding data quality](#)

[Troubleshooting and analysis](#)

[Navigating the intricacy and the anatomy of ML governance](#)

[ML governance pillars](#)

[Model interpretability](#)

[The many facets of ML development](#)

[Beyond training – model deployment and monitoring](#)

[A guide to architectural governance](#)

[The five pillars of architectural governance](#)

[Transformative architectural principles](#)

[Zooming in on architectural dimensions](#)

[Summary](#)

[15](#)

[Navigating Real-World Data Science Case Studies in Action](#)

[Introduction to the COMPAS dataset case study](#)

[Understanding the task/outlining success](#)

[Preliminary data exploration](#)

[Preparing the data for modeling](#)

[Final thoughts](#)

[Text embeddings using pretrained models and OpenAI](#)

[Setting up and importing necessary libraries](#)

[Data collection – fetching the textbook data](#)

[Converting text to embeddings](#)

[Querying – searching for relevant information](#)

[Concluding thoughts – the power of modern pre-trained models](#)

[Summary](#)

[Index](#)

[Other Books You May Enjoy](#)

Preface

Principles of Data Science bridges mathematics, programming, and business analysis, empowering you to confidently pose and address complex data questions and construct effective machine learning pipelines. This book will equip you with the tools you need to transform abstract concepts and raw statistics into actionable insights.

Starting with cleaning and preparation, you'll explore effective data mining strategies and techniques before moving on to building a holistic picture of how every piece of the data science puzzle fits together. Throughout the book, you'll discover statistical models with which you can control and navigate even the densest or sparsest of datasets and learn how to create powerful visualizations that communicate the stories hidden in your data.

With a focus on application, this edition covers advanced transfer learning and pre-trained models for NLP and vision tasks. You'll get to grips with advanced techniques for mitigating algorithmic bias in data as well as models and addressing model and data drift. Finally, you'll explore medium-level data governance, including data provenance, privacy, and deletion request handling.

By the end of this data science book, you'll have learned the fundamentals of computational mathematics and statistics, all while navigating the intricacies of modern machine learning and large pre-trained models such as GPT and BERT.

Who is this book for?

If you are an aspiring novice data scientist eager to expand your knowledge, this book is for you.

Whether you have basic math skills and want to apply them in the field of data science, or you excel in programming but lack the necessary mathematical foundations, you'll find this book useful. Familiarity with Python programming will further enhance your learning experience.

What this book covers

[*Chapter 1, Data Science Terminology*](#), describes the basic terminology used by data scientists. We will cover the differences between often-confused terms as well as looking at examples of each term used in order to truly understand how to communicate in the language of data science. We will begin by looking at the broad term *data science* and then, little by little, get more specific until we arrive at the individual subdomains of data science, such as machine learning and statistical inference. This chapter will also look at the three main areas of data science, which are math, programming, and domain expertise. We will look at each one individually and understand the uses of each. We will also look at the basic Python packages and the syntax that will be used throughout the book.

[*Chapter 2, Types of Data*](#), deals with data types and the way data is observed. We will explore the different levels of data as well as the different forms of data. Specifically, we will understand the differences between structured/unstructured data, quantitative/qualitative data, and more.

[*Chapter 3, The Five Steps of Data Science*](#), deals with the data science process as well as data wrangling and preparation. We will go into the five steps of data science and give examples of the process at every step of the way. After we cover the five steps of data science, we will turn to data wrangling, which is the data exploration/preparation stage of the process. In order to best understand these principles, we will use extensive examples to explain each step. I will also provide tips to look for when exploring data, including looking for data on different scales, categorical variables, and missing data. We will use pandas to check for and fix all of these things.

[*Chapter 4, Basic Mathematics*](#), goes over the elementary mathematical skills needed by any data scientist. We will dive into functional analysis and use matrix algebra as well as calculus to show and prove various outcomes based on real-world data problems.

[*Chapter 5, Impossible or Improbable – A Gentle Introduction to Probability*](#), focuses heavily on the basic probability that is required for data science. We will derive results from data using probability rules and begin to see how we view real-world problems using probability. This chapter will be highly practical and Python will be used to code the examples.

[*Chapter 6, Advanced Probability*](#), is where we explore how to use Python to solve more complex probability problems and also look at a new type of probability called Bayesian inference. We will use these theorems to solve real-world data scenarios such as weather predictions.

[*Chapter 7, What Are the Chances? An Introduction to Statistics*](#), is on basic statistics, which is required for data science. We will also explore the types of statistical errors, including type I and type II errors, using examples. These errors are as essential to our analysis as the actual results. Errors and

their different types allow us to dig deeper into our conclusions and avoid potentially disastrous results. Python will be used to code up statistical problems and results.

[Chapter 8](#), *Advanced Statistics*, is where normalization is key. Understanding why and how we normalize data will be crucial. We will cover basic plotting, such as scatter plots, bar plots, and histograms. This chapter will also get into statistical modeling using data. We will not only define the concept as using math to model a real-world situation, but we will also use real data in order to extrapolate our own statistical models. We will also discuss overfitting. Python will be used to code up statistical problems and results.

[Chapter 9](#), *Communicating Data*, deals with the different ways of communicating results from our analysis. We will look at different presentation styles as well as different visualization techniques. The point of this chapter is to take our results and be able to explain them in a coherent, intelligible way so that anyone, whether they are data-savvy or not, may understand and use our results. Much of what we will discuss will be how to create effective graphs through labels, keys, colors, and more. We will also look at more advanced visualization techniques such as parallel coordinates plots.

[Chapter 10](#), *How to Tell if Your Toaster is Learning – Machine Learning Essentials*, focuses on machine learning as a part of data science. We will define the different types of machine learning and see examples of each kind. We will specifically cover areas in regression, classification, and unsupervised learning. This chapter will cover what machine learning is and how it is used in data science. We will revisit the differences between machine learning and statistical modeling and how machine learning is a broader category of the latter. Our aim will be to utilize statistics and probability in order to understand and apply essential machine learning skills to practical industries such as marketing. Examples will include predicting star ratings of restaurant reviews, predicting the presence of disease, spam email detection, and much more. This chapter focuses more on statistical and probabilistic models. The next chapter will deal with models that do not fall into this category. We will also focus on metrics that tell us how accurate our models are. We will use metrics in order to conclude results and make predictions using machine learning.

[Chapter 11](#), *Predictions Don't Grow on Trees, or Do They?*, focuses heavily on machine learning that is not considered a statistical or probabilistic model. These constitute models that cannot be contained in a single equation, such as linear regression or naïve Bayes. The models in this chapter are, while still based on mathematical principles, more complex than a single equation. The models include KNN, decision trees, and an introduction to unsupervised clustering. Metrics will become very important here as they will form the basis for measuring our understanding and our models. We will also peer into some of the ethics of data science in this chapter. We will see where machine learning can perhaps boundaries in areas such as privacy and advertising and try to draw a conclusion about the ethics of predictions.

[Chapter 12](#), *Introduction to Transfer Learning and Pre-Trained Models*, introduces transfer learning and gives examples of how to transfer a machine’s learning from a pre-trained model to fine-tuned models. We will navigate the world of open source models and achieve state-of-the-art performance in NLP and vision tasks.

[Chapter 13](#), *Mitigating Algorithmic Bias and Tackling Model and Data Drift*, introduces algorithmic bias and how to quantify, identify, and mitigate biases in data and models. We will see how biased data can lead to biased models. We will also see how we can identify bias as early as possible and catch new biases that arise in existing models.

[Chapter 14](#), *AI Governance*, introduces drift in models and data and the proper ways to quantify and combat drift. We will see how data can drift over time and how we can update models properly to combat draft to keep our pipelines as performant as possible.

[Chapter 15](#), *Navigating Real-World Data Science Case Studies in Action*, introduces basic governance structures and how to navigate deletion requests, privacy/permission structures, and data provenance.

To get the most out of this book

You will need Python version 3.4 or higher with the specified Python package versions of libraries specified in the GitHub `requirements.txt` file. You can install Python using pip or conda or even run our code on Google Colab if you wish!

Software/hardware covered in the book	Operating system requirements
Python	Windows, macOS, or Linux

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book’s GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

If you are looking for more content on machine learning/AI/large language models, check out Sinan’s other books and courses online at `sinanozdemir.ai`!

Download the example code files

You can download the example code files for this book from GitHub at <https://github.com/PacktPublishing/Principles-of-Data-Science-Third-Edition>. If there’s an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

code in text: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: “In this example, the tweet in question is RT @robdtv: \$TWTR now top holding for Andor, unseating \$AAPL. ”

A block of code is set as follows:

```
tweet = "RT @j_o_n_dnger: $TWTR now top holding for Andor, unseating $AAPL"
words_in_tweet = tweet.split(' ') # list of words in tweet
for word in words_in_tweet: # for each word in list
    if "$" in word: # if word has a "cashtag"
        print("THIS TWEET IS ABOUT", word) # alert the user
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: “The **words_in_tweet** variable tokenizes the tweet (separates it by word).”

TIPS OR IMPORTANT NOTES

Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at customer@packtpub.com and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/support/errata and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Share Your Thoughts

Once you've read *Principles of Data Science*, we'd love to hear your thoughts! Please [click here to go straight to the Amazon review page](#) for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below



<https://packt.link/free-ebook/9781837636303>

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly

Data Science Terminology

We live in the Data Age. No matter the industry you work in, be it IT, fashion, food, or finance, there is no doubt that data affects your life and work. At some point today, this week, or this month, you will either have or hear about a conversation about data. News outlets are covering more and more stories about data leaks, cybercrimes, and how modern artificial intelligence and machine learning algorithms are changing the way we work and live.

In this book, we will attempt to cover, to put it simply, the principles of how we should interpret, interact with, manipulate, and utilize data. We will attempt to cover the principles of data science. Before we can begin covering such a huge topic, first, we have to build a solid foundation below our feet.

To begin our journey, this chapter will explore the terminology and vocabulary of the modern data scientist. We will learn keywords and phrases that will be essential in our discussion of data science throughout this book. We will also learn why we use data science and learn about the three key domains that data science is derived from before we begin to look at the code in Python, the primary language that will be used in this book.

This chapter will cover the following topics:

- The basic terminology of data science
- The three domains of data science
- The basic Python syntax

What is data science?

This is a simple question, but before we go any further, let's look at some basic definitions that we will use throughout this book. The great/awful thing about the field of data science is that it is young enough that sometimes, even basic definitions and terminology can be debated across publications and people. The basic definition is that *data science is the process of acquiring knowledge through data*.

It may seem like a small definition for such a big topic, and rightfully so! Data science covers so many things that it would take pages to list them all out. Put another way, data science is all about how we take data, use it to acquire knowledge, and then use that knowledge to do the following:

- Make informed decisions
- Predict the future
- Understand the past/present

- Create new industries/products

This book is all about the methods of data science, including how to process data, gather insights, and use those insights to make informed decisions and predictions.

Understanding basic data science terminology

The definitions that follow are general enough to be used in daily conversations and work to serve the purpose of this book, which is *an introduction to the principles of data science*.

Let's start by defining what data is. This might seem like a silly first definition to look at, but it is very important. Whenever we use the word "data," we refer to a collection of information in either a structured or unstructured format. These formats have the following qualities:

- **Structured data:** This refers to data that is sorted into a row/column structure, where every row represents a single observation and the columns represent the characteristics of that observation
- **Unstructured data:** This is the type of data that is in a free form, usually text or raw audio/signals that must be parsed further to become structured

Data is everywhere around us and originates from a multitude of sources, including everyday internet browsing, social media activities, and technological processes such as system logs. This data, when structured, becomes a useful tool for various algorithms and businesses. Consider the data from your online shopping history. Each transaction you make is recorded with details such as the product, price, date and time, and payment method. This structured information, laid out in rows and columns, forms a clear picture of your shopping habits, preferences, and patterns.

Yet not all data comes neatly packaged. Unstructured data, such as comments and reviews on social media or an e-commerce site, don't follow a set format. They might include text, images, or even videos, making it more challenging to organize and analyze. However, once processed correctly, this free-flowing information offers valuable insights such as sentiment analysis, providing a deeper understanding of customer attitudes and opinions. In essence, the ability to harness both structured and unstructured data is key to unlocking the potential of the vast amounts of information we generate daily.

Opening Excel, or any spreadsheet software, presents you with a blank grid meant for structured data. It's not ideally suited for handling unstructured data. While our primary focus will be structured data, given its ease of interpretation, we won't overlook the richness of raw text and other unstructured data types, and the techniques to make them comprehensible.

The crux of data science lies in employing data to unveil insights that would otherwise remain hidden. Consider a healthcare setting, where data science techniques can predict which patients are likely not to attend their appointments. This not only optimizes resource allocation but also ensures other patients can

utilize these slots. Understanding data science is more than grasping what it does – it's about appreciating its importance and recognizing why mastering it is in such high demand.

Why data science?

Data science won't replace the human brain (at least not for a while), but rather augment and complement it, working alongside it. Data science should not be thought of as an end-all solution to our data woes; it is merely an opinion – a very informed opinion, but still an opinion, nonetheless. It deserves a seat at the table.

In this Data Age, it's clear that we have a surplus of data. But why should that necessitate an entirely new set of vocabulary? What was wrong with our previous forms of analysis? For one, the sheer volume of data makes it impossible for a human to parse it in a reasonable time frame. Data is collected in various forms and from different sources and often comes in a very unstructured format.

Data can be missing, incomplete, or just flat-out wrong. Oftentimes, we will have data on very different scales, and that makes it tough to compare it. Say we are looking at data concerning pricing used cars. One characteristic of a car is the year it was made, and another might be the number of miles on that car. Once we clean our data (which we will spend a great deal of time looking at in this book), the relationships between the data become more obvious, and the knowledge that was once buried deep in millions of rows of data simply pops out. One of the main goals of data science is to make explicit practices and procedures to discover and apply these relationships in the data.

Let's take a minute to discuss its role today using a very relevant example.

Example – predicting COVID-19 with machine learning

A large component of this book is about how we can leverage powerful machine learning algorithms, including deep learning, to solve modern and complicated tasks. One such problem is using deep learning to be able to aid in the diagnosis, treatment, and prevention of fatal illnesses, including COVID-19. Since the global pandemic erupted in 2020, numerous organizations around the globe turned to data science to alleviate and solve problems related to COVID-19. For example, the following figure shows a visualization of a process for using machine learning (deep learning, in this case) to screen for COVID-19 that was published in March 2020. By then, the world had only known of COVID-19 for a few months, and yet we were able to apply machine learning techniques to such a novel use case with relative ease:

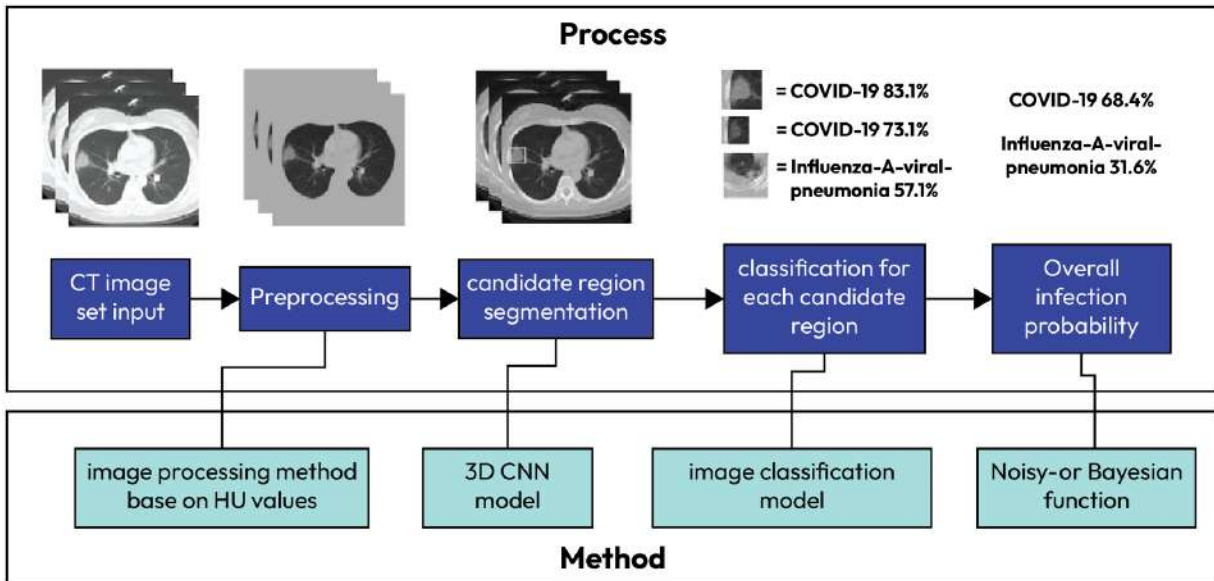


Figure 1.1 – A visualization of a COVID-19 screening algorithm based on deep learning from 2020

This screening algorithm was one of the first of its kind working to identify COVID-19 and recognize it from known illnesses such as the flu. Algorithms like these suggested that we could turn to data and machine learning to aid when unforeseen catastrophes strike. We will learn how to develop algorithms such as this life-changing system later in this book. Creating such algorithms takes a combination of three distinct skills that, when combined, form the backbone of data science. It requires people who are knowledgeable about COVID-19, people who know how to create statistical models, and people who know how to productionize those models so that people can benefit from them.

The data science Venn diagram

It is a common misconception that only those with advanced degrees such as a Ph.D. or math prodigies can understand the math/programming behind data science. This is false. Understanding data science begins with three basic areas:

- **Math/statistics:** This involves using equations and formulas to perform analysis
- **Computer programming:** This is the ability to use code to create outcomes on a computer
- **Domain knowledge:** This refers to understanding the problem domain (medicine, finance, social science, and so on)

The following Venn diagram provides a visual representation of how these three areas of data science intersect:

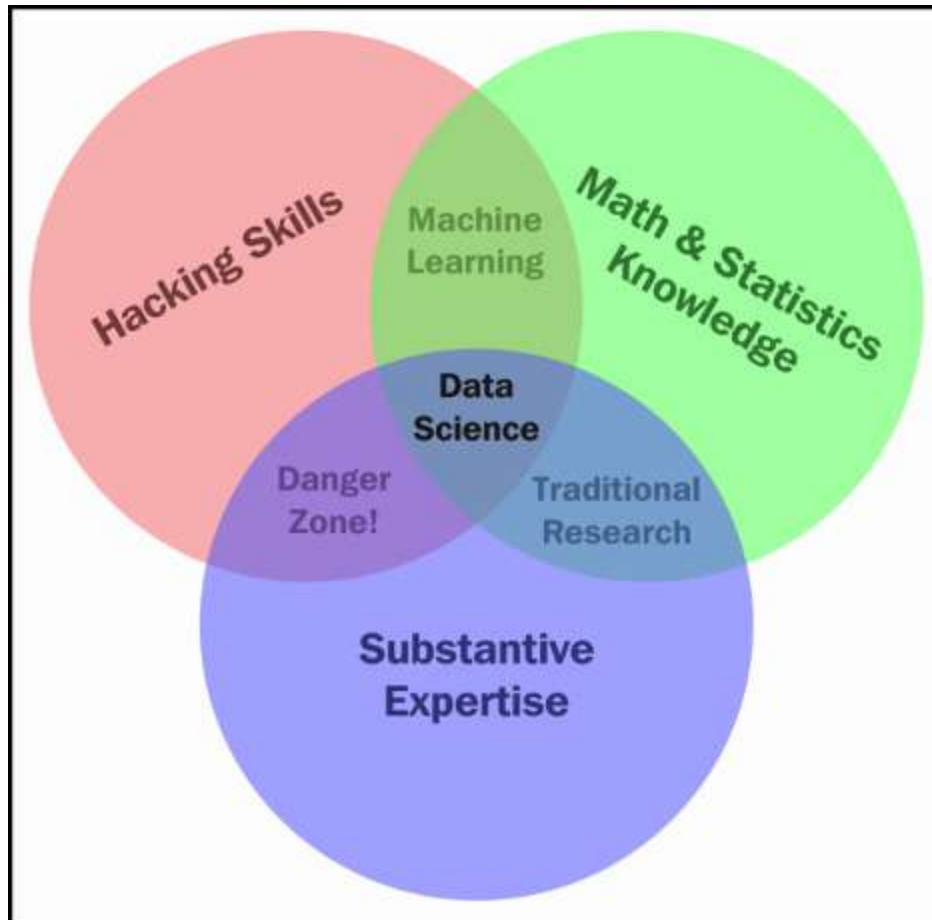


Figure 1.2 – The Venn diagram of data science

Those with hacking skills can conceptualize and program complicated algorithms using computer languages. Having a math and statistics background allows you to theorize and evaluate algorithms and tweak the existing procedures to fit specific situations. Having substantive expertise (domain expertise) allows you to apply concepts and results in a meaningful and effective way.

While having only two of these three qualities can make you intelligent, it will also leave a gap. Let's say that you are very skilled in coding and have formal training in day trading. You might create an automated system to trade in your place but lack the math skills to evaluate your algorithms. This will mean that you end up losing money in the long run. It is only when you boost your skills in coding, math, and domain knowledge that you can truly perform data science.

The quality that was probably a surprise for you was domain knowledge. It is just knowledge of the area you are working in. If a financial analyst started analyzing data about heart attacks, they might need the help of a cardiologist to make sense of a lot of the numbers.

Data science is the intersection of the three key areas mentioned earlier. To gain knowledge from data, we must be able to utilize computer programming to access the data, understand the mathematics

behind the models we derive, and, above all, understand our analyses' place in the domain we are in. This includes the presentation of data. If we are creating a model to predict heart attacks in patients, is it better to create a PDF of information or an app where we can type in numbers and get a quick prediction? All these decisions must be made by the data scientist.

The intersection of math and coding is machine learning. This book will look at machine learning in great detail later on, but it is important to note that without the explicit ability to generalize any models or results to a domain, machine learning algorithms remain just that – algorithms sitting on your computer. You might have the best algorithm to predict cancer. You could be able to predict cancer with over 99% accuracy based on past cancer patient data, but if you don't understand how to apply this model in a practical sense so that doctors and nurses can easily use it, your model might be useless.

Both computer programming and math will be covered extensively in this book. Domain knowledge comes with both the practice of data science and reading examples of other people's analyses.

The math

Most people stop listening once someone says the word "math." They'll nod along in an attempt to hide their utter disdain for the topic but hear me out. As an experienced math teacher, I promise that this book will guide you through the math needed for data science, specifically statistics and probability. We will use these subdomains of mathematics to create what are called **models**. A data model refers to an organized and formal relationship between elements of data, usually meant to simulate a real-world phenomenon.

The central idea of using math is that we will use it to formalize relationships between variables. As a former pure mathematician and current math teacher, I know how difficult this can be. I will do my best to explain everything as clearly as I can. Of the three areas of data science, math is what allows us to move from domain to domain. Understanding the theory allows us to apply a model that we built for the fashion industry to a financial domain.

The math covered in this book ranges from basic algebra to advanced probabilistic and statistical modeling. Do not skip over these chapters, even if you already know these topics or you're afraid of them. Every mathematical concept that I will introduce will be introduced with care and purpose, using examples. The math in this book is essential for data scientists.

There are many types of data models, including probabilistic and statistical models. Both of these are subsets of a larger paradigm, called **machine learning**. The essential idea behind these three topics is that we use data to come up with the best model possible. We no longer rely on human instincts – rather, we rely on data. Math and coding are vehicles that allow data scientists to step back and apply their skills virtually anywhere.

Computer programming

Let's be honest: you probably think computer science is way cooler than mathematics. That's OK, I don't blame you. The news isn't filled with math news like it is with news on technology (although I think that's a shame). You don't turn on the TV to see a new theory on primes or about Euler's equation – rather, you see investigative reports on how the latest smartphone can take better photos of cats or how generative AI models such as ChatGPT can learn to create websites from scratch. Computer languages are how we communicate with machines and tell them to do our bidding. A computer speaks many languages and, like a book, can be written in many languages; similarly, data science can also be done in many languages. Python, Julia, and R are some of the many languages that are available to us. This book will exclusively use Python.

Why Python?

We will use Python for a variety of reasons, listed as follows:

- Python is an extremely simple language to read and write, even if you've never coded before, which will make future examples easy to understand and read later on, even after you have read this book.
- It is one of the most common languages, both in production and in an academic setting (one of the fastest growing, as a matter of fact).
- The language's online community is vast and friendly. This means that a quick search for the solution to a problem should yield many people who have faced and solved similar (if not the same) situations.
- Python has prebuilt data science modules that both novice and veteran data scientists can utilize.

The last point is probably the biggest reason we will focus on Python. These pre built modules are not only powerful but also easy to pick up. By the end of the first few chapters, you will be very comfortable with these modules. Some of these modules include the following:

- pandas
- PyTorch
- Scikit-learn
- Seaborn
- NumPy/scipy
- Requests (to mine data from the web)
- BeautifulSoup (for web-HTML parsing)

We will assume that you have basic Python coding skills. This includes the ability to recognize and use the basic types (`int`, `float`, `boolean`, and so on) and create functions and classes with ease. We will also assume that you have mastery over logistical operators, including `==`, `>=`, and `<=`.

Example – parsing a single tweet

Here is some Python code that should be readable to you. In this example, I will be parsing some tweets about stock prices. If you can follow along with this example easily, then you are ready to proceed!

```
tweet = "RT @j_o_n_dnger: $TWTR now top holding for Andor, unseating $AAPL"
words_in_tweet = tweet.split(' ') # list of words in tweet
for word in words_in_tweet: # for each word in list
    if "$" in word: # if word has a "cashtag"
        print("THIS TWEET IS ABOUT", word) # alert the user
```

I will point out a few things about this code snippet line by line, as follows:

1. First, we set a variable to hold some text (known as a string in Python). In this example, the tweet in question is **RT @robdiv: \$TWTR now top holding for Andor, unseating \$AAPL**.
2. The **words_in_tweet** variable tokenizes the tweet (separates it by word). If you were to print this variable, you would see the following:

```
['RT', '@robdiv:', '$TWTR', 'now', 'top', 'holding', 'for', 'Andor,', 'unseating', '$AAPL']
```

3. We iterate through this list of words using a **for** loop, going through the list one by one.
4. Here, we have another **if** statement. For each word in this tweet, if the word contains the **\$** character, this represents stock tickers on Twitter.

```
if "$" in word: # if word has a "cashtag"
```

5. If the preceding **if** statement is **True** (that is, if the tweet contains a cashtag), print it and show it to the user.

The output of this code will be as follows:

```
THIS TWEET IS ABOUT $TWTR

THIS TWEET IS ABOUT $AAPL
```

Whenever I use Python in this book, I will ensure that I am as explicit as possible about what I am doing in each line of code. I know what it's like to feel lost in code. I know what it's like to see someone coding and want them to just slow down and explain what is going on.

As someone who is entirely self-taught in Python and has also taught Python at the highest levels, I promise you that I will do everything in my power to not let you feel that way when reading this book.

Domain knowledge

As I mentioned earlier, domain knowledge focuses mainly on knowing the particular topic you are working on. For example, if you are a financial analyst working on stock market data, you have a lot of domain knowledge. If you are a journalist looking at worldwide adoption rates, you might benefit from consulting an expert in the field. This book will attempt to show examples from several problem domains, including medicine, marketing, finance, and even UFO sightings!

Does this mean that if you're not a doctor, you can't work with medical data? Of course not! Great data scientists can apply their skills to any area, even if they aren't fluent in it. Data scientists can adapt to the field and contribute meaningfully when their analysis is complete.

A big part of domain knowledge is presentation. Depending on your audience, it can matter greatly how you present your findings. Your results are only as good as your vehicle of communication. You can predict the movement of the market with 99.99% accuracy, but if your program is impossible to execute, your results will go unused. Likewise, if your vehicle is inappropriate for the field, your results will go equally unused. I know I'm throwing a lot at you already, but we should look at just a few more relevant terminologies so that we can hit the ground running.

Some more terminology

At this point, you're probably excitedly looking up a lot of data science material and seeing words and phrases I haven't used yet. Here are some common terms that you are likely to encounter:

- **Machine learning:** This refers to giving computers the ability to learn from data without explicit "rules" being given by a programmer. Earlier in this chapter, we saw the concept of machine learning as the union of someone who has both coding and math skills. Here, we are attempting to formalize this definition. Machine learning combines the power of computers with intelligent learning algorithms to automate the discovery of relationships in data and create powerful data models.
- **Statistical model:** This refers to taking advantage of statistical theorems to formalize relationships between data elements in a (usually) simple mathematical formula.
- **Exploratory data analysis (EDA):** This refers to preparing data to standardize results and gain quick insights. EDA is concerned with data visualization and preparation. This is where we turn unstructured data into structured data and clean up missing/incorrect data points. During EDA, we will create many types of plots and use these plots to identify key features and relationships to exploit in our data models.
- **Data mining:** This is the process of finding relationships between elements of data. Data mining is the part of data science where we try to find relationships between variables (think the spawn-recruit model).

I have tried pretty hard not to use the term **big data** up until now. This is because I think this term is misused – a lot. Big data is data that is too large to be processed by a single machine (if your laptop crashed, it might be suffering from a case of big data).

The following diagram shows the relationship between these data science concepts.

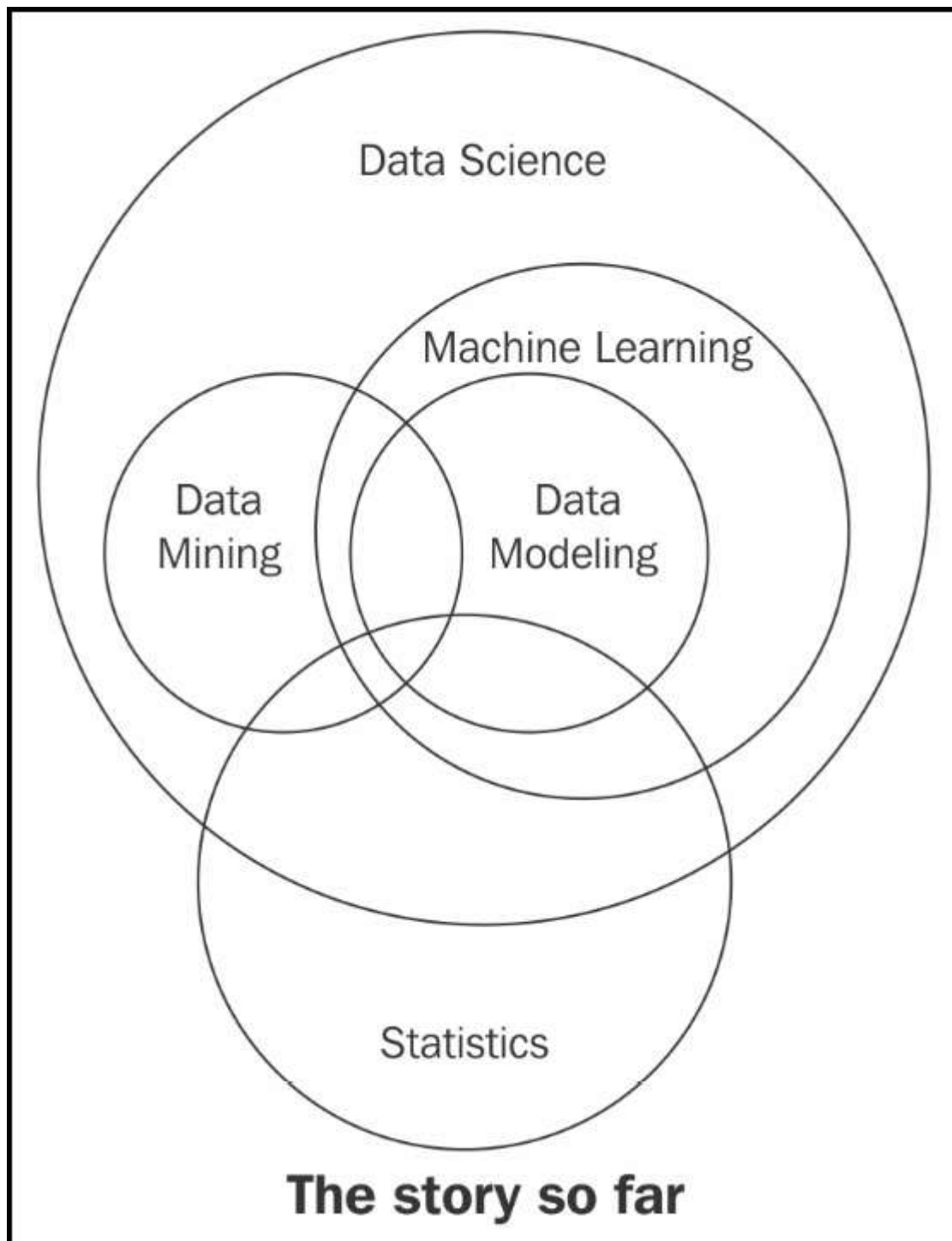


Figure 1.3 – The state of data science (so far)

With these terms securely stored in our brains, we can move on to the main educational resource in this book: data science case studies.

Data science case studies

We will spend much of this book looking at real-life examples of using data science and machine learning. The combination of math, computer programming, and domain knowledge is what makes data science so powerful but it can often be too abstract without concrete coding examples.

Oftentimes, it is difficult for a single person to master all three of these areas. That's why it's very common for companies to hire teams of data scientists instead of a single person. Let's look at a few powerful examples of data science in action and its outcomes.

Case study – automating government paper pushing

Social security claims are known to be a major hassle for both the agent reading them and the person who wrote the claims. Some claims take over two years to get resolved in their entirety, and that's absurd! Let's look at the following figure, which shows what goes into a claim:



B. To be completed by the claimant		
PLEASE PRINT		
Please Answer the Following Questions:		
(1) Have you been treated or examined by a doctor (other than a doctor at a hospital) since the above date?  <input type="checkbox"/> Yes <input type="checkbox"/> No		
<i>(If yes, please list the names, addresses and telephone numbers of doctors who have treated or examined you since the above date. Also list the dates of treatment or examination. If possible, send updated reports from these doctors to the Administrative Law Judge before the date of your hearing.)</i>		
DOCTORS NAME(S)	ADDRESS(ES) & TELEPHONE NO.(S)	DATE(S)
(2) What have these doctors told you about your condition?		
(3) Have you been hospitalized since the above date?  <input type="checkbox"/> Yes <input type="checkbox"/> No		
<i>(If yes, please list the name and address of the hospital. Also, explain why you were hospitalized and what treatment you received.)</i>		

Figure 1.4 – Sample social security form

Not bad. It's mostly just text, though. Fill this in, then that, then this, and so on. You can see how it would be difficult for an agent to read these all day, form after form. There must be a better way!

Well, there is. Elder Research Inc. parsed this unstructured data and was able to automate 20% of all disability social security forms. This means that a computer could look at 20% of these written forms and give its opinion on the approval.

Apart from this, the third-party company that is hired to rate the approvals of the forms gave the machine-graded forms a higher grade than the human forms. So, not only did the computer handle 20% of the load on average, but it also did better than a human.

Modern language models such as GPT-3 and BERT have taken the world of NLP by storm by pushing the boundaries of what we thought was possible way beyond its previously considered limits. We will spend a lengthy amount of time talking about these models later in this book.

Fire all humans, am I right?

Before I get a load of angry emails and tweets claiming that data science is bringing about the end of human workers, keep in mind that the computer was only able to handle 20% of the load in our previous example. This means that it probably performed terribly on 80% of the forms! This is because the computer was probably great at simple forms. The claims that would have taken a human minutes to compute took the computer seconds. But these minutes add up, and before you know it, each human is being saved over an hour a day!

Forms that might be easy for a human to read are also likely easy for the computer. It's when the forms are very terse, or when the writer starts deviating from the usual grammar, that the computer starts to fail. This model is great because it lets humans spend more time on those difficult claims and give them more attention without getting distracted by the sheer volume of papers.

Note that I used the word "model." Remember that a model is a relationship between elements. In this case, the relationship is between written words and the approval status of a claim.

Case study – what's in a job description?

Looking for a job in data science? Great! Let me help. In this case study, I have **scraped** (used code to read from the web) 1,000 job descriptions for companies that are actively hiring data scientists. The goal here is to look at some of the most common keywords that people use in their job descriptions, as shown in the following screenshot:

Machine Learning Quantitative Analyst

Bloomberg - ★★★★★ 282 reviews - New York, NY

The Machine Learning Quantitative Analyst will work in Bloomberg's Enterprise Solutions area and work collaboratively to build a liquidity tool for banks,...

8 days ago - [email](#)

Sponsored

Save lives with machine learning

Blue Owl - San Francisco, CA

Requirements for all data scientists. Expert in Python and core libraries used by data scientists (Numpy, Scipy, Pandas, Scikit-learn, Matplotlib/Seaborn, etc.)...

30+ days ago - [email](#)

Sponsored

Data Scientist

Indeed - ★★★★★ 132 reviews - Austin, TX

How a Data Scientist works. As a Data Scientist at Indeed your role is to follow the data. We are looking for a mixture between a statistician, scientist,...

Easily apply

30+ days ago - [email](#)

Sponsored

Figure 1.5 – An example of data scientist job listings

In the following Python code, the first two imports are used to grab web data from `indeed.com`, and the third import is meant to simply count the number of times a word or phrase appears, as shown in the following code:

```
import requests
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import CountVectorizer
# grab postings from the web
texts = []
# cycle through 100 pages of indeed job resources
for i in range(0,1000,10):
    response = requests.get('http://www.indeed.com/jobs?q=data+scientist&sta
rt='+str(i)).text
    soup = BeautifulSoup(response)
    texts += [a.text for a in soup.findAll('span', {'class':'summary'})]
print(type(texts))
print(texts[0]) # first job description
```

All this loop is doing is going through 100 pages of job descriptions, and for each page, it is grabbing each job description. The important variable here is `texts`, which is a list of over 1,000 job descriptions, as shown in the following code:

```
type(texts) # == list
vectorizer = CountVectorizer(ngram_range=(1,2), stop_words='english')
# Get basic counts of one and two word phrases
matrix = vectorizer.fit_transform(texts)
```

```
# fit and learn to the vocabulary in the corpus
print len(vect.get_feature_names()) # how many features there are
```

There are 10,857 total one- and two-word phrases in my case! Since web pages are scraped in real time and these pages may change when you run this code, you may get a different number than 10,857.

I have omitted some code here because we will cover these packages in more depth in our NLP chapters later, but it exists in the GitHub repository for this book. The results are as follows (represented by the phrase and then the number of times it occurred):

```
10857
data 2641
analytics 857
learning 570
machine 545
machine learning 529
science 414
business 398
experience 387
marketing 313
scientist 296
data science 293
data scientist 287
marketing analytics 265
scientists 258
statistics 241
data scientists 240
algorithms 238
data analytics 229
experience data 225
applied 215
using 207
relevant 203
learning algorithms 202
analytics data 197
best 186
```

Figure 1.6 – The top one- and two-word phrases when looking at job descriptions on Indeed for the title of “Data Scientist”

There are many interesting things to note about this case study, but the biggest takeaway is that there are many keywords and phrases that make up a data science role. It isn’t just math, coding, or domain knowledge; it truly is a combination of these three ideas (whether exemplified in a single-person team or across a multi-person team) that makes data science possible and powerful.

Summary

In this chapter, we explored basic data science terminology and saw how even the term “data science” can be fraught with ambiguity and misconception. We also learned that coding, math, and domain expertise are the fundamental building blocks of data science. As we seek new and innovative ways to discover data trends, a beast lurks in the shadows. I’m not talking about the learning curve of

mathematics or programming, nor am I referring to the surplus of data. The Industrial Age left us with an ongoing battle against pollution. The subsequent Information Age left behind a trail of big data. So, what dangers might the Data Age bring us?

The Data Age can lead to something much more sinister – the dehumanization of the individual through mass data and the rise of automated bias in machine learning systems.

More and more people are jumping headfirst into the field of data science, most with no prior experience in math or computer science, which, on the surface, is great. The average data scientist has access to millions of dating profiles' data, tweets, online reviews, and much more to jump-start their education. However, if you jump into data science without the proper exposure to theory or coding practices, and without respect for the domain you are working in, you face the risk of oversimplifying the very phenomenon you are trying to model.

Now, it's time to begin. In the next chapter, we will explore the different types of data that exist in the world, ranging from free-form text to highly structured row/column files. We will also look at the mathematical operations that are allowed for different types of data, as well as deduce insights based on the form the data comes in.

Types of Data

For our first step into the world of data science, let's take a look at the various ways in which data can be formed. In this chapter, we will explore three critical categorizations of data:

- Structured versus unstructured data
- Quantitative versus qualitative data
- The four levels of data

We will dive further into each of these topics by showing examples of how data scientists look at and work with data. This chapter aims to familiarize us with the fundamental types of data so that when we eventually see our first dataset, we will know exactly how to dissect, diagnose, and analyze the contents to maximize our insight value and machine learning performance.

The first thing to note is my use of the word *data*. In the previous chapter, I defined data as merely a collection of information. This vague definition exists because we may separate data into different categories and need our definition to be loose.

The next thing to remember while we go through this chapter is that for the most part, when I talk about the type of data, I will refer to either a specific characteristic (column/feature) of a dataset or the entire dataset as a whole. I will be very clear about which one I refer to at any given time.

At first thought, it might seem worthless to stop and think about what type of data we have before getting into the fun stuff, such as statistics and machine learning, but this is arguably one of the most important steps you need to take to perform data science.

When given a new dataset to analyze, it is tempting to jump right into exploring, applying statistical models, and researching the applications of machine learning to get results as soon as possible.

However, if you don't understand the type of data that you are working with, then you might waste a lot of time applying models that are known to be ineffective with that specific type of data.

Structured versus unstructured data

The first question we want to ask ourselves about an entire dataset is whether we are working with structured or unstructured data. The answer to this question can mean the difference between needing three days or three weeks to perform a proper analysis.

The basic breakdown is as follows (this is a rehashed definition of organized and unorganized data from [Chapter 1](#)):

- **Structured (that is, organized) data:** This is data that can be thought of as observations and characteristics. It is usually organized using a table method (rows and columns) that can be organized in a spreadsheet format or a relational database.
- **Unstructured (that is, unorganized) data:** This data exists as a free entity and does not follow any standard organization hierarchy such as images, text, or videos.

Here are a few examples that could help you differentiate between the two:

- Most data that exists in text form, including server logs and Facebook posts, is unstructured
- Scientific observations, as recorded by scientists, are kept in a very neat and organized (structured) format
- A genetic sequence of chemical nucleotides (for example, ACGTATTGCA) is unstructured, even if the order of the nucleotides matters, as we cannot form descriptors of the sequence using a row/column format without taking a further look

Structured data is generally thought of as being much easier to work with and analyze. Most statistical and machine learning models were built with structured data in mind and cannot work on the loose interpretation of unstructured data. The natural row and column structure is easy to digest for human and machine eyes. So, why even talk about unstructured data? Because it is so common! Most estimates place unstructured data as 80-90% of the world's data. This data exists in many forms and, for the most part, goes unnoticed by humans as a potential source of data. Tweets, emails, literature, and server logs are generally unstructured forms of data.

While a data scientist likely prefers structured data, they must be able to deal with the world's massive amounts of unstructured data. If 90% of the world's data is unstructured, that implies that about 90% of the world's information is trapped in a difficult format.

So, with most of our data existing in this free-form format, we must turn to pre-analysis techniques, called **pre-processing**, to apply structure to at least a part of the data for further analysis. A later chapter will deal with pre-processing in great detail; for now, we will consider the part of pre-processing wherein we attempt to apply transformations to convert unstructured data into a structured counterpart. We will see several examples of this later in this book.

In between the realms of structured and unstructured data lies a hybrid category known as **semi-structured data**. While structured data follows a strict schema with a defined row and column format, and unstructured data lacks any specific format, semi-structured data contains elements of both.

Semi-structured data is a form of data that does not conform entirely to the formal structure of data models associated with relational databases or other forms of data tables, yet contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Examples of semi-structured data include XML and JSON files, emails, and some types of NoSQL databases. This type of data is common in web data and certain types of scientific and health research.

Quantitative versus qualitative data

When you ask a data scientist, *what type of data is this?* they will usually assume that you are asking them whether or not it is mostly quantitative or qualitative. It is likely the most common way of describing the specific characteristics of a dataset.

For the most part, when talking about quantitative data, you are *usually* (not always) talking about a structured dataset with a strict row/column structure (because we don't assume unstructured data even *has* any characteristics). That's all the more reason why the pre-processing step is so important.

These two data types can be defined as follows:

- **Quantitative data:** This data can be described using numbers, and basic mathematical procedures, including addition, are possible on the set.
- **Qualitative data:** This data cannot be described using numbers and basic mathematics. This data is generally thought of as being described using **natural categories** and language.

Let's take a look at an example of qualitative and quantitative data in a small business.

Example – coffee shop data

Say that we were processing the customers of a local coffee shop in a major city using five descriptors (characteristics) for each customer:

- Name of a coffee shop
- Revenue (in thousands of dollars)
- Zip code
- Average monthly customers
- Country of coffee origin

Each of these characteristics can be classified as either quantitative or qualitative, and that simple distinction can change everything. Let's take a look at each one:

- *Name of a coffee shop:* Qualitative

The name of a coffee shop is not expressed as a number and we cannot perform mathematical operations on the name of the shop.

- *Revenue:* Quantitative

How much money a coffee shop brings in can be described using a number. Also, we can do basic operations, such as adding up the revenue for 12 months to get a year's worth of revenue.

- *Zip code:* Qualitative

This one is tricky. A zip code is always represented using numbers, but what makes it qualitative is that it does not fit the second part of the definition of quantitative – we cannot perform basic mathematical operations on a zip code. If we add together two zip codes, it is a nonsensical measurement. We don't necessarily get a new zip code and we don't get "double the zip code."

- *Average monthly customers*: Quantitative

Again, describing this factor using numbers and addition makes sense. Add up all of your monthly customers and you get your yearly customers.

- *Country of coffee origin*: Qualitative

We will assume this is a very small coffee shop with coffee from a single origin. This country is described using a name (Ethiopian, Colombian), not numbers.

NOTE

Even though a zip code is described using numbers, it is not quantitative. This is because you can't talk about the sum of all zip codes or an average zip code. These are nonsensical descriptions.

If you are having trouble identifying which is which, when you're trying to decide whether or not the data is qualitative or quantitative, ask yourself a few basic questions about the data characteristics:

- Can you describe the value with numbers?
 - No? It is most likely **qualitative**.
 - Yes? Move on to the next question.
- Do the values make numerical sense if you add them together?
 - No? They are most likely **qualitative**.
 - Yes? You probably have **quantitative** data on your hands.

This method will help you to classify most, if not all, data into one of these two categories. If you are wondering how qualitative data can be described with numbers, imagine survey results asking people to rank something on a scale from 1 to 5. While the contents are being described with numbers, it doesn't make sense to "add" them together. Someone's score of 1 plus someone else's score of 3 doesn't make a score of 4.

The difference between these two categories defines the types of questions you may ask about each column. For a quantitative column, you may ask questions such as the following:

- What is the average value?
- Does this quantity increase or decrease over time (if time is a factor)?
- Is there a threshold where if this number became too high or too low, it would signal trouble for the company?

For a qualitative column, none of the preceding questions can be answered. However, the following questions *only* apply to qualitative values:

- Which value occurs the most and the least?
- How many unique values are there?
- What are these unique values?

Example – inspecting world alcohol consumption data

The **World Health Organization (WHO)** released a dataset describing the average drinking habits of people in countries across the world. We will use Python and the data exploration tool **pandas** to gain a better look:

```
import pandas as pd
read in the CSV file from a URL drinks =
pd.read_csv('https://raw.githubusercontent.com/sinanuozdemir/principles_of_
data_science/master/data/chapter_2/drinks.csv')
examine the data's first five rows
drinks.head()# print the first 5 rows
```

The preceding code block produces this DataFrame:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	AS
1	Albania	89	132	54	4.9	EU
2	Algeria	25	0	14	0.7	AF
3	Andorra	245	138	312	12.4	EU
4	Angola	217	57	45	5.9	AF

Figure 2.1 – The first five rows from our WHO alcohol consumption data

These three lines do the following:

- Import **pandas**, which will be referred to as **pd** in the future
- Read in a **comma-separated value (CSV)** file as a variable called **drinks**
- Call a method, **head**, that reveals the first five rows of the dataset

The table in the preceding figure lists the first five rows of data from the **drinks.csv** file. We have six different columns that we are working on within this example:

- **country**: Qualitative
- **beer_servings**: Quantitative
- **spirit_servings**: Quantitative
- **wine_servings**: Quantitative
- **total_litres_of_pure_alcohol**: Quantitative
- **continent**: Qualitative

Let's look at the qualitative column, **continent**. We can use **pandas** to get some basic summary statistics about this non-numerical characteristic. The **describe()** method is being used here, which first identifies whether the column is likely to be quantitative or qualitative, and then gives basic information about the column as a whole. This can be done as follows:

```
drinks['continent'].describe()
>> count
170
>> unique
5
>>
top
AF
>>
freq
53
```

The preceding code reveals that WHO has gathered data about five unique continents, the most frequent being **AF** (Africa), which occurred 53 times in the 193 observations.

If we take a look at one of the quantitative columns and call the same method, we can see the difference in output, as shown here:

```
drinks['beer_servings'].describe()
count
193.000000
mean
106.160622
std
101.143103
min
0.000000
25%
20.000000
50%
76.000000
75%
188.000000
max
376.000000
```

Now, we can look at the mean (average) beer serving per person per country (106.2 servings), as well as the lowest beer serving, 0, and the highest beer serving recorded, 376 (that's more than a beer a day).

Digging deeper

Quantitative data can be broken down into **discrete** and **continuous** quantities.

These can be defined as follows:

- **Discrete data:** This describes data that is counted. It can only take on certain values.

Examples of discrete quantitative data include a dice roll, since it can only take on six values, and the number of customers in a coffee shop, because you can't have a real range of people.

- **Continuous data:** This describes data that is measured. It exists on an infinite range of values.

A good example of continuous data would be a person's weight because it can be 150 pounds or 197.66 pounds (note the decimals). The height of a person or building is a continuous number

because an infinite scale of decimals is possible. Other examples of continuous data would be time and temperature.

Data as a whole can either be structured or unstructured, meaning that the data can either take on an organized row/column structure with distinct features that describe each row of the dataset or exist in a free-form state that usually must be pre-processed into a form that is easily digestible.

If data is structured, we can look at each column (feature) of the dataset as being either quantitative or qualitative. Basically, can the column be described using mathematics and numbers or not? The next part of this chapter will break down data into four very specific and detailed levels. At each order, we will apply more complicated rules of mathematics, and, in turn, gain a more intuitive and quantifiable understanding of the data.

The four levels of data

It is generally understood that a specific characteristic (feature/column) of structured data can be broken down into one of four levels of data. These levels are as follows:

- The nominal level
- The ordinal level
- The interval level
- The ratio level

As we move down the list, we gain more structure and, therefore, more returns from our analysis. Each level comes with its own accepted practice in measuring the center of the data. We usually think of the mean/average as being an acceptable form of center.

However, this is only true for a specific type of data.

The nominal level

The first level of data, the **nominal** level, consists of data that is described purely by name or category. Basic examples include gender, nationality, species, or yeast strain in a beer. They are not described by numbers and are therefore qualitative. The following are some examples:

- A type of animal is on the nominal level of data. We may also say that if you are a chimpanzee, then you belong to the mammalian class as well.
- A part of speech is also considered at the nominal level of data. The word she is a pronoun, and it is also a noun.

Of course, being qualitative, we cannot perform any quantitative mathematical operations, such as addition or division. These would not make any sense.

Mathematical operations allowed at the nominal level

We cannot perform mathematics at the nominal level of data except the basic equality and set membership functions, as shown in the following two examples:

- Being a tech entrepreneur is the same as being in the tech industry, but not the other way around
- A figure described as a square falls under the description of being a rectangle, but not the other way around

Now, we'll cover measures of center.

Measures of center

A **measure of center** is a number that describes what the data tends to. It is sometimes referred to as the *balance point* of the data. Common examples include the mean, median, and mode.

To find the **center** of nominal data, we generally turn to the **mode** (the most common element) of the dataset. For example, let's look back at the WHO alcohol consumption data. The most common continent surveyed was Africa, making that a possible choice for the center of the **continent** column.

Measures of center, such as the mean and median, do not make sense at this level as we cannot order the observations or even add them together.

What data is like at the nominal level

Data at the nominal level is mostly categorical. Because we can generally only use words to describe the data, it can be lost in translation between countries, or can even be misspelled.

While data at this level can certainly be useful, we must be careful about what insights we may draw from them. With only the mode as a basic measure of center, we are unable to draw conclusions about an **average** observation. This concept does not exist at this level. It is only at the next level that we may begin to perform true mathematics on our observations.

The ordinal level

The nominal level did not provide us with much flexibility in terms of mathematical operations due to one seemingly unimportant fact: we could not order the observations in any natural way. Data at the **ordinal** level provides us with a rank order or the means to place one observation before the other. However, it does not provide us with relative differences between observations, meaning that while we may order the observations from first to last, we cannot add or subtract them to get any real meaning.

Examples

Likert is among the most common ordinal-level scales. Whenever you are given a survey asking you to rate your satisfaction on a scale from 1 to 10, you are providing data at the ordinal level. Your answer, which must fall between 1 and 10, can be ordered: 8 is better than 7 while 3 is worse than 9.

However, the differences between the numbers do not make much sense. The difference between a 7 and a 6 might be different from the difference between a 2 and a 1.

Mathematical operations allowed at the ordinal level

We are allowed much more freedom at this level in terms of mathematical operations. We inherit all mathematics from the ordinal level (equality and set membership) and we can also add the following to the list of operations allowed at the nominal level:

- Ordering
- Comparison

Ordering refers to the natural order provided to us by the data. However, this can be tricky to figure out sometimes. When speaking about the spectrum of visible light, we can refer to the names of colors – **Red, Orange, Yellow, Green, Blue, Indigo, and Violet**. Naturally, as we move from left to right, the light is gaining energy and other properties. We may refer to this as a natural order:

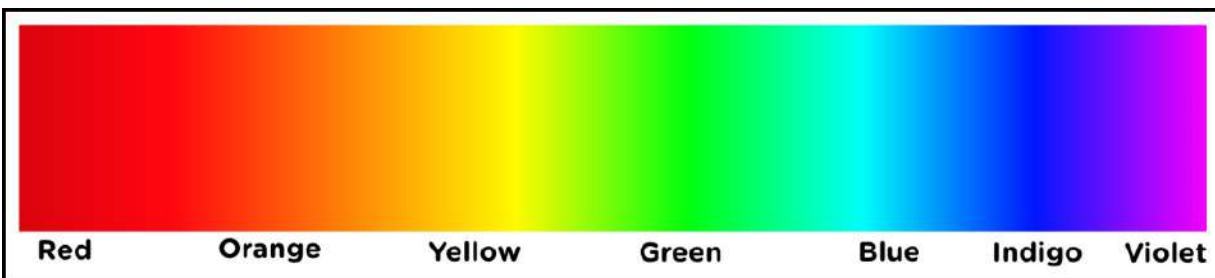


Figure 2.2 – The natural order of color

However, if needed, an artist may impose another order on the data, such as sorting the colors based on the cost of the material to make the said color. This could change the order of the data, but so long as we are consistent in what defines the order, it does not matter what defines it.

Comparisons are another new operation allowed at this level. At the ordinal level, it would not make sense to say that one country was naturally better than another or that one part of speech is worse than another. At the ordinal level, we can make these comparisons. For example, we can talk about how putting a 7 on a survey is worse than putting a 10.

Measures of center

At the ordinal level, the **median** is usually an appropriate way of defining the center of the data. The mean, however, would be impossible because division is not allowed at this level. We can also use the mode, similar to how we could at the nominal level.

Let's look at an example of using the median.

Imagine you have conducted a survey among your employees asking “*How happy are you to be working here on a scale from 1-5?*” and your results are as follows:

```
5, 4, 3, 4, 5, 3, 2, 5, 3, 2, 1, 4, 5, 3, 4, 4, 5, 4, 2, 1, 4, 5, 4, 3,
2, 4, 4, 5, 4, 3, 2, 1]
```

Let's use Python to find the median of this data. It is worth noting that most people would argue that the mean of these scores would work just fine. The reason that the mean would not be as mathematically viable is that if we subtract/add two scores, say a score of 4 minus a score of 2, the difference of 2 does not mean anything. If addition/subtraction among the scores doesn't make sense, the mean won't make sense either:

```
import numpy
results = [5, 4, 3, 4, 5, 3, 2, 5, 3, 2, 1, 4, 5, 3, 4, 4, 5, 4, 2, 1, 4, 5, 4, 3, 2,
4, 4, 5, 4, 3, 2, 1]
sorted_results = sorted(results)
print(sorted_results)
'''
[1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
5, 5, 5, 5, 5, 5, 5]
''' # The ''' (triple apostrophe) denotes a longer (over two lines) comment
print(numpy.mean(results)) # == 3.4375
print(numpy.median(results)) # == 4.0
```

IMPORTANT NOTE

It turns out that the median is not only more sound but also makes the survey results look much better.

The interval level

Now, we are getting somewhere interesting. At the **interval** level, we are beginning to look at data that can be expressed through very quantifiable means, and where much more complicated mathematical formulas are allowed. The basic difference between the ordinal level and the interval level is, well, just that difference.

Data at the interval level allows meaningful subtraction between data points.

Examples of data at the interval level

Temperature serves as an excellent illustration of interval-level data. When comparing a 100-degree Fahrenheit reading in Texas, US, to an 80-degree Fahrenheit measurement in Istanbul, Turkey, it becomes evident that Texas is 20 degrees warmer than Istanbul. This straightforward comparison highlights the increased potential for manipulation and analysis offered by interval-level data in contrast to other levels of measurement.

It seems as though the example at the ordinal level (using the 1 to 5 survey) fits the bill of the interval level. However, remember that the *difference* between the scores (when you subtract them) does not make sense; therefore, this data cannot be called at the interval level.

Mathematical operations allowed at the interval level

We can use all the operations allowed at the lower levels (ordering, comparisons, and so on), along with two other notable operations:

- Addition
- Subtraction

The allowance of these two operations allows us to talk about data at this level in a whole new way.

Measures of center

At this level, we can use the median and mode to describe this data. However, usually, the most accurate description of the center of data would be the **arithmetic mean**, more commonly referred to as the **mean**. Recall that the definition of the mean requires us to add all the measurements together. At the previous levels, addition was meaningless. Therefore, the mean would have lost extreme value. It is only at the interval level and above that the arithmetic mean makes sense.

Now, let's look at an example of using the mean.

Suppose we're looking at the temperature of a fridge containing a pharmaceutical company's new vaccine. We measure the temperature every hour with the following data points (in Fahrenheit):

```
31, 32, 32, 31, 28, 29, 31, 38, 32, 31, 30, 29, 30, 31, 26
```

Using Python again, let's find the mean and median of the data:

```
import numpy
temps = [31, 32, 32, 31, 28, 29, 31, 38, 32, 31, 30, 29, 30, 31, 26]
print(numpy.mean(temps))
#
==
30.73
print(numpy.median(temps))
#
==
31.0
```

Note how the mean and median are quite close to each other and both are around 31 degrees. The question *How cold is the fridge?* on average has an answer of about 31. However, the vaccine comes with a warning:

“Do not keep this vaccine at a temperature under 29 degrees.”

Note that the temperature dropped below 29 degrees at least twice, but we ended up assuming that it isn't enough for it to be detrimental.

This is where the measure of variation can help us understand how bad the fridge situation can be.

Measures of variation

This is something new that we have not discussed yet. It is one thing to talk about the center of the data but, in data science, it is also very important to mention how “spread out” the data is. The measures that describe this phenomenon are called **measures of variation**. You have likely heard of *standard*

deviation from your statistics classes. This idea is extremely important and I would like to address it briefly.

A measure of variation (such as the standard deviation) is a number that attempts to describe how spread out the data is.

Along with a measure of center, a measure of variation can almost entirely describe a dataset with only two numbers.

Standard deviation

Arguably, the standard deviation is the most common measure of the variation of data at the interval level and beyond. The standard deviation can be thought of as the “average distance a data point is at from the mean.” While this description is technically and mathematically incorrect, it is a good way to think about it. The formula for standard deviation can be broken down into the following steps:

1. Find the mean of the data.
2. For each number in the dataset, subtract it from the mean and then square it.
3. Find the average of each square difference.
4. Take the square root of the number obtained in *Step 3* – this is the standard deviation.

Notice how we calculate the arithmetic mean as one of the steps.

For example, let’s look back at the temperature dataset. Let’s find the standard deviation of the dataset using Python:

```
import numpy
temps = [31, 32, 32, 31, 28, 29, 31, 38, 32, 31, 30, 29, 30, 31, 26]
mean = numpy.mean(temps)
# == 30.73
squared_differences = []
# empty list o squared differences
for temperature in temps:
    difference = temperature - mean
    # how far is the point from the mean
    squared_difference = difference**2
    # square the difference
    squared_differences.append(squared_difference)
# add it to our list
average_squared_difference = numpy.mean(squared_differences)
# This number is also called the "Variance"
standard_deviation = numpy.sqrt(average_squared_difference)
# We did it!
print(standard_deviation) # == 2.5157
```

All of this code led us to find out that the standard deviation of the dataset is around 2.5, meaning that, *on average*, a data point is 2.5 degrees off from the average temperature of around 31 degrees. This means that the temperature could likely dip below 29 degrees again in the near future.

IMPORTANT NOTE

The reason we want the “square difference” between each point and the mean and not the “actual difference” is because squaring the value emphasizes outliers – data points that are abnormally far away.

Measures of variation give us a very clear picture of how spread out or dispersed our data is. This is especially important when we are concerned with ranges of data and how data can fluctuate (think percentage return on stocks).

The big difference between data at this level and the next level lies in something that is not obvious. Data at the interval level does not have a *natural starting point or a natural 0*. However, being at 0 degrees Celsius does not mean that you have no temperature

The ratio level

Finally, we will take a look at the ratio level. After moving through three different levels with differing levels of allowed mathematical operations, the ratio level proves to be the strongest of the four.

Mathematical operations allowed at the ratio level

Not only can we define order and difference, but the ratio level allows us to *multiply and divide* as well. This might seem like not much to make a fuss over but it changes almost everything about the way we view data at this level.

Examples

While Fahrenheit and Celsius are stuck at the interval level, the Kelvin scale of temperature boasts a natural zero. A measurement of 0 Kelvin means the absence of heat. It is a non-arbitrary starting zero. We can scientifically say that 200 Kelvin is twice as much heat as 100 Kelvin.

Money in the bank is at the ratio level. You can have no money in the bank and it makes sense that \$200,000 is twice as much as \$100,000.

Many people may argue that Celsius and Fahrenheit also have a starting point (mainly because we can convert from Kelvin to either of the two). The real difference here might seem silly, but because the conversion to Celsius and Fahrenheit makes the calculations go into the negative, it does not define a clear and “natural” zero.

Measures of center

The arithmetic mean still holds meaning at this level, as does a new type of mean called the **geometric mean**, which is the square root of the product of all the values.

For example, in our fridge temperature data, we can calculate the geometric mean as follows:

```
import numpy
temps = [31, 32, 32, 31, 28, 29, 31, 38, 32, 31, 30, 29, 30, 31, 26]
num_items = len(temps)
product = 1.
for temperature in temps:
    product *= temperature
```

```
geometric_mean = product**(1./num_items)
print(geometric_mean) # == 30.634
```

Note again how it is close to the arithmetic mean and median, as calculated previously. This is not always the case and will be talked about at great length in. [Chapter 8](#), *Advanced statistics*.

Problems with the ratio level

Even with all of this added functionality at this level, we must generally also make a very large assumption that makes the ratio level a bit restrictive. For this reason alone, many data scientists prefer the interval level to the ratio level. The reason for this restrictive property is that if we allowed negative values, the ratio might not always make sense.

Consider that we allowed debt to occur in our money in the bank example. If we had a balance of \$50,000, the following ratio would not make sense at all:

$$50,000 / -50,000 = -1$$

Data is in the eye of the beholder

It is possible to impose structure on data. For example, while I said that you technically cannot use a mean for the 1 to 5 data at the ordinal scale, many statisticians would not have a problem using this number as a descriptor of the dataset.

The level at which you are interpreting data is a *massive* assumption that should be made at the beginning of any analysis and done with much care and deliberation. If you are looking at data that is generally thought of at the ordinal level and applying tools such as the arithmetic mean and standard deviation, this is something that data scientists must be aware of. This is mainly because if you continue to hold these assumptions as valid in your analysis, you may encounter problems. For example, if you also assume divisibility at the ordinal level by mistake, you are imposing a structure where the structure may not exist.

We've seen a lot of data about, well, data. Let's wrap up with a summary of what we've learned.

Summary

This chapter has provided an overview of the crucial role data types play in data science, emphasizing the importance of understanding the nature of the data before commencing any analysis. We discussed the significance of asking three key questions when encountering a new dataset: whether the data is structured or unstructured, whether each column is quantitative or qualitative, and the level of data within each column (nominal, ordinal, interval, or ratio).

By completing this chapter, you should be able to identify the types of data they are working with and understand the implications of those data types on their analysis. This knowledge will help you select

appropriate graphs, interpret results, and determine the next steps in the analytical process. You should also be familiar with the concept of converting data from one level to another to gain more insights.

With this knowledge, and with the ability to classify data as nominal or ordinal through various examples, we can begin to approach data challenges more effectively and make informed decisions in data-driven projects.

In the upcoming chapter, we will delve into how data types are utilized by data scientists in the process of data discovery and visualization, further expanding on the practical applications of the concepts discussed in this chapter.

Questions and answers

For the following statements, classify them as ordinal or nominal:

- The origin of the beans in your cup of coffee: *Nominal*
- The place someone receives after completing a foot race: *Ordinal*
- The metal used to make the medal that they receive after placing in the race: *Nominal*
- The telephone number of a client: *Nominal*
- How many cups of coffee you drink in a day: *Ordinal*

The Five Steps of Data Science

This chapter will dive into the five core steps involved in the data science process, with examples every step of the way. These five steps include defining a real problem, collecting and preprocessing the data, exploring and analyzing the data, drawing conclusions, and communicating results effectively.

We will also delve into the important topics of data exploration and data visualization. Data exploration involves examining the characteristics and patterns in your data to better understand it, while data visualization involves using graphs, charts, and other visual aids to represent and communicate your data and findings.

By the end of this chapter, you will have a solid understanding of the data science process and how to apply it to solve real-world problems. So, let's get started!

We will also cover the following topics in this chapter:

- An introduction to what data science really is
- Exploring data effectively
- Exploration tips for all levels of data
- Using **pandas** to manipulate and optimize data

Introduction to data science

A question I've gotten at least once a month for the past decade is *What's the difference between data science and data analytics?* One could argue that there is no difference between the two; others will argue that there are hundreds of differences! I believe that, regardless of how many differences there are between the two terms, the following applies:

Data science follows a structured, step-by-step process that, when followed, preserves the integrity of the results and leads to a deeper understanding of the data and the environment the data comes from.

As with any other scientific endeavor, this process must be adhered to, or else the analysis and the results are in danger of scrutiny. On a simpler level, following a strict process can make it much easier for any data scientist, hobbyist, or professional to obtain results faster than if they were exploring data with no clear vision.

While these steps are a guiding lesson for amateur analysts, they also provide the foundation for all data scientists, even those in the highest levels of business and academia. Every data scientist recognizes the

value of these steps and follows them in some way or another.

Overview of the five steps

The process of data science involves a series of steps that are essential for effectively extracting insights and knowledge from data. These steps are presented as follows:

1. **Asking an interesting question:** The first step in any data science project is to identify a question or challenge that you want to address with your analysis. This involves finding a topic that is relevant, important, and that can be addressed with data.
2. **Obtaining the data:** Once you have identified your question, the next step is to collect the data that you will need to answer it. This can involve sourcing data from a variety of sources, such as databases, online platforms, or through data scraping or data collection methods.
3. **Exploring the data:** After you have collected your data, the next step is to explore it and get a better understanding of its characteristics and patterns. This might involve examining summary statistics, visualizing the data, or applying statistical or **machine learning (ML)** techniques to identify trends or relationships.
4. **Modeling the data:** Once you have explored your data, the next step is to build models that can be used to make predictions or inform decision-making. This might involve applying ML algorithms, building statistical models, or using other techniques to find patterns in the data.
5. **Communicating and visualizing the results:** Finally, it's important to communicate your findings to others in a clear and effective way. This might involve creating reports, presentations, or visualizations that help to explain your results and their implications.

By following these five essential steps, you can effectively use data science to solve real-world problems and extract valuable insights from data.

It's important to note that different data scientists may have different approaches to the data science process, and the steps outlined previously are just one way of organizing the process. Some data scientists might group the steps differently or include additional steps such as feature engineering or model evaluation.

Despite these differences, most data scientists agree that the steps listed previously are essential to the data science process. Whether they are organized in this specific way or not, these steps are all crucial for effectively using data to solve problems and extract valuable insights. Let's dive into these steps one by one.

Asking an interesting question

This is probably my favorite step. Asking an interesting and relevant question is the first and perhaps most important step in the data science process. It sets the direction and focus of your analysis and determines the data and resources that you will need to collect and analyze.

As an entrepreneur, you are likely accustomed to constantly asking questions and seeking answers. This step can be approached like a brainstorming session, where you write down questions and ideas

regardless of whether or not you think data exists to answer them. This helps to avoid bias and allows you to consider a wide range of possibilities.

It's important to be specific and narrow in your focus when asking your question. This will help you to effectively address the problem and extract valuable insights from your data. It's also important to consider the scope and feasibility of your question, as well as the resources and data that you will need to answer it.

By asking an interesting and relevant question, you can set the foundation for a successful data science project and begin the journey of extracting valuable insights from data.

Obtaining the data

Obtaining the data is a crucial step in the data science process. It involves sourcing and collecting the data that you will need to answer the question or solve the problem you have identified. The data can come from a variety of sources, including databases, online platforms, research studies, or data scraping or data collection methods.

This step can be very creative as you will need to think creatively about where to find the data that is most relevant to your question. You may need to explore different sources and platforms, and you may need to use a variety of data collection methods to gather the data you need.

It's important to be mindful of the quality of the data you are collecting, as well as any potential biases or limitations that may be present in the data. It's also important to consider ethical and legal considerations, such as obtaining proper consent and protecting sensitive or confidential data.

Once you have collected your data, it's essential to clean and preprocess it so that it is in a usable format. This can involve removing missing or inaccurate data, formatting the data in a way that makes it easier to work with, and ensuring that the data is consistent and accurate.

By effectively collecting and preprocessing your data, you can set the stage for a successful data science project and be well prepared to move on to the next steps of exploring and analyzing the data.

Exploring the data

Exploring the data is an essential step in the data science process as it involves examining the characteristics and patterns in your data to gain a better understanding of it. This step is crucial for identifying trends, relationships, and insights that can inform your analysis and answer your research question.

There are many different ways to explore data, including visualizing it using graphs, charts, and plots, as well as applying statistical and ML techniques to identify patterns and relationships. It's important to be mindful of the types of data you are working with, as different types of data may require different approaches to exploration.

This step can be time-consuming as it may involve spending several hours learning about the domain and using code or other tools to manipulate and explore the data. By the time this step is completed, the analyst should have a good understanding of the potential insights that the data might contain and be able to form hypotheses about what the data might be trying to tell them.

Exploring the data is a crucial step in the data science process as it helps to inform the direction of your analysis and guide your choice of modeling and analysis techniques. It's important to be thorough and meticulous in this step as the insights you gain can have significant implications for your results and conclusions.

Modeling the data

Modeling the data is an important step in the data science process as it involves using statistical and ML techniques to build models that can be used to make predictions or inform decision-making. This step can be complex as it involves fitting and choosing the appropriate models for your data and objectives, as well as implementing mathematical validation metrics to quantify the effectiveness of the models.

Many different types of models can be used in data science, including linear regression models, logistic regression models, decision tree models, and **neural network (NN)** models, to name a few. It's important to choose the model that is most appropriate for your data and the question you are trying to answer.

Once you have chosen your model, you will need to fit it to your data. This involves using statistical or ML algorithms to find the parameters that best fit the patterns in your data. You will also need to evaluate the performance of your model using validation metrics such as accuracy, precision, and recall. These metrics can help you to understand the effectiveness of your model and identify any areas for improvement.

By effectively modeling your data, you can gain insights and make informed decisions based on your analysis. It's important to carefully consider the choices you make at this step, as the quality of your model can greatly affect the accuracy and usefulness of your results.

Communicating and visualizing the results

This is arguably the most important step. Communicating and visualizing results involves effectively sharing your findings and insights with others. This step can be challenging as it requires you to clearly and concisely convey your results in a way that is understandable and digestible to your audience.

There are many different ways to communicate and visualize your results, including creating reports, presentations, and visualizations such as graphs, charts, and plots. It's important to choose the method that is most appropriate for your audience and the message you are trying to convey.

It's also important to consider the impact of your results and the implications of your findings. This can involve discussing the limitations of your analysis, the implications of your results, and any

recommendations or next steps that may be warranted.

In this book, we will focus mainly on *steps 3, 4, and 5* of the data science process, which involve exploring and analyzing the data, modeling the data, and communicating and visualizing the results. While the first two steps of asking an interesting question and obtaining the data are also essential to the process, we will only touch upon these steps briefly and focus mainly on the more scientific aspects of the process. By focusing on these steps, we aim to provide interesting questions and datasets that can be used to explore and analyze data.

WHY ARE WE SKIPPING STEPS 1 AND 2 IN THIS BOOK?

While the first two steps are undoubtedly imperative to the process, they generally precede statistical and programmatic systems. Later in this book, we will touch upon the different ways to obtain data; however, to focus on the more scientific aspects of the process, we will begin with exploration right away.

Let's focus on the third step a bit more – exploring our data.

Exploring the data

The process of exploring data is not always straightforward and can involve a variety of approaches and techniques. Some common tasks that are involved in data exploration include recognizing different types of data, transforming data types, and using code to systematically improve the quality of the entire dataset. These tasks can be accomplished using tools such as the `pandas` Python package, which is commonly used for data manipulation and analysis.

There are a few basic questions that you should consider when exploring a new dataset. These questions can help you to get a sense of the data and guide your analysis. The three basic questions are presented here:

- What are the types of data that are present in the dataset?
- What are the characteristics and patterns of the data?
- How is the data organized, and what transformations might be necessary to make it more usable?

By answering these questions and exploring your data thoroughly, you can gain a deeper understanding of the data and be well prepared to move on to the next steps of modeling and analyzing the data.

Guiding questions for data exploration

When given a new dataset, whether it is familiar to you or not, it is important to use the following questions as guidelines for your preliminary analysis:

- **Is the data structured or not?:** We are checking whether or not the data is presented in a row/column structure. For the most part, data will be presented in a structured fashion. In this book, over 90% of our examples will begin with structured data. Nevertheless, this is the most basic question that we can answer before diving any deeper into our analysis. A general rule of

thumb is that if we have unstructured data, we want to transform it into a row/column structure. For example, earlier in this book, we looked at ways to transform text into a row/column structure by counting the number of words/phrases.

- **What does each row represent?:** Once we have an answer to how the data is organized and are looking at a nice row/column-based dataset, we should identify what each row actually represents. This step is usually very quick and can help put things into perspective much more quickly.
- **What does each column represent?:** We should identify each column by the level of data, whether or not it is quantitative/qualitative, and so on. This categorization might change as our analysis progresses, but it is important to begin this step as early as possible.
- **Are there any missing data points?:** Data isn't perfect. Sometimes, we might be missing data because of human or mechanical error. When this happens, we, as data scientists, must make decisions about how to deal with these discrepancies.
- **Do we need to perform any transformations on the columns?:** Depending on the level/type of data in each column, we might need to perform certain types of transformation. For example, generally speaking, for the sake of statistical modeling and ML, we would like each column to be numerical, so would use Python to make any transformations.

All the while, we are asking ourselves the overall question *What can we infer from the preliminary inferential statistics?* We want to be able to understand our data better than when we first found it.

Enough talk; let's see a hands-on example.

Dataset 1 – Yelp

The first dataset we will look at is a public dataset made available by the restaurant review site *Yelp*. All **personally identifiable information (PII)** has been removed. I am purposefully not giving much information because part of our goal is to ascertain elements about this data for ourselves. If I told you more, where would be the fun in that?

Let's say we were just given this data. The first thing we have to do is read it in, as shown here:

```
import pandas as pd
yelp_raw_data = pd.read_csv("yelp.csv")
yelp_raw_data.head()
```

Here's a quick recap of what the preceding code does:

1. It imports the **pandas** package and nicknames it **pd**.
2. It reads in the **.csv** file from the web and calls it **yelp_raw_data/**.
3. It looks at the head of the data (just the first few rows).

We get the following output:

	business_id	date	review_id	stars	text	type	user_id	cool	useful	funny
0	9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	My wife took me here on my birthday for breakf...	review	rLtI8ZkDX5vH5nAx9C3q5Q	2	5	0
1	ZRJwVLyzEJq1VAIhDhYlow	2011-07-27	IjZ33sJrzXqU-0X6U8NwyA	5	I have no idea why some people give bad review...	review	0a2KyEL0d3Yb1V6aivbluQ	0	0	0
2	6oRAC4uyJCsj11X0WZpVSA	2012-06-14	IESLBzqUCldSzSqm0eCSxQ	4	love the gyro plate. Rice is so good and I als...	review	0ht2KtfLiobPvh6cDC8JQg	0	1	0
3	_1QQZu14zZOyFCvXc0o6Vg	2010-05-27	G-WvGalSbqqaMHINnByodA	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!!...	review	uZeti9T0NcROGOyFfughhg	1	2	0
4	6ozycU1RpktNG2-1BroVtw	2012-01-05	1uJFq2r5QfJG_6ExMRCaGw	5	General Manager Scott Petello is a good egg!!!!...	review	vYmM4KTsC8ZfQBg-j5MWkw	0	0	0

Figure 3.1 – The first five rows (the head) of the pandas DataFrame of our Yelp dataset reveals a mix of data levels, including nominal (for example, text, business_id) and ordinal (cool, useful, and funny)

From here, we can begin to ask some key questions. Let's start with this one: *Is the data structured or not?* Because we have a nice row/column structure, we can conclude that this data seems to be structured.

So, then, the next logical question would be *What does each row represent?* It seems pretty obvious that each row represents a user giving a review of a business. The next thing we should do is examine each row and label it by the type of data it contains. At this point, we can also use Python to figure out just how big our dataset is. We can use the shape quality of a DataFrame to find this out, as shown:

```
yelp_raw_data.shape
# (10000,10)
```

It tells us that this dataset has 10000 rows and 10 columns. Another way to say this is that this dataset has 10,000 observations and 10 characteristics.

OK—so, then, the next question: *What does each column represent?* (Note that we have 10 columns.) Let's have a look at the answers:

- **business_id**: This is likely to be a unique identifier for the business the review is for. This would be at the **nominal level** because there is no natural order to this identifier.
- **date**: This is probably the date on which the review was posted. Note that it seems to be only specific to the day, month, and year. Even though time is usually considered continuous, this column would likely be considered discrete and at the **ordinal level** because of the natural order that dates have.
- **review_id**: This is likely to be a unique identifier for the review that each post represents. This would be at the nominal level because, again, there is no natural order to this identifier.
- **stars**: From a quick look (don't worry; we will perform some further analysis soon), we can see that this is an ordered column that represents what the reviewer gave the restaurant as a final score. This is ordered and qualitative, so is at the ordinal level.
- **text**: This is probably the raw text that each reviewer wrote. As with most text, we place this at the nominal level.

- **type**: In the first five rows, all we see is the word **review**. This might be a column that identifies that each row is a review, implying that there might be another type of row other than a review. We will take a look at this later. We place this at the nominal level.
- **user_id**: This is likely to be a unique identifier for the user who is writing the review. Just as with the other unique identifiers, we place this data at the nominal level.

Our next question is *Are there any missing data points?* Perform an `isnull` operation. For example, if your DataFrame is called `awesome_dataframe`, then try the `awesome_dataframe.isnull().sum()` Python command, which will show the number of missing values in each column.

Our question after that would be *Do we need to perform any transformations on the columns?* At this point, we are looking for a few things. For example, will we need to change the scale of some of the quantitative data, or do we need to create dummy variables for the qualitative variables? As this dataset only has qualitative columns, we can only focus on transformations at the ordinal and nominal scales.

Before starting, let's go over some quick terminology for `pandas`, the Python data exploration module.

DataFrames

When we read in a dataset, `pandas` creates a custom object called `DataFrame`. Think of this as the Python version of a spreadsheet (but way better). In this case, the `yelp_raw_data` variable is a `DataFrame`.

To check whether this is true in Python, type in the following code:

```
type(yelp_raw_data)
# pandas.core.frame.DataFrame
```

DataFrames are two-dimensional in nature, meaning that they are organized in a row/column structure just as spreadsheets are. The main benefit of using DataFrames over, say, spreadsheet software is that a `DataFrame` can handle much larger data than most common spreadsheet software. If you are familiar with the R language, you might recognize the word `DataFrame`. This is because the name was actually borrowed from the language!

As most of the data that we will deal with is structured, DataFrames are likely to be the most used object in `pandas`, second only to the `Series` object.

Series

The `series` object is simply a `DataFrame`, but only with one dimension. Essentially, it is a list of data points. Each column of a `DataFrame` is considered to be a `series` object. Let's check this—the first thing we need to do is grab a single column from our `DataFrame`; we generally use what is known as **bracket notation**. The following is an example:

```
yelp_raw_data['business_id'] # grabs a single column of the DataFrame
```

We will list the first and last few rows, as follows:

```
9yKzy9PApeiPPOUJEtnvkg
ZRJwVLyzEJq1VAihDhYiow
6oRAC4uyJCsJl1X0WZpVSA
_1QQZuf4zZOyFCvXc0o6Vg
6ozycU1RpktNG2-1BroVtw
-yxfBYGB6SEqszmxJxd97A
6zp713qNhx8d9KCJJnrw1xA
```

Let's use the `type` function to check that this column is a `Series` object:

```
type(yelp_raw_data['business_id'])
# pandas.core.series.Series
```

The `pandas Series` object will come up time and time again as it is a core component of `DataFrame`.

Exploration tips for qualitative data

Using these two `pandas` objects, let's start performing some preliminary data exploration. For qualitative data, we will specifically look at the nominal and ordinal levels.

Nominal-level columns

As we are at the nominal level, let's recall that at this level, data is qualitative and is described purely by name. In this dataset, this refers to `business_id`, `review_id`, `text`, `type`, and `user_id`. Let's use `pandas` in order to dive a bit deeper, as shown here:

```
yelp_raw_data['business_id'].describe()
# count
10000
# unique
4174
#
top
ntN85eu27C04nwyPa8IHtw
#
freq
37
```

The `describe` function will give us some quick stats about the column whose name we enter into the quotation marks. Note how `pandas` automatically recognized that `business_id` was a qualitative column and gave us stats that make sense. When `describe` is called on a qualitative column, we will always get the following four items:

- **count**: How many values are filled in
- **unique**: How many unique values are filled in
- **top**: The name of the most common item in the dataset
- **freq**: How often the most common item appears in the dataset

At the nominal level, we are usually looking for a few things that would signal a transformation:

- Do we have a reasonable number (usually under 20) of unique items?
- Is this column text?
- Is this column unique across all rows?

So, for the `business_id` column, we have a count of 10,000. Don't be fooled, though! This does not mean that we have 10,000 businesses being reviewed here. It just means that of the 10,000 rows of reviews, the `business_id` column is filled in all 10,000 times. The next qualifier, `unique`, tells us that we have 4,174 unique businesses being reviewed in this dataset. The most reviewed business is the `JokKtdXU7zXHcr20Lrk29A` business, which was reviewed 37 times:

If we run the following code snippet, we will see that we have a totally unique column called `review_id` which is a unique identifier for each review.

```
yelp_raw_data['review_id'].describe()
# count          10000
# unique         10000
#              Top    M3jTv5NIipi_N4mgmZiIEg
#              Freq      1
```

We have a `count` of 10000 and `unique` of 10000. Think for a second—does this make sense? Think about what each row represents and what this column represents.

(Insert Jeopardy theme song here.)

Of course it does! Each row of this dataset is supposed to represent a single, unique review of a business, and this column is meant to serve as a unique identifier for a review. So, it makes sense that the `review_id` column has 10000 unique items in it. So, why is `eTa5KD-LTgQv6UT1ZmiJmw` the *most common* review? This is just a random choice from 10000 and means nothing:

```
yelp_raw_data['text'].describe()
count      10000
unique      9998
top         This review is for the chain in general. The l...
freq        2
```

This column, which represents the actual text people wrote, is interesting. We would imagine that this should also be similar to `review_id` in that each one should contain unique text because it would be weird if two people wrote exactly the same thing, but we have two reviews with the exact same text! Let's take a second to learn about DataFrame filtering to examine this further.

Filtering in pandas

Let's talk a bit about how filtering works. Filtering rows based on certain criteria is quite easy in `pandas`. In a DataFrame, if we wish to filter out rows based on some search criteria, we will need to go row by row and check whether or not a row satisfies that particular condition; `pandas` handles this by passing in a `Series` object of `True` and `False` (Booleans).

We literally pass into the DataFrame a list of **True** and **False** data that mean the following:

- **True**: This row satisfies the condition
- **False**: This row does not satisfy the condition

So, first, let's make the conditions. In the following lines of code, I will grab the text that occurs twice:

```
yelp_raw_data['text'].describe()['top']
```

Here is a snippet of the text:

```
"This review is for the chain in general. The location we went to is new so it isn't  
in Yelp yet. Once it is I will put this review there as well....."
```

Right off the bat, we can guess that this might actually be one person who went to review two businesses that belong to the same chain and wrote the exact same review. However, this is just a guess right now.

IMPORTANT NOTE

The `duplicate_text` variable is of the `string` type.

Now that we have this text, let's use some magic to create that **Series** object of **True** and **False**:

```
duplicate_text = yelp_raw_data['text'].describe()['top']  
text_is_the_duplicate = yelp_raw_data['text'] == duplicate_text
```

Right away, you might be confused. What we have done here is take the `text` column of the DataFrame and compare it to the `duplicate_text` string. This is strange because we seem to be comparing a list of 10,000 elements to a single string. Of course, the answer should be a straight false, right?

Series has a very interesting feature in that if you compare the series to an object, it will return another series of Booleans of the same length where each **True** and **False** instance is the answer to the question *Is this element the same as the element you are comparing it to?* Very handy!

This next code block shows us a preview of the contents of this new Series.

```
type(text_is_the_duplicate) # it is a Series of Trues and Falses  
text_is_the_duplicate.head() # shows a few Falses out of the Series
```

In Python, we can add and subtract **True** and **False** as if they were 1 and 0, respectively—for example, **True + False - True + False + True == 1**. So, we can verify that this **Series** object is correct by adding up all of the values. As only two of these rows should contain the duplicate text, the sum of the **Series** object should only be 2, which it is! This is shown as follows:

```
sum(text_is_the_duplicate) # == 2
```

Now that we have our series of Booleans, we can pass them directly into our DataFrame, using bracket notation, and get our filtered rows, as illustrated:


```
filtered_dataframe = yelp_raw_data[text_is_the_duplicate] # the filtered Dataframe
```

	business_id	date	review_id	stars	text	type	user_id	cool	useful	funny
4372	jvvh4Q00Hq2XyicfmAAT2A	2012-06-16	ivGRamFF3KurE9bjkl6uMw	2	This review is for the chain in general. The I...	review	KLekdmo4FdNnP0huUhzZNw	0	0	0
9680	r1onUa02zMz_ki8eF-Adug	2012-06-16	mutQE6UfjLlpJ6Wozpq5UA	2	This review is for the chain in general. The I...	review	KLekdmo4FdNnP0huUhzZNw	0	0	0

Figure 3.2 – Representing two rows with duplicate text, stars, and survey scores

Data is hardly ever perfect, and a common thing to check for when dealing with raw text is the amount of duplicate fields. In this figure, we can see that there are two rows with duplicate text, stars, and survey scores. This is likely written by the same person given to two locations of a chain.

It seems that our suspicions were correct, and one person, on the same day, gave the exact same review to two different `business_id`s column, presumably a part of the same chain. Let's keep moving along to the rest of our columns:

```
# count          10000
# unique          1
#           top    Review
#           freq   10000
yelp_raw_data['type'].describe()
```

Remember this column? Turns out they are all the exact same type, namely `review`:

```
yelp_raw_data['user_id'].describe()
# count          10000
# unique          6403
#           top    fczQCSmaWF78toLEmb0Zsw
#           freq     38
```

Similar to the `business_id` column, all 10,000 values are filled in with 6,403 unique users and one user reviewing 38 times!

In this example, we won't have to perform any transformations.

Ordinal-level columns

As far as ordinal columns go, we are looking at `date` and `stars`. For each of these columns, let's look at what the `describe` method brings back:

```
yelp_raw_data['stars'].describe()
# count    10000.000000
# mean      3.777500
# std       1.214636
# min       1.000000
# 25%       3.000000
# 50%       4.000000
# 75%       5.000000
# max       5.000000
```

Woah! Even though this column is ordinal, the `describe` method returns stats that we might expect for a quantitative column. This is because the software saw a bunch of numbers and just assumed that we wanted stats such as the mean or the min and max. This is not a problem. Let's use a method called `value_counts` to see the count distribution, as follows:

```
yelp_raw_data['stars'].value_counts()
# 4          3526
# 5          3337
# 3          1461
# 2           927
# 1           749
```

The `value_counts` method will return the distribution of values for any column. In this case, we see that the star rating 4 is the most common, with 3526 values, followed closely by rating 5. We can also plot this data to get a nice visual. First, let's sort by star rating, and then use the prebuilt `plot` method to make a bar chart:

```
import datetime
dates = yelp_raw_data['stars'].value_counts()
dates.sort_values
dates.plot(kind='bar')
```

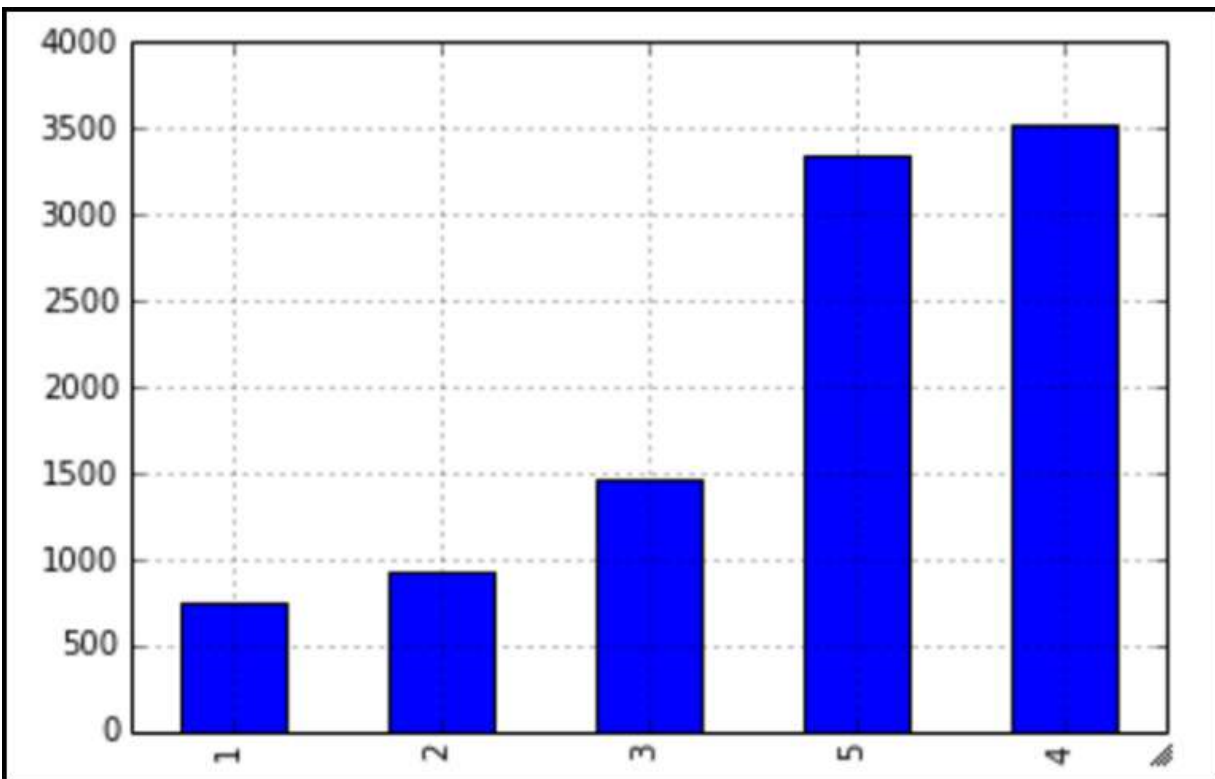


Figure 3.3 – A distribution of the star ratings given reveals that most people are giving 4 or 5 stars, with the most common star rating being 4

From this graph, we can conclude that people are definitely more likely to give good star ratings over bad ones! We can follow this procedure for the `date` column. I will leave you to try it on your own. For

now, let's look at a new dataset.

Dataset 2 – Titanic

The `titanic` dataset is one of those “rites of passage” datasets in data science. Everyone, at some point, has seen this dataset in some tutorial or book. It contains a sample of people who were on the Titanic when it struck an iceberg in 1912. I give you my word that, going forward, we will be using much more interesting datasets, but trust me, it's easier to start off with a dataset such as this one and Yelp to warm up the data science muscles.

Let's go ahead and import our new dataset and output the first five rows using the `head` method:

```
titanic = pd.read_csv('short_titanic.csv')
titanic.head()
```

This table represents the DataFrame for the `short_titanic.csv` dataset. This data is definitely organized in a row/column structure, as is most spreadsheet data. Let's take a quick peek at its size:

```
titanic.shape
# (891, 5)
```

We have **891** rows and **5** columns. Each row seems to represent a single passenger on the ship, and as far as columns are concerned, the following list tells us what they indicate:

- **Survived:** This is a binary variable that indicates whether or not the passenger survived the accident (**1** if they survived, **0** if they died). This is likely to be at the nominal level because there are only two options.
- **Pclass:** This is the class that the passenger was traveling in (**3** for third class, and so on). This is at the ordinal level.
- **Name:** This is the name of the passenger, and it is definitely at the nominal level.
- **Sex:** This indicates the gender of the passenger. It is at the nominal level.
- **Age:** This one is a bit tricky. Arguably, you may place age at either a qualitative or quantitative level; however, I think that age belongs to a quantitative state, and thus, to the ratio level.

As far as transformations are concerned, usually, we want all columns to be numerical, regardless of their qualitative state. This means that `Name` and `Sex` will have to be converted into numerical columns somehow. For `Sex`, we can change the column to hold 1 if the passenger was female and 0 if they were male. Let's use `pandas` to make the change. We will have to import another Python module, called `numpy` or numerical Python, as illustrated:

```
import numpy as np
titanic['Sex'] = np.where(titanic['Sex']=='female', 1, 0)
```

The `np.where` method takes in three things, as follows:

- A list of Booleans (**True** or **False**)
- A new value
- A backup value

The method will replace all `True` instances with the first value (in this case, 1) and the `False` instances with the second value (in this case, 0), leaving us with a new numerical column that represents the same thing as the original `sex` column:

```
titanic['Sex']  
# 0      0  
# 1      1  
# 2      1  
# 3      1  
# 4      0  
# 5      0  
# 6      0  
# 7      0
```

Let's use a shortcut and describe all the columns at once so that we can get a bird's-eye view of what our data looks like and how it is roughly shaped. The code to do this is shown in this code snippet:

```
titanic.describe()
```

	Survived	Pclass	Sex	Age
count	891.000000	891.000000	891.000000	714.000000
mean	0.383838	2.308642	0.352413	29.699118
std	0.486592	0.836071	0.477990	14.526497
min	0.000000	1.000000	0.000000	0.420000
25%	0.000000	2.000000	0.000000	20.125000
50%	0.000000	3.000000	0.000000	28.000000
75%	1.000000	3.000000	1.000000	38.000000
max	1.000000	3.000000	1.000000	80.000000

Figure 3.4 – Descriptive statistics of our Titanic dataset's numerically formatted columns

Reveal that these features have pretty different ranges and means. Note that just because the column is “numerical” doesn't mean it is quantitative. `Pclass` would generally be recommended as qualitative even though it is shown here as integers.

This table lists descriptive statistics of the Titanic dataset. Note how our qualitative columns are being treated as quantitative; however, I'm looking for something irrelevant to the data type. Note the `count` row: **Survived**, **Pclass**, and **Sex** all have 891 values (the number of rows), but **Age** only has 714 values. Some are missing! To double-verify, let's use the `pandas isnull` and `sum` functions, as shown:

```
titanic.isnull().sum()
Survived      0
Pclass        0
Name          0
Sex           0
Age           0
```

This will show us the number of missing values in each column. So, **Age** is the only column with missing values to deal with.

When dealing with missing values, you usually have the following two options:

- Drop the row with the missing value
- Try to fill it in

Dropping the row is the easy choice; however, you run the risk of losing valuable data! For example, in this case, we have 177 missing age values (891-714), which is nearly 20% of the data. To fill in the data, we could either go back to the history books, find each person one by one, and fill in their age, or we could fill in the age with a placeholder value.

Let's fill in each missing value of the **Age** column with the overall average age of the people in the dataset. For this, we will use two new methods, called `mean` and `fillna`. We use `isnull` to tell us which values are `null` and the `mean` function to give us the average value of the **Age** column. The `fillna` method is a `pandas` method that replaces null values with a given value:

```
print sum(titanic['Age'].isnull()) # == 177 missing values
average_age = titanic['Age'].mean() # get the average age
titanic['Age'].fillna(average_age, inplace = True) #use the fillna method to remove null values
print sum(titanic['Age'].isnull()) # == 0 missing values
```

We're done! We have replaced each value with 26.69, the average age in the dataset. The following code now confirms that no null values exist:

```
titanic.isnull().sum()
Survived      0
Pclass        0
Name          0
Sex           0
Age           0
```

Great! Nothing is missing, and we did not have to remove any rows. Let's check back in with our data by looking at `head` again:

```
titanic.head()
```

	Survived	Pclass	Name	Sex	Age
0	0	3	Braund, Mr. Owen Harris	0	22
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38
2	1	3	Heikkinen, Miss. Laina	1	26
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35
4	0	3	Allen, Mr. William Henry	0	35

Figure 3.5 – The first five rows of our Titanic dataset

At this point, we could start getting a bit more complicated with our questions—for example, *What is the average age for a female or a male?* To answer this, we can filter by each gender and take the mean age; `pandas` has a built-in function for this, called `groupby`, as illustrated here:

```
titanic.groupby('Sex')['Age'].mean()
```

This means the following: group the data by the `sex` column, and then give me the mean age for each group. This gives us the following output:

```
Sex
0    30.505824
1    28.216730
```

We will ask more of these difficult and complex questions and will be able to answer them with Python and statistics.

Summary

Exploring the data is only one of the essential steps in the data science process, and it is something that we will continue to do throughout this book as we work with different datasets. By following the steps of data exploration, we can transform, break down, and standardize our data to prepare it for modeling and analysis.

Our five steps serve as a standard practice for data scientists and can be applied to any dataset that requires analysis. While they are only guidelines, they provide a framework for exploring and understanding new data, and they can help us to identify trends, relationships, and insights that can inform our analysis.

As we progress in this book, we will delve into the use of statistical, probabilistic, and ML models to analyze and make predictions from data. Before we can fully delve into these more complex models, however, it is important to review some of the basic mathematics that underlie these techniques. In the next chapter, we will cover some of the math that is necessary to perform some of the more

complicated operations in modeling. Don't worry—the math required for this process is minimal, and we will go through it step by step to ensure that you have a solid foundation. By understanding the underlying math, we can better understand the models and techniques that we will be using, and we can more effectively apply them to our data analysis

Basic Mathematics

As we delve deeper into the realm of data science, it is essential to understand the basic mathematical principles and concepts that are fundamental to the field. While math may often be perceived as intimidating, my goal is to make this learning experience as engaging and enjoyable as possible. In this chapter, we will cover key topics such as basic symbols and terminology, logarithms, and exponents, set theory, calculus, and matrix (linear) algebra. Additionally, we will explore other fields of mathematics and their applications in data science and other scientific endeavors, including the following:

- Basic symbols/terminology
- Logarithms/exponents
- Set theory
- Calculus
- Matrix (linear) algebra

It is important to remember that, as discussed previously, mathematics is one of the three crucial components of data science. The concepts presented in this chapter will not only be useful in later chapters but also in understanding probabilistic and statistical models. These are fundamental building blocks for anyone aspiring to become a data scientist, and as such, should be thoroughly understood.

As a math teacher, it is my duty to educate and enlighten my students on the undeniable importance of mathematics in our daily lives. From the simplest tasks, such as watering plants and feeding pets, to more complex endeavors, mathematical principles and concepts are constantly at play. Even though these calculations and predictions may not always be done consciously, they are still being made by the human brain. It is my goal to help my students understand and appreciate the fundamental role that math plays in our daily lives and to realize the innate mathematical abilities that reside within us all.

Basic symbols and terminology

In the following section, we will review the mathematical concepts of vectors, matrices, arithmetic symbols, and linear algebra, as well as some more subtle notations used by data scientists.

Vectors and matrices

A **vector** is defined as an object with both magnitude and direction. This definition, however, is a bit complicated. For our purpose, a vector is simply a one-dimensional array representing a series of

numbers. Put another way, a vector is a list of numbers.

It is generally represented using an arrow or bold font, as shown here:

\vec{x} or \mathbf{x}

Vectors are broken into components, which are individual members of the vector. We use index notations to denote the element that we are referring to, as illustrated here:

$$\text{If } \vec{x} = \begin{pmatrix} 3 \\ 6 \\ 8 \end{pmatrix} \text{ then } x_1 = 3$$

NOTE

In math, we generally refer to the first element as index 1, as opposed to computer science, where we generally refer to the first element as index 0. It is important to remember which index system you are using.

In Python, we can represent arrays in many ways. We could simply use a Python list to represent the preceding array: $x = [3, 6, 8]$. However, it is better to use the `numpy` array type to represent arrays, as shown here, because it gives us much more utility when performing vector operations:

```
import numpy as np
x = np.array([3, 6, 8])
```

Regardless of the Python representation, vectors give us a simple way of storing *multiple dimensions* of a single data point/observation.

If we measure the average satisfaction rating (0-100) of employees in three departments of a company as being 57 for HR, 89 for engineering, and 94 for management, we can represent this as a vector with the following formula:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 57 \\ 89 \\ 94 \end{pmatrix}$$

This vector holds three different bits of information about our data. This is the perfect use of a vector in data science.

You can also think of a vector as being the theoretical generalization of the pandas `series` object. So, naturally, we need something to represent the DataFrame.

We can extend our notion of an array to move beyond a single dimension and represent data in multiple dimensions.

A **matrix** is a two-dimensional representation of arrays of numbers. **Matrices** (plural of matrix) have two main characteristics that we need to be aware of. The dimension of a matrix, denoted by $n \times m$ (*n by m*), tells us that the matrix has n rows and m columns. Matrices are generally denoted by a capital, bold-faced letter, such as X . Consider the following example:

$$\begin{pmatrix} 3 & 4 \\ 8 & 55 \\ 5 & 9 \end{pmatrix}$$

This is a 3×2 (3 by 2) matrix because it has three rows and two columns.

NOTE

If a matrix has the same number of rows and columns, it is called a **square matrix**.

The matrix is our generalization of the pandas DataFrame. It is arguably one of the most important mathematical objects in our toolkit. It is used to hold organized information – in our case, data.

Revisiting our previous example, let's say we have three offices in different locations, each with the same three departments: HR, engineering, and management. We could make three different vectors, each holding a different office's satisfaction scores, as shown here:

$$x = \begin{pmatrix} 57 \\ 89 \\ 94 \end{pmatrix}, y = \begin{pmatrix} 67 \\ 87 \\ 94 \end{pmatrix}, z = \begin{pmatrix} 65 \\ 98 \\ 60 \end{pmatrix}$$

However, this is not only cumbersome but also unscalable. What if you have 100 different offices? In this case, you would need to have 100 different one-dimensional arrays to hold this information.

This is where a matrix alleviates this problem. Let's make a matrix where each row represents a different department and each column represents a different office, as shown in *Table 4.1*:

	Office 1	Office 2	Office 3
HR	57	67	65
Engineering	89	87	98
Management	94	84	60

Table 4.1 – Some sample data we want to model as a matrix

This is much more natural. Now, let's strip away the labels; we'll be left with a matrix:

$$X = \begin{pmatrix} 57 & 67 & 65 \\ 89 & 87 & 98 \\ 94 & 94 & 60 \end{pmatrix}$$

Quick exercises

The following is a list of quick exercises you can do to understand matrices better:

1. If we added a fourth office, would we need a new row or column?
2. What would the dimension of the matrix be after we added the fourth office?
3. If we eliminate the management department from the original X matrix, what would the dimension of the new matrix be?
4. What is the general formula to find out the number of elements in the matrix?

Answers

Here are the answers:

1. Column
2. 3×4
3. 2×3
4. $n \times m$ (n being the number of rows and m being the number of columns)

Let's move on to arithmetic symbols.

Arithmetic symbols

In this section, we will go over some symbols associated with basic arithmetic that appear in most, if not all, data science tutorials and books.

Summation

The uppercase sigma, Σ , symbol is a universal symbol for addition. Whatever is to the right of the sigma symbol is usually something iterable, meaning that we can go over it one by one (for example, a vector).

For example, let's create the representation of a vector, $X=[1,2,3,4,5]$.

To find the sum of the content, we can use the following formula:

$$\sum x_i = 15$$

In Python, we can use the following formula:

$$\text{sum}(x) \# = 15$$

For example, the formula for calculating the mean of a series of numbers is quite common. If we have a vector (x) of length n , the mean of the vector can be calculated as follows:

$$\text{mean} = 1/n \sum x_i = 15$$

This means that we will add up each element of x , denoted by x_i , and then multiply the sum by $1/n$, otherwise known as dividing by n (the length of the vector).

In Python, we can use the following formula to get the mean of the array, x :

$$\text{mean} = \text{sum}(x) / \text{len}(x) \# = 3$$

Dot product

The dot product is an operator such as addition and multiplication. It is used to combine two vectors, as shown here:

$$\begin{pmatrix} 3 \\ 7 \end{pmatrix} \cdot \begin{pmatrix} 9 \\ 5 \end{pmatrix} = 3 \cdot 9 + 7 \cdot 5 = 62$$

What does this mean? Let's say we have a vector that represents a customer's sentiments toward three genres of movies: comedy, romance, and action.

NOTE

When using a dot product, note that an answer is a single number, known as a **scalar**.

On a scale of 1 to 5, a customer loves comedies, hates romantic movies, and is fine with action movies.

We might represent this as follows:

$$\begin{pmatrix} 5 \\ 1 \\ 3 \end{pmatrix}$$

Here, 5 denotes their love for comedies, 1 denotes their hatred of romantic movies, and 3 denotes the customer's indifference toward action movies.

Now, let's assume that we have two new movies, one of which is a romantic comedy and the other is a funny action movie. The movies would have their own vector of qualities, as shown here:

$$m1 = \begin{pmatrix} 4 \\ 5 \\ 1 \end{pmatrix} \quad m2 = \begin{pmatrix} 5 \\ 1 \\ 5 \end{pmatrix}$$

Here, $m1$ is our romantic comedy and $m2$ is our funny action movie.

To make a recommendation, we must apply the dot product between the customer's preferences for each movie. The higher value will win and, therefore, will be recommended to the user.

Let's compute the recommendation score for each movie. For movie 1, we want to compute the following:

$$\begin{pmatrix} 5 \\ 1 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 5 \\ 1 \end{pmatrix}$$

We can think of this problem as follows:

Customer:	M_1			
$\begin{pmatrix} 5 \\ 1 \\ 3 \end{pmatrix}$	+	$\begin{pmatrix} 4 \\ 5 \\ 1 \end{pmatrix}$	=	$(5,4) \rightarrow$ user loves comedies and this move is funny $(1,5) \rightarrow$ user loves romance but this move is romantic $(3,1) \rightarrow$ user doesn't mind action and the move is not action packed <hr style="width: 20%; margin: 10px auto;"/> 28

Figure 4.1 – How to interpret a dot product

The answer we obtain is 28, but what does this number mean? On what scale is it? Well, the best score anyone can ever get is when all values are 5, making the outcome as follows:

$$\begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix} = 5^2 + 5^2 + 5^2 = 75$$

The lowest possible score is when all values are 1, as shown here:

$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 1^2 + 1^2 + 1^2 = 3$$

So, we must think about 28 on a scale from 3 to 75. The number 28 is closer to 3 than it is to 75. Let's try this for movie 2:

$$\begin{pmatrix} 5 \\ 1 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 1 \\ 5 \end{pmatrix} = \begin{pmatrix} 5 \cdot 5 \end{pmatrix} + \begin{pmatrix} 1 \cdot 1 \end{pmatrix} + \begin{pmatrix} 3 \cdot 5 \end{pmatrix} = 41$$

This is higher than 28! So, between movie 1 and movie 2, we would recommend movie 2 to our user. This is, in essence, how most movie prediction engines work. They build a customer profile, which is represented as a vector. They then take a vector representation of each movie they have to offer, combine them with the customer profile (perhaps with a dot product), and make recommendations from there. Of course, most companies must do this on a much larger scale, which is where a particular field of mathematics, called **linear algebra**, can be very useful; we will look at it later in this chapter.

Logarithms/exponents

An **exponent** tells you how many times you have to multiply a number by itself, as illustrated in *Figures 4.3 and 4.4*:

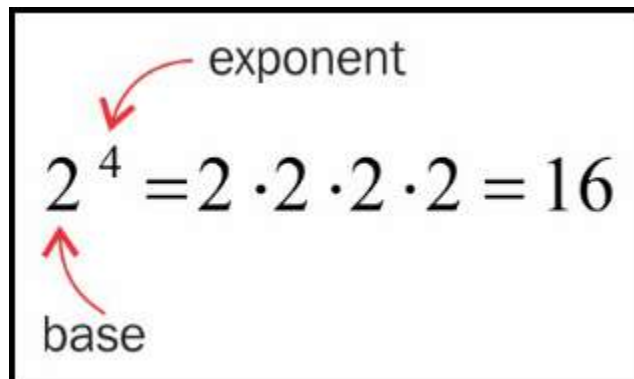

$$2^4 = 2 \cdot 2 \cdot 2 \cdot 2 = 16$$

Figure 4.2 – The exponent tells you how many times to multiply a number by itself

A **logarithm** is a number that answers the question “What exponent gets me from the base to this other number?” This can be denoted as follows:

$$\log_2(16) = 4$$

base logarithm

Figure 4.3 – The exponent from Figure 4.3 written in logarithm form

If these two concepts seem similar, then you are correct! Exponents and logarithms are heavily related. In fact, the words exponent and logarithm mean the same thing! A logarithm is an exponent. The preceding two equations are two versions of the same thing. The basic idea is that 2 times 2 times 2 times 2 is 16.

Figure 4.5 depicts how we can use both versions to say the same thing. Note how I use arrows to move from the log formula to the exponent formula:

$$\log_2(16) = 4 \leftrightarrow 2^4 = 16$$

Figure 4.4 – Logarithms and exponents are the same!

Consider the following examples:

- $\log_{31} 81 = 4$ because $3^4 = 81$
- $\log_5 125 = 3$ because $5^3 = 125$

Let's rewrite the first equation to note something interesting:

$$\log_3 81 = 4$$

Now, let's replace 81 with the equivalent statement, 3^4 , as follows:

$$\log_5 125 = 3$$

Something interesting to note is that the 3s seem to *cancel out*. This is very important when dealing with numbers that are more difficult to work with than 3s and 4s.

Exponents and logarithms are most important when dealing with growth. More often than not, if a quantity is growing (or declining in growth), an exponent/logarithm can help model this behavior.

For example, the number e is around 2.718 and has many practical applications. A very common application is interest calculation for saving. Suppose you have \$5,000 deposited in a bank with continuously compounded interest at the rate of 3%. In this case, you can use the following formula to model the growth of your deposit:

$$A = Pe^{rt}$$

In this formula, we have the following:

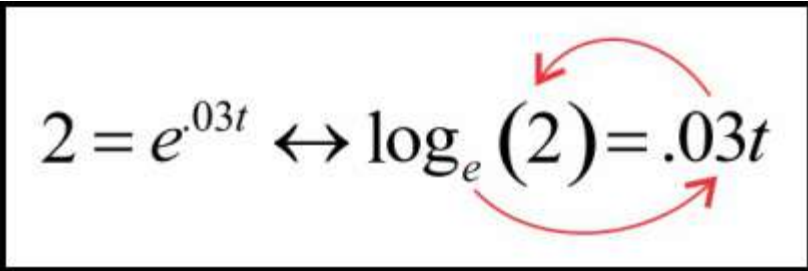
- A denotes the final amount
- P denotes the principal investment (5000)
- e denotes a constant (2.718)
- r denotes the rate of growth (.03)
- t denotes the time (in years)

When will our investment double? How long would I have to have my money in this investment to achieve 100% growth? We can write this in mathematical form, as follows:

$$10,000 = 5,000e^{.03t}$$

$$2 = e^{.03t} \text{ (divided by 5,000 on both sides)}$$

At this point, we have a variable in the exponent that we want to solve. When this happens, we can use the logarithm notation to figure it out:



$$2 = e^{.03t} \leftrightarrow \log_e(2) = .03t$$

Figure 4.5 – The conversion from exponent form to logarithm form

This leaves us with $\log_e(2) = .03t$.

When we take the logarithm of a number with a base of e , it is called a **natural logarithm**. We can rewrite the logarithm as follows:

$$\ln(2) = .03t$$

Using a calculator (or Python), we'll find that $\ln(2) = .69$:

$$.69 = .03t$$

$$t = 2.31$$

This means that it would take *2.31* years to double our money.

Set theory

Set theory involves mathematical operations at the set level. It is sometimes thought of as a basic fundamental group of theorems that governs the rest of mathematics. For our purpose, we'll use set theory to manipulate groups of elements.

A **set** is a collection of distinct objects.

That's it! A set can be thought of as a list in Python but with no repeat objects. There is even a set of objects in Python:

```
s = set()
s = set([1, 2, 2, 3, 2, 1, 2, 2, 3, 2])
# will remove duplicates from a list
s == {1, 2, 3}
```

In Python, curly braces, { }, can denote a set or a dictionary. Remember that a dictionary in Python is a set of key-value pairs. An example is shown here:

```
dict = {"dog": "human's best friend", "cat": "destroyer of world"} dict["dog"]# ==
"human's best friend"
len(dict["cat"]) # == 18
but if we try to create a pair with the same key as an existing key dict["dog"] =
"Arf"
dict
{"dog": "Arf", "cat": "destroyer of world"}
It will override the previous value
dictionaries cannot have two values for one key.
```

They share this notation because they share a quality in that sets cannot have duplicate elements, just as dictionaries cannot have duplicate keys.

The **magnitude** of a set is the number of elements in the set and is represented as follows:

$|A|$ = magnitude of A

We can get the magnitude of a set in Python using the `len` command:

```
# s == {1,2,3}
len(s) == 3 # magnitude of s
```

NOTE

The concept of an empty set exists and is denoted by { }. This null set is said to have a magnitude of 0.

If we wish to denote that an element is within a set, we can use the epsilon notation, as shown here:

$2 \in \{1,2,3\}$

This notation means that the element, 2, exists in the set of 1, 2, and 3. If one set is entirely inside another set, we say that it is a **subset** of its larger counterpart:

$$A = \{1,5,6\}, B = \{1,5,6,7,8\}$$

$$A \subseteq B$$

(A is a subset of B because every element in A is also in B.)

So, A is a subset of B and B is called the **superset** of A. If A is a subset of B but A does not equal B (meaning that there is at least one element in B that is not in A), then A is called a **proper subset** of B.

Consider the following examples:

- A set of even numbers is a subset of all integers
- Every set is a subset, but not a proper subset, of itself
- A set of all tweets is a superset of English tweets

In data science, we use sets (and lists) to represent a list of objects and, often, to generalize the behavior of consumers. It is common to reduce a customer to a set of characteristics.

Imagine that we are a marketing firm trying to predict where a person wants to shop for clothes. We are given a set of clothing brands the user has previously visited, and our goal is to predict a new store that they would also enjoy. Suppose a specific user has previously shopped at the following stores:

```
user1 = {"Target", "Banana Republic", "Old Navy"}  
note that we use {} notation to create a set  
compare that to using [] to make a list
```

So, `user1` has previously shopped at `Target`, `Banana Republic`, and `Old Navy`. Let's also look at a different user, called `user2`, as shown here:

```
user2 = {"Banana Republic", "Gap", "Kohl's"}
```

Suppose we are wondering how similar these users are. With the limited information we have, one way to define similarity is to see how many stores there are that they both shop at. This is called an **intersection**.

The intersection of two sets is a set whose elements appear in both sets. It is denoted using the \cap symbol, as shown here:

$$user1 \cap user2 = \{Banana Republic\}$$

$$user1 \cap user2 = \{Banana Republic\}$$

$$|user1 \cap user2| = 1$$

The intersection of the two users is just one store. So, right away, that doesn't seem great. However, each user only has three elements in their set, so having 1/3 does not seem as bad. Suppose we are

curious about how many stores are represented between the two of them; this is called a **union**.

The union of two sets is a set whose elements appear in either set. It is denoted using the \cup symbol, as shown here:

$$\text{user1} \cup \text{user2} = \{\text{Banana Republic}, \text{Target}, \text{Old Navy}, \text{Gap}, \text{Kohl's}\}$$

When looking at the similarities between `user1` and `user2`, we should use a combination of the **union** and the **intersection** of their sets. `user1` and `user2` have one element in common out of a total of five distinct elements between them. So, we can define the similarity between the two users as follows:

$$\frac{|\text{user1} \cap \text{user2}|}{|\text{user1} \cup \text{user2}|} = \frac{1}{5} = .2$$

This has a name in set theory: the **Jaccard measure**. In general, for the A and B sets, the Jaccard measure (Jaccard similarity) between the two sets is defined as follows:

$$JS(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

It can also be defined as the magnitude of the intersection of the two sets divided by the magnitude of the union of the two sets.

This gives us a way to quantify similarities between elements represented with sets.

Intuitively, the Jaccard measure is a number between 0 and 1, such that when the number is closer to 0, people are more dissimilar, and when the measure is closer to 1, people are considered similar to each other.

If we think about the definition, then it makes sense. Take a look at the measure once more:

$$JS(A, B) = \frac{\text{Number of stores they share in common}}{\text{Unique number of stores they liked combined}}$$

Here, the numerator represents the number of stores that the users have in common (in the sense that they like shopping there), while the denominator represents the unique number of stores that they like put together.

We can represent this in Python using some simple code, as shown here:

```
user1 = {"Target", "Banana Republic", "Old Navy"} user2 = {"Banana Republic", "Gap", "Kohl's"}
def jaccard(user1, user2):
    stores_in_common = len(user1 & user2)
    stores_all_together = len(user1 | user2)
    return stores_in_common / stores_all_together
# using our new jaccard function
jaccard(user1, user2) == # 0.2 or 1/5
```

Set theory becomes highly prevalent when we enter the world of probability and also when dealing with high-dimensional data. We can use sets to represent real-world events taking place, and probability becomes set theory with vocabulary on top of it.

Linear algebra

As we've already seen, a movie recommendation engine utilizes several mathematical concepts to provide accurate and personalized recommendations to users. However, in a scenario where a vast collection of 10,000 movies is available for recommendation, computational efficiency becomes a crucial factor. Linear algebra, an area of mathematics that deals with matrices and vectors, provides the necessary tools to perform these calculations efficiently.

Linear algebra focuses on analyzing and manipulating matrices and vectors to extract useful information and apply it in practical situations. As we proceed, it is important to have a solid understanding of the basic principles of linear algebra. Therefore, we will be reviewing several key rules of linear algebra before delving further into the topic.

Matrix multiplication

Like numbers, we can multiply matrices together. Multiplying matrices is, in essence, a mass-produced way of taking several dot products at once. Let's, for example, try to multiply the following matrices:

$$\begin{pmatrix} 1 & 5 \\ 5 & 8 \\ 7 & 8 \end{pmatrix} \cdot \begin{pmatrix} 3 & 4 \\ 2 & 5 \end{pmatrix}$$

We need to consider a couple of things:

- Unlike numbers, the multiplication of matrices is not *commutative*, meaning that the order in which you multiply matrices matters a great deal.
- To multiply matrices, their dimensions must match up. This means that the first matrix must have the same number of columns as the second matrix has rows.

To remember this, write out the dimensions of the matrices. In this case, we have a 3×2 matrix times a 2×2 matrix. You can multiply matrices together if the second number in the first-dimension pair is the same as the first number in the second-dimension pair. The resulting matrix will always have dimensions equal to the outer numbers in the dimension pairs (the ones you did not circle). In this case, the resulting matrix will have a dimension of 3×2 .

How to multiply matrices together

To multiply matrices, there is a simple procedure we can follow. Essentially, we are performing a bunch of dot products.

Recall our earlier sample problem, which was as follows:

$$\begin{pmatrix} 1 & 5 \\ 5 & 8 \\ 7 & 8 \end{pmatrix} \cdot \begin{pmatrix} 3 & 4 \\ 2 & 5 \end{pmatrix}$$

We know that our resulting matrix will have a dimension of 3×2 . So, we know it will look something like this:

$$\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \\ m_{31} & m_{32} \end{pmatrix}$$

NOTE

Note that each element of the matrix is indexed using a double index. The first number represents the row, while the second number represents the column. So, the m_{32} element is the element in the third row of the second column. Each element is the result of a dot product between rows and columns of the original matrices.

The m_{xy} element is the result of the dot product of the x th row of the first matrix and the y th column of the second matrix. Let's solve a few:

$$m_{11} = \begin{pmatrix} 1 \\ 5 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix} = 13$$

$$m_{11} = \begin{pmatrix} 1 \\ 5 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix} = 13$$

We eventually get a resulting matrix that looks as follows:

$$\begin{pmatrix} 13 & 29 \\ 31 & 60 \\ 37 & 68 \end{pmatrix}$$

Nice! Let's go back to the movie recommendation example. Recall the user's movie genre preferences of comedy, romance, and action, illustrated as follows:

$$U = \text{userprefs} = \begin{pmatrix} 5 \\ 1 \\ 3 \end{pmatrix}$$

Now, suppose we have 10,000 movies, all with a rating for these three categories. To make a recommendation, we need to take the dot product of the preference vector with each of the 10,000 movies. We can use matrix multiplication to represent this.

Instead of writing them all out, let's express them using the matrix notation. We already have U , defined here as the user's preference vector (it can also be thought of as a 3×1 matrix), but we also need a movie matrix:

$$M = \text{movies} = 3 \times 10,000$$

Now, we have two matrices; one is 3×1 and the other is $3 \times 10,000$. We can't multiply these matrices as they are because the dimensions do not work out. We will have to change U a bit. To do this, we can take the *transpose* of the matrix (turning all rows into columns and all columns into rows). This will switch the dimensions around:

$$U^T = \text{transpose of } U = (5 \ 1 \ 3)$$

So, now, we have two matrices that can be multiplied together. Let's visualize what this looks like:

$$\begin{pmatrix} 5 & 1 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 & \dots \\ 1 & 4 & \dots \\ 51 & 13 & \dots \end{pmatrix} \quad 3 \times 1,000$$

The resulting matrix will be a $1 \times 1,000$ matrix (a vector) of 10,000 predictions for each movie. Let's try this out in Python:

```
import numpy as np
# create user preferences
user_pref = np.array([5, 1, 3])
create a random movie matrix of 10,000 movies movies = np.random.randint(5,size=
(3,1000))+1
Note that the randint will make random integers from 0-4
so I added a 1 at the end to increase the scale from 1-5
```

We are using the **numpy** array function to create our matrices. We will have both a **user_pref** matrix and a **movies** matrix to represent our data.

To check our dimensions, we can use the **numpy shape** variable, as shown here:

```
print(user_pref.shape) # (1, 3)
print(movies.shape) # (3, 1000)
```

This checks out. Last but not least, let's use the matrix multiplication method of **numpy** (called dot) to perform the operation:

```
np.dot does both dot products and matrix multiplication np.dot(user_pref, movies)
```

The result is an array of integers that represents the recommendations for each movie.

For a quick extension of this, let's run some code that predicts across more than 10,000 movies:

```
import numpy as np
import time
for num_movies in (10000, 100000, 1000000, 10000000, 100000000):
    movies = np.random.randint(5,size=(3, num_movies))+1
    now = time.time()
    np.dot(user_pref, movies)
    print((time.time() - now), "seconds to run", num_movies, "movies")
0.000160932540894 seconds to run 10000 movies
0.00121188163757 seconds to run 100000 movies
0.0105860233307 seconds to run 1000000 movies
0.096577167511 seconds to run 10000000 movies
4.16197991371 seconds to run 100000000 movies
```

It took only a bit longer than 4 seconds to run through 100 million movies using matrix multiplication.

Summary

In this chapter, we took a look at some basic mathematical principles that will become very important as we progress through this book. Between logarithms/exponents, matrix algebra, and proportionality, mathematics has a big role not just in analyzing data but in many aspects of our lives.

The coming chapters will take a much deeper dive into two big areas of mathematics: probability and statistics. Our goal will be to define and interpret the smallest and biggest theorems in these two giant fields of mathematics.

It is in these next few chapters that everything will start to come together. So far in this book, we have looked at math examples, data exploration guidelines, and basic insights into types of data. It is time to begin to tie all of these concepts together.

Impossible or Improbable – A Gentle Introduction to Probability

Over the next few chapters, we will explore both probability and statistics as methods of examining both data-driven situations and real-world scenarios. The rules of probability govern the basics of prediction. We use probability to define the chances of the occurrence of an event.

In this chapter, we will look at the following topics:

- What do we mean by “probability”?
- The differences between the frequentist approach and the Bayesian approach
- How to visualize probability
- How to utilize the rules of probability
- Using confusion matrices to look at the basic metrics

Probability will help us model real-life events that include a sense of randomness and chance. Over the next two chapters, we will look at the terminology behind probability theorems and how to apply them to model situations that can appear unexpectedly.

Basic definitions

One of the most basic concepts of probability is the concept of a procedure. A **procedure** is an act that leads to a result, for example, throwing a die or visiting a website.

An **event** is a collection of the outcomes of a procedure, such as getting heads on a coin flip or leaving a website after only four seconds. A simple event is an outcome/event of a procedure that cannot be broken down further. For example, rolling two dice can be broken down into two simple events: rolling die 1 and rolling die 2.

The **sample space** of a procedure is the set of all possible simple events. For example, an experiment is performed in which a coin is flipped three times in succession. What is the size of the sample space for this experiment?

The answer is eight because the results could be any one of the possibilities in the following sample space: {HHH, HHT, HTT, HTH, TTT, TTH, THH, or THT}.

What do we mean by “probability”?

The **probability** of an event represents the frequency, or chance, that the event will happen.

For **notation**, if A is an event, P(A) is the probability of the occurrence of the event.

We can define the actual probability of an event, A, as follows:

$$P(A) = \frac{\text{number of ways } A \text{ occurs}}{\text{size of sample space}}$$

Here, A is the event in question. Think of an entire universe of events where anything is possible, and let's represent it as a circle. We can think of a single event, A, as being a smaller circle within that larger universe, as shown in the following diagram:

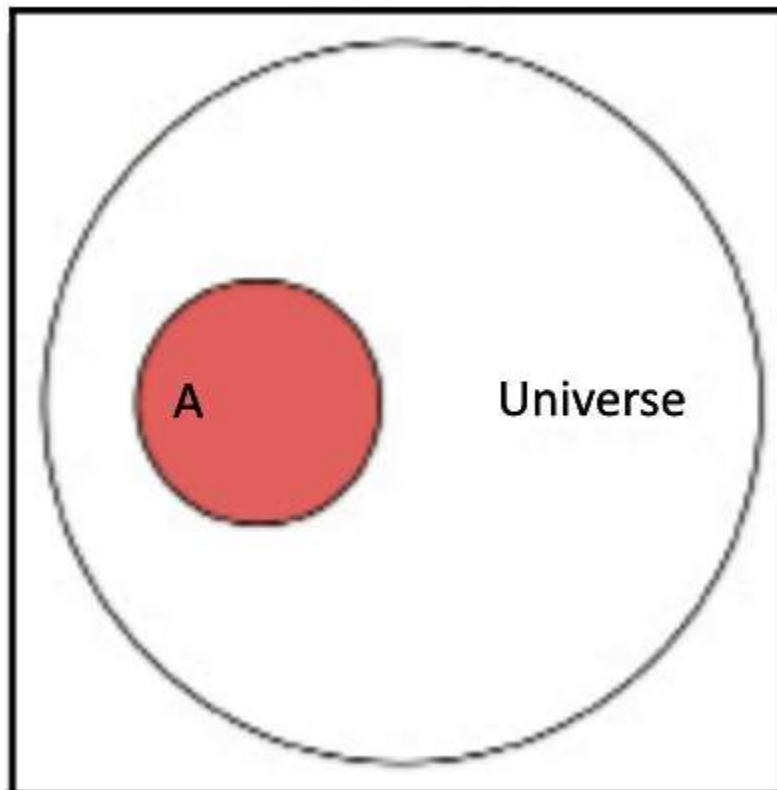


Figure 5.1 – An event (generally labeled with letters such as A) is only a single event in a universe of possible events

Let's now pretend that our universe involves a research study on humans and the A event is people in that study who have cancer.

If our study has 100 people and A has 25 people, the probability of A or P(A) is 25/100.

The maximum probability of any event is 1. This can be understood as the red circle grows so large that it is the size of the universe (the larger circle).

The most basic examples (I promise they will get more interesting) are coin flips. Let's say we have two coins and we want the probability that we will get two heads. We can very easily count the number

of ways two coins could end up being two heads. There's only one! Both coins have to be heads. But how many options are there? It could either be two heads, two tails, or a heads/tails combination.

First, let's define A. It is the event in which two heads occur. The number of ways that A can occur is 1.

The sample space of the experiment is {HH, HT, TH, TT}, where each two-letter *word* indicates the outcome of the first and second coin simultaneously. The size of the sample space is four. So, $P(\text{getting two heads}) = 1/4$.

Let's refer to a quick visual table to prove it. The following table denotes the options for coin 1 as the columns, and the options for coin 2 as the rows. In each cell, there is either **True** or **False**. A **True** value indicates that it satisfies the condition (both heads), and **False** indicates otherwise:

	Coin 1 is heads	Coin 1 is tails
Coin 2 is heads	True	False
Coin 2 is tails	False	False

Table 5.1 – This two-way table shows all four possible options for what could happen if you flip two independent coins (coin 1 and coin 2)

Let's try, with the help of an example, to understand the previous figure; for example, the bottom-right value of **False** represents the scenario where both coins come up tails. There are only four possible combinations, so we say that the “sample space” of the experiment is size 4.

So, we have one out of a total of four possible outcomes when we flip these two coins.

Bayesian versus frequentist

The preceding example was frankly too easy. In practice, we can hardly ever truly count the number of ways something can happen. For example, let's say that we want to know the probability of a random person smoking cigarettes at least once a day. If we wanted to approach this problem using the classical method (the previous formula), we would need to figure out how many different ways a person is a smoker—someone who smokes at least once a day—which is not possible!

When faced with such a problem, two main schools of thought are considered when it comes to calculating probabilities in practice: the **Frequentist approach** and the **Bayesian approach**. This chapter will focus heavily on the Frequentist approach, while the subsequent chapter will dive into the Bayesian approach.

Frequentist approach

In a Frequentist approach, the probability of an event is calculated through experimentation. It uses the past in order to predict the future chance of an event. The basic formula is as follows:

$$P(A) = \frac{\text{number of times } A \text{ occurred}}{\text{number of times the procedure was repeated}}$$

Basically, we observe several instances of the event and count the number of times A was satisfied. The division of these numbers is an approximation of the probability.

The Bayesian approach differs by dictating that probabilities must be discerned using theoretical means. Using the Bayesian approach, we would have to think a bit more critically about events and why they occur. Neither methodology is wholly the correct answer all of the time. Usually, it comes down to the problem and the difficulty of using either approach.

The crux of the Frequentist approach is the relative frequency.

The **relative frequency** of an event is how often an event occurs divided by the total number of observations.

Example – marketing stats

Let's say that you are interested in figuring out how often a person who visits your website is likely to return at a later date. This is sometimes called the rate of repeat visitors. In the previous definition, we would define our A event as being a visitor coming back to the site. We would then have to calculate the number of ways a person can come back, which doesn't really make sense at all! In this case, many people would turn to a Bayesian approach; however, we can calculate what is known as relative frequency.

So, in this case, we can take the visitor logs and calculate the relative frequency of event A (repeat visitors). Let's say, of the 1,458 unique visitors in the past week, 452 were repeat visitors. We can calculate this as follows:

$$RF(A) = \frac{452}{1458} = 0.31$$

So, about 31% of our visitors are repeat visitors.

The law of large numbers

The reason that we can use the Frequentist approach to solve problems such as our marketing example is because of the law of large numbers, which states that if we repeat a procedure over and over, the relative frequency probability will approach the actual probability. Let's try to demonstrate this using Python.

If I were to ask you the average of the numbers 1 and 10, you would very quickly answer around 5. This question is identical to asking you to pick the average number between 1 and 10. Let's design the experiment to be as follows.

Python will choose n random numbers between 1 and 10 and find their average.

We will repeat this experiment several times using a larger n each time, and then we will graph the outcome. The steps are as follows:

1. Pick a random number between 1 and 10 and find the average.
2. Pick two random numbers between 1 and 10 and find their average.
3. Pick three random numbers between 1 and 10 and find their average.
4. Pick 10,000 random numbers between 1 and 10 and find their average.
5. Graph the results.

Let's take a look at the code:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
results = []
for n in range(1, 10000):
    nums = np.random.randint(low=1, high=10, size=n) # choose n numbers between 1 and 10
    mean = nums.mean() # find the average of these numbers
    results.append(mean) # add the average to a running list
# POP QUIZ: How large is the list results? len(results) # 9999
This was tricky because I took the range from 1 to 10000 and usually we do from 0 to 10000
df = pd.DataFrame({'means' : results})
print (df.head()) # the averages in the beginning are all over the place!
means
9.0
5.0
6.0
4.5
4.0
print (df.tail()) # as n, our size of the sample size, increases, the averages get closer to 5!
means
4.998799
5.060924
4.990597
5.008802
4.979198
df.plot(title='Law of Large Numbers')
plt.xlabel("Number of throws in sample")
plt.ylabel("Average Of Sample")
```

The resulting graph is shown in *Figure 5.2*:

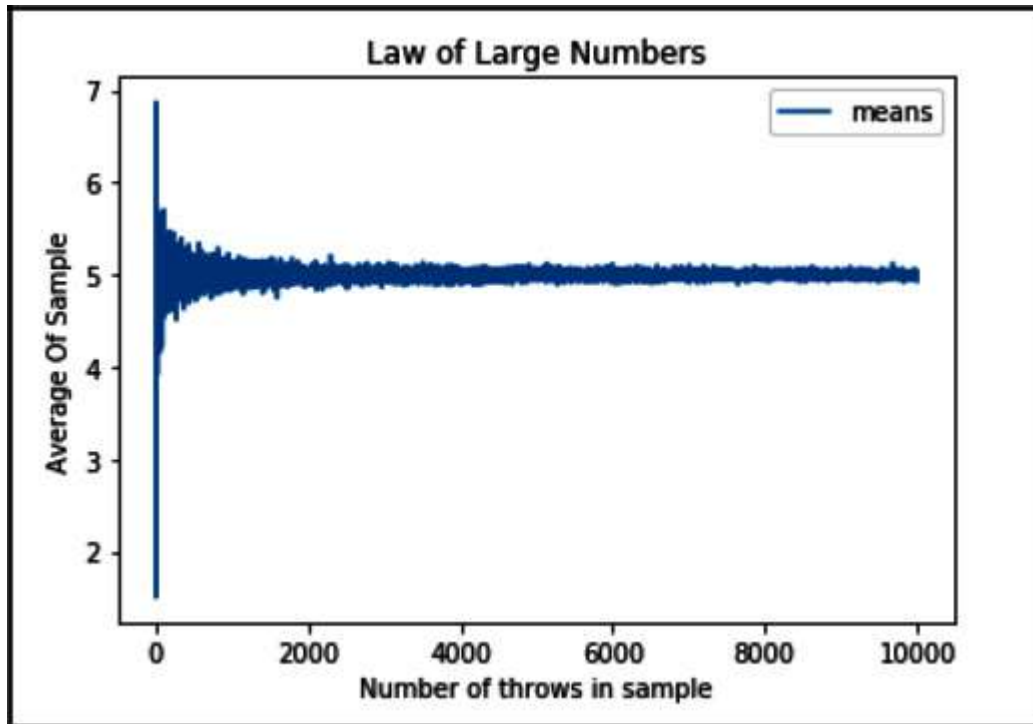


Figure 5.2 – Graphical representation of the law of large numbers

If we take 10,000 random numbers between 1 and 10 and average them, we get a number very close to 5. This is because as we increase the sample size of our experiment, the resulting value approaches the true average.

Cool, right? What this is essentially showing us is that as we increase the sample size of our relative frequency, the frequency approaches the actual average (probability) of 5.

In our statistics chapters, we will work to define this law much more rigorously, but for now, just know that it is used to link the relative frequency of an event to its actual probability.

Compound events

Sometimes, we need to deal with two or more events. These are called **compound events**. A compound event is an event that combines two or more simple events. When this happens, we need a special notation.

Given events A and B, note the following:

- The probability that A and B occur is $P(A \cap B) = P(A \text{ and } B)$
- The probability that either A or B occurs is $P(A \cup B) = P(A \text{ or } B)$

Understanding why we use the set notation for these compound events is very important. Remember how we represented events in a universe using circles earlier? Let's say that our *universe* is 100 people

who showed up for an experiment in which a new test for cancer is being developed:

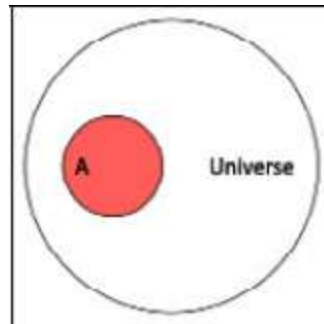


Figure 5.3 – Performing probability and statistics with our universe as 100 people and an event A

Recall that when we talk about an event such as A, we are sometimes talking about people we are trying to help. Probability and statistics are often performed in the service of helping our fellow humankind

In the preceding diagram, the red circle, A, represents 25 people who actually have cancer. Using the relative frequency approach, we can say that $P(A) = \text{number of people with cancer} / \text{number of people in the study}$, that is, $25/100 = 1/4 = 0.25$. This means that there is a 25% chance that someone has cancer.

Let's introduce a second event, called B, as shown, which contains people for whom the test was positive (it claimed that they had cancer). Let's say that this is for 30 people. So, $P(B) = 30/100 = 3/10 = 0.3$. This means that there is a 30% chance that the test said positive for any given person:

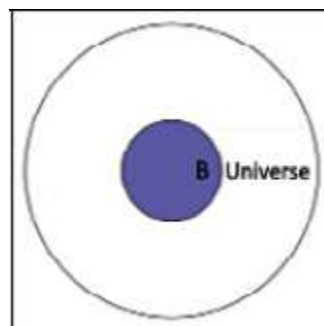


Figure 5.4 – We often need to consider multiple events and so we have an event B in conjunction with event A

These are two separate events, but they interact with each other. Namely, they might *intersect* or have people in common, as shown here:

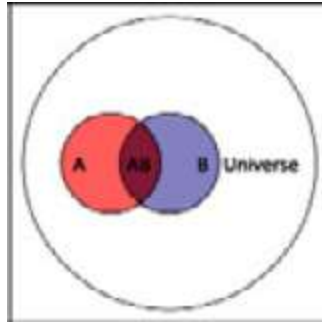


Figure 5.5 – When there is a chance that two events could happen simultaneously, we say that these events intersect

Anyone in the space that both A and B occupy, otherwise known as A intersect B or $A \cap B$, are people for whom the test claimed they were positive for cancer (A), and they actually do have cancer. Let's say that's 20 people. The test said positive for 20 people, that is, they have cancer, as shown here:

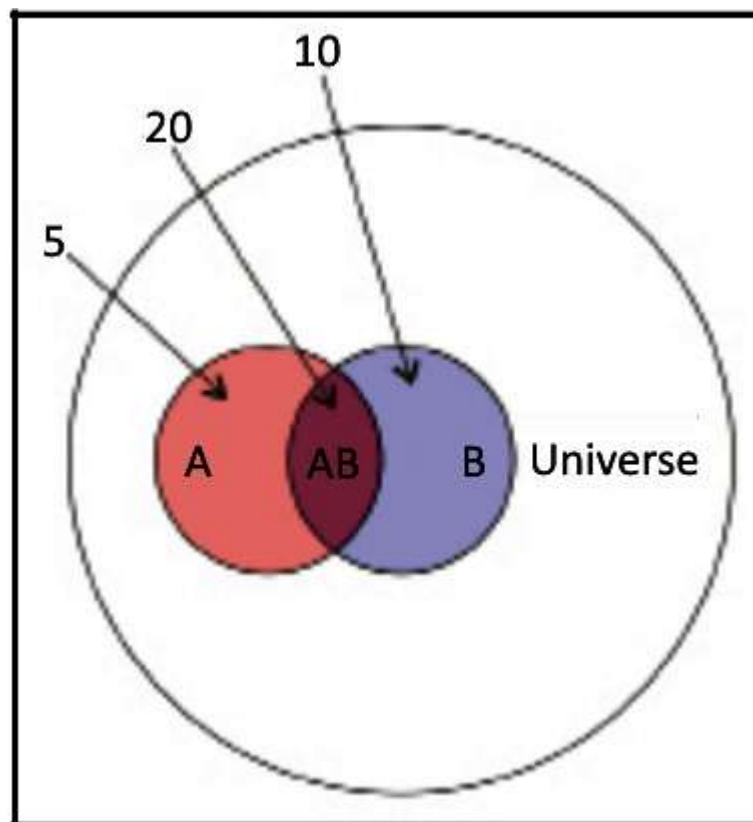


Figure 5.6 – If event A represents people who have cancer and B represents people whose cancer tests were positive, the intersection of A and B represents people who were told they have cancer and the test was accurate in its diagnosis

This means that $P(A \text{ and } B) = 20/100 = 1/5 = 0.2 = 20\%$.

If we want to say that someone has cancer, or the test came back positive, this would be the total sum (or union) of the two events, namely, the sum of 5, 20, and 10, which is 35. So, $35/100$ people either have cancer or have a positive test outcome. That means $P(A \text{ or } B) = 35/100 = 0.35 = 35\%$.

All in all, we have people in the following four different classes:

- **Pink:** This refers to those people who have cancer and had a negative test outcome
- **Purple (A intersect B):** These people have cancer and had a positive test outcome
- **Blue:** This refers to those people who don't have cancer and had a positive test outcome
- **White:** This refers to those people who don't have cancer and had a negative test outcome

So, effectively, the only times the test was *accurate* were in the white and purple regions. In the blue and pink regions, the test was incorrect.

Conditional probability

Let's pick an arbitrary person from this study of 100 people. Let's also assume that you are told that their test result was positive. So, we are told that event B has already taken place and that their test came back positive. The question now is: what is the probability that they have cancer, that is, $P(A)$? This is called a **conditional probability of A given B** or $P(A|B)$. Effectively, it is asking you to calculate the probability of an event given that another event has already happened.

You can think of conditional probability as changing the relevant universe. $P(A|B)$ (called the probability of A given B) is a way of saying, given that my entire universe is now B, what is the probability of A? This is also known as transforming the sample space.

Zooming in on our previous diagram, our universe is now B, and we are concerned with AB (A and B) inside B.

The formula can be given as follows:

$$P(A|B) = P(A \text{ and } B) / P(B) = (20/100) / (30/100) = 20/30 = 0.66 = 66\%$$

There is a 66% chance that if a test result came back positive, that person had cancer. In reality, this is the main probability that the experimenters want. They want to know how good the test is at predicting cancer.

How to utilize the rules of probability

In probability, we have some rules that become very useful when visualization gets too cumbersome. These rules help us calculate compound probabilities with ease.

The addition rule

The addition rule is used to calculate the probability of *either/or* events. To calculate $P(A \cup B)$, or $P(A \text{ or } B)$, we use the following formula:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

The first part of the formula ($P(A) + P(B)$) helps us get the union of the two events; we have to add together the area of the circles in the universe. But why the subtraction of $P(A \text{ and } B)$? This is because when we add the two circles, we are adding the area of intersection twice, as shown in the following diagram:

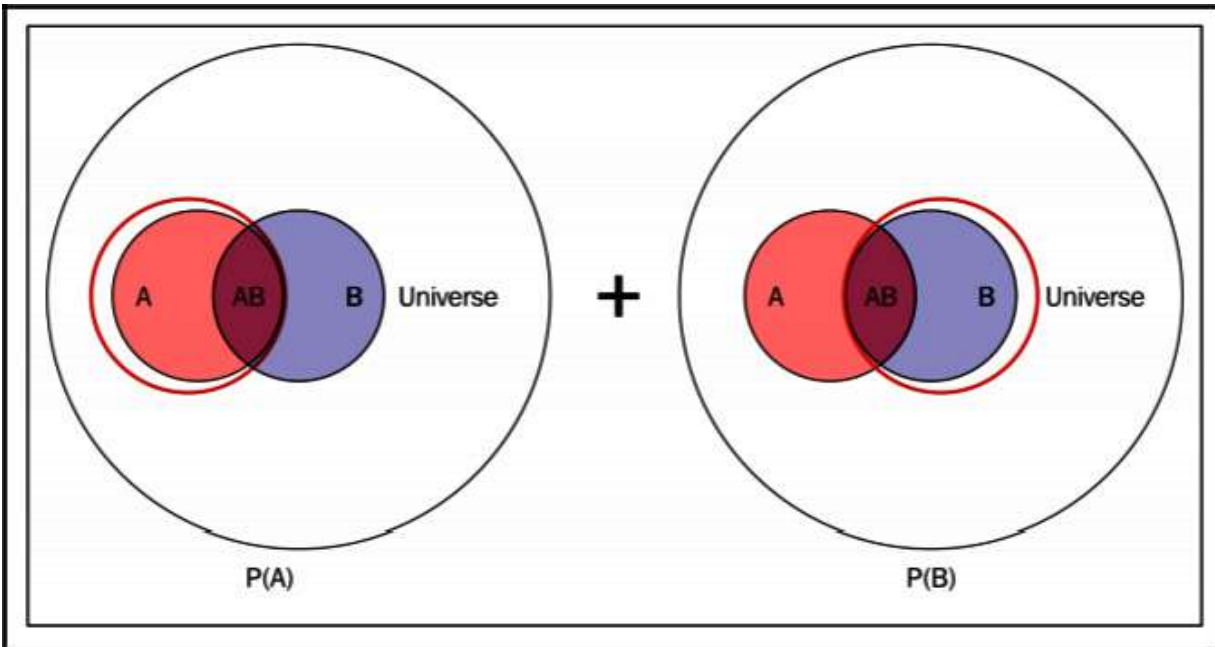


Figure 5.7 – Adding the probabilities of two events and being careful not to double count their intersection

See how both the red circles include the intersection of A and B? So, when we add them, we need to subtract just one of them to account for this, leaving us with our formula.

You will recall that we wanted the number of people who either had cancer or had a positive test result. If A is the event that someone has cancer, and B is that the test result was positive, we have the following:

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B) = 0.25 + 0.30 - 0.2 = 0.35$$

Mutual exclusivity

We say that two events are mutually exclusive if they cannot occur at the same time. This means that $A \cap B = \emptyset$, or just that the intersection of the events is the empty set. When this happens, $P(A \cap B) = P(A \text{ and } B) = 0$.

If two events are mutually exclusive, then the following applies:

$$P(A \cup B) = P(A \text{ or } B)$$

$$= P(A) + P(B) - P(A \cap B) = P(A) + P(B)$$

This makes the addition rule much easier. Some examples of mutually exclusive events include the following:

- A customer seeing your site for the first time on both Twitter and Facebook
- Today is Saturday and today is Wednesday
- I failed Econ 101 and I passed Econ 101

None of these events can occur simultaneously.

The multiplication rule

The multiplication rule is used to calculate the probability of *and* events. To calculate $P(A \cap B) = P(A \text{ and } B)$, we use the following formula:

$$P(A \cap B) = P(A \text{ and } B) = P(A) P(B|A)$$

Why do we use $B|A$ instead of B ? This is because it is possible that B depends on A . If this is the case, then just multiplying $P(A)$ and $P(B)$ does not give us the whole picture.

In our cancer trial example, let's find $P(A \text{ and } B)$. To do this, let's redefine A to be the event that the trial is positive, and B to be the person having cancer (because it doesn't matter what we call the events). The equation will be as follows:

$$P(A \cap B) = P(A \text{ and } B) = P(A) P(B|A) = 0.3 * 0.6666 = 0.2 = 20\%$$

It's difficult to see the true necessity of using conditional probability, so let's try another, more difficult problem.

For example, among a randomly selected set of 10 people, 6 have iPhones and 4 have Androids. What is the probability that if I randomly select two people, they will both have iPhones? This example can be retold using event spaces, as follows.

I have the following two events:

- A : This event shows the probability that I choose a person with an iPhone first
- B : This event shows the probability that I choose a person with an iPhone second

So, basically, I want the following:

- $P(A \text{ and } B)$: $P(\text{I choose a person with an iPhone and a person with an iPhone})$. So, we can use the $P(A \text{ and } B) = P(A) P(B|A)$ formula.

$P(A)$ is simple, right? Six out of 10 people have an iPhone. So, I have a $6/10 = 3/5 = 0.6$ chance of A . This means $P(A) = 0.6$.

So, if I have a 0.6 chance of choosing someone with an iPhone, the probability of choosing two should just be $0.6 * 0.6$, right?

But wait! We only have nine people left to choose our second person from, because one was taken away. So, in our new transformed sample space, we have nine people in total, five with iPhones and four with Androids, making $P(B) = 5/9 = 0.555$.

So, the probability of choosing two people with iPhones is $0.6 * 0.555 = 0.333 = 33\%$.

I have a $1/3$ chance of choosing two people with iPhones out of 10. The conditional probability is very important in the multiplication rule, as it can drastically alter your answer.

Independence

Two events are independent if one event does not affect the outcome of the other, that is, $P(B|A) = P(B)$ and $P(A|B) = P(A)$.

If two events are independent, then the following applies:

$$P(A \cap B) = P(A) P(B|A) = P(A) P(B)$$

Some examples of independent events are as follows:

- It was raining in San Francisco, and a puppy was born in India
- I flip a coin and get heads, and I flip another coin and get tails

None of these pairs of events affects each other.

Complementary events

The complement of A is the opposite or negation of A. If A is an event, \bar{A} represents the complement of A. For example, if A is the event where someone has cancer, \bar{A} is the event where someone is cancer-free.

To calculate the probability of \bar{A} , use the following formula:

$$P(\bar{A}) = 1 - P(A)$$

For example, when you throw two dice, what is the probability that you rolled higher than a 3?

Let A represent rolling higher than a 3.

The following formula represents rolling a 3 or less.

$$P(\bar{A}) = 1 - P(A)$$

$$P(A) = 1 - (P(2) + P(3)) = 1 - \left(\frac{2}{36} + \frac{2}{36}\right) = 1 - \left(\frac{4}{36}\right) = \frac{32}{36} = \frac{8}{9} = 0.89$$

For example, a start-up team has three investor meetings coming up. We will have the following probabilities:

- A 60% chance of getting money from the first meeting
- A 15% chance of getting money from the second
- A 45% chance of getting money from the third

What is the probability of them getting money from at least one meeting?

Let A be the team getting money from at least one investor, and \bar{A} be the team not getting any money. $P(A)$ can be calculated as follows:

$$P(\bar{A}) = 1 - P(A)$$

To calculate $P(\bar{A})$, we need to calculate the following:

$$P(\bar{A}) = P(\text{no money from investor 1 AND no money from investor 2 AND no money from investor 3})$$

Let's assume that these events are independent (they don't talk to each other):

$$P(\bar{A}) = P(\text{no money from investor 1}) * P(\text{no money from investor 2}) * P(\text{no money from investor 3}) = 0.4 * 0.85 * 0.55 = 0.187$$

$$P(A) = 1 - 0.187 = 0.813 = 81\%$$

So, the start-up has an 81% chance of getting money from at least one meeting!

Introduction to binary classifiers

Without getting too deep into machine learning terminology, our previous example of a cancer test is what is known as a **binary classifier**, which means that it is trying to predict from only two options: having cancer or not having cancer. When we are dealing with binary classifiers, we can draw what is called **confusion matrices**, which are 2 x 2 matrices that house all four possible outcomes of our experiment.

Let's try some different numbers. Let's say 165 people walked in for the study. So, our n (sample size) is 165 people. All 165 people are given the test and asked whether they have cancer (provided through various other means). The following **confusion matrix** shows us the results of this experiment:

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Figure 5.8 – Confusion matrix

The matrix shows that 50 people were predicted to not have cancer and did not have it, 100 people were predicted to have cancer and actually did have it, and so on. We have the following four classes, again, all with different names:

- The *true positives* are the tests correctly predicting positive (cancer) = 100
- The *true negatives* are the tests correctly predicting negative (no cancer) = 50
- The *false positives* are the tests incorrectly predicting positive (cancer) = 10
- The *false negatives* are the tests incorrectly predicting negative (no cancer) = 5

The first two classes indicate where the test was correct, or true. The last two classes indicate where the test was incorrect, or false.

False positives are sometimes called **type I errors**, whereas false negatives are called **type II errors**.

We will talk more about types of errors in later chapters. For now, we just need to understand why we use the set notation to denote probabilities for compound events. This is because that's what they are.

When events A and B exist in the same universe, we can use intersections and unions to represent them happening either at the same time or to represent one happening versus the other.

We will go into this much more in later chapters, but it is good to introduce it now.

Summary

In this chapter, we looked at the basics of probability, and will continue to dive deeper into this field in the following chapter. We approached most of our thinking as a Frequentist and expressed the basics of experimentation and using probability to predict an outcome.

The next chapter will look at the Bayesian approach to probability and will also explore the use of probability to solve much more complex problems. We will incorporate these basic probability principles into much more difficult scenarios.

Advanced Probability

In the previous chapter, we went over the basics of probability and how we can apply simple theorems to complex tasks. To briefly summarize, probability is the mathematics of modeling events that may or may not occur. We use formulas in order to describe these events and even look at how multiple events can behave together.

In this chapter, we will explore more complicated theorems of probability and how we can use them in a predictive capacity. Advanced topics, such as **Bayes' theorem** and **random variables**, give rise to common machine learning algorithms, such as the **Naïve Bayes algorithm** (also covered in this book).

This chapter will focus on some of the more advanced topics in probability theory, including the following topics:

- Exhaustive events
- Bayes' theorem
- Basic prediction rules
- Random variables

In later chapters, we will revisit Bayes' theorem and use it to create a very powerful and fast machine learning algorithm, called the Naïve Bayes algorithm. This algorithm captures the power of Bayesian thinking and applies it directly to the problem of predictive learning. For now, let's get started with Bayesian thinking!

Bayesian ideas revisited

In the last chapter, we talked very briefly about Bayesian ways of thinking. Recall that the Bayesian way of thinking is to let our data shape and update our beliefs. We start with a prior probability, or what we naïvely think about a hypothesis, and then we have a posterior probability, which is what we think about a hypothesis, given some data.

Bayes' theorem

Bayes' theorem is arguably the most well-known part of Bayesian inference. Recall that we previously defined the following:

- $P(A)$ = the probability that event A occurs

- $P(A|B)$ = the probability that A occurs, given that B occurred
- $P(A, B)$ = the probability that A and B occur
- $P(A, B) = P(A) * P(B|A)$

That last bullet can be read as “the probability that both A and B occur is equal to the probability that A occurs x times the probability that B occurred, given that A has already occurred.”

Starting from the last bullet points, we know the following:

$$P(A, B) = P(A) * P(B|A)$$

and

$$P(B, A) = P(B) * P(A|B)$$

And we also know that:

$$P(A, B) = P(B, A)$$

So, we can combine these together and see that:

$$P(B) * P(A|B) = P(A) * P(B|A)$$

Dividing both sides by $P(B)$ gives us Bayes’ theorem, as shown here:

$$P(A|B) = \frac{P(A) * P(B|A)}{P(B)}$$

So, our end result is Bayes’ theorem! This is a way to get from $P(A|B)$ to $P(B|A)$ (if you only have one of them) and a way to get $P(A|B)$ if you already know $P(A)$ (without knowing B).

Let’s try thinking about Bayes using the terms *hypothesis* and *data*.

Suppose H = your hypothesis about the given data and D = the data that you are given.

Bayes can be interpreted as trying to figure out $P(H|D)$ (the probability that our hypothesis is correct, given the data at hand).

Let’s use our terminology from before:

$$P(H|D) = \frac{P(D|H) P(H)}{P(D)}$$

Let’s take a look at that formula:

- $P(H)$ is the probability of the hypothesis before we observe the data, called the **prior probability**, or just **prior**
- $P(H|D)$ is what we want to compute, the probability of the hypothesis after we observe the data, called the **posterior**
- $P(D|H)$ is the probability of the data under the given hypothesis, called the **likelihood**
- $P(D)$ is the probability of the data under any hypothesis, called the **normalizing constant**

This concept is not far off from the idea of machine learning and predictive analytics. In many cases, when considering predictive analytics, we use the given data to predict an outcome. Using the current

terminology, H (our hypothesis) can be considered our outcome, and $P(H|D)$ (the probability that our hypothesis is true, given our data) is another way of saying, what is the chance that my hypothesis is correct, given the data in front of me?

Let's take a look at an example of how we can use Bayes' formula in the workplace.

Consider that you have two people in charge of writing blog posts for your company, Lucy and Avinash. From past performance, you like 80% of Lucy's work and only 50% of Avinash's work. A new blog post comes to your desk in the morning, but the author isn't mentioned. You love the article. A+. What is the probability that it came from Avinash? Each blogger blogs at a very similar rate.

Before we freak out, let's do what any experienced mathematician (and now you) would do. Let's write out all of our information, as shown here:

- H (hypothesis) = the blog came from Avinash
- D (data) = you loved the blog post
- $P(H|D)$ = the chance that it came from Avinash, given that you loved it
- $P(D|H)$ = the chance that you loved it, given that it came from Avinash
- $P(H)$ = the chance that an article came from Avinash
- $P(D)$ = the chance that you love an article

Note that some of these variables make almost no sense without context. $P(D)$, the probability that you would love any given article put on your desk, is a weird concept, but trust me – in the context of Bayes' formula, it will be relevant very soon.

Also, note that in the last two items, they assume nothing else. *$P(D)$ does not assume the origin of the blog post; think of $P(D)$ as asking, if an article was plopped on your desk from some unknown source, what is the chance that you'd like it?* (again, I know it sounds weird out of context).

So, we want to know $P(H|D)$. Let's try to use Bayes' theorem, as shown here:

$$P(H|D) = \frac{P(D|H) P(H)}{P(D)}$$

But do we know the numbers on the right-hand side of this equation? I say that we do! Let's see here:

- $P(H)$ is the probability that any given blog post comes from Avinash. As bloggers write at a very similar rate, we can assume this is 0.5 because we have a 50/50 chance that it came from either blogger (note how I did not assume D , the data, for this).
- $P(D|H)$ is the probability that you love a post from Avinash, which we previously said was 50% – so, 0.5.
- $P(D)$ is interesting. This is the chance that you love an article *in general*. It means that we must take into account the scenario where the post came from either Lucy or Avinash. Now, if the hypothesis forms a suite, we can use our laws of probability, as mentioned in the previous chapter. A suite is formed when a set of hypotheses is both collectively exhaustive – meaning at least one must occur – and mutually exclusive. In layman's terms, in a suite of events, exactly one and only one hypothesis can occur. In our case, the two hypotheses are that the article came from Lucy, or that the article came from Avinash. This is definitely a suite for the following reasons:

- At least one of them wrote it
- At most one of them wrote it
- Therefore, *exactly one* of them wrote it

When we have a suite, we can use our multiplication and addition rules, as follows:

$$\begin{aligned}
 D &= (\text{From Avinash AND loved it}) \text{ OR } (\text{From Lucy AND loved it}) \\
 P(D) &= P(\text{Loved AND from Avinash}) \text{ OR } P(\text{Loved AND from Lucy}) \\
 P(D) &= P(\text{From Avinash})P(\text{Loved} \mid \text{from Avinash}) \\
 &\quad + P(\text{from Lucy})P(\text{Loved} \mid \text{from Lucy}) \\
 P(D) &= .5(.5) + .5(.8) = \mathbf{.65}
 \end{aligned}$$

Figure 6.1 – An example of the multiplication and addition rules in action

Whew! Way to go! Now, we can finish our equation, as shown here:

$$P(H|D) = \frac{P(D|H) P(H)}{P(D)} P(H|D) = \frac{.5 * .5}{.65} = .38$$

This means that there is a 38% chance that this article comes from Avinash. What is interesting is that $P(H) = 0.5$ and $P(H|D) = 0.38$. This means that without any data, the chance that a blog post came from Avinash was a coin flip, or 50/50. Given some data (your thoughts on the article), we updated our beliefs about the hypothesis, and it actually lowered the chance. This is what Bayesian thinking is all about – updating our posterior beliefs about something from a prior assumption, given some new data about the subject.

More applications of Bayes' theorem

Bayes' theorem shows up in a lot of applications, usually when we need to make fast decisions based on data and probability. Most recommendation engines, such as Netflix's, use some elements of Bayesian updating. And if you consider why that might be, it makes sense.

Let's suppose that, in our simplistic world, Netflix only has 10 categories to choose from. Now, suppose that, given no data, a user's chance of liking a comedy movie out of 10 categories is 10% (just 1/10).

Okay, now suppose that the user has given a few comedy movies 5/5 stars. Now, when Netflix wonders what the chance is that the user would like another comedy, the probability that they might like a comedy, $P(H|D)$, is going to be larger than a random guess of 10%!

Let's try some more examples of applying Bayes' theorem using more data. This time, let's get a bit grittier.

Example – Titanic

A very famous dataset involves looking at the survivors of the sinking of the Titanic in 1912. We will use an application of probability in order to figure out whether there were any demographic features that showed a relationship to passenger survival. Mainly, we are curious to see whether we can isolate any features of our dataset that can tell us more about the types of people who were likely to survive this disaster.

First, let's read in the data, as shown here:

```
titanic =  
pd.read_csv(https://raw.githubusercontent.com/sinanuozdemir/SF\_DAT\_15/master/data/titanic.csv)#read in a csv  
titanic = titanic[['Sex', 'Survived']] #the Sex and Survived column titanic.head()
```

We get the following table:

	Sex	Survived
0	male	no
1	female	yes
2	female	yes
3	female	yes
4	male	no

Table 6.1 – Representing the Titanic dataset options Sex and Survived

The Titanic dataset only has two options for **Sex** and two options for **Survived**. While Survived is a relatively straightforward feature, this is one of our first examples of using a dataset in which data

interpretations have “drifted” over time. We will revisit the concept of drift in a later chapter, and how datasets such as the Titanic dataset can be easy to work with but are outdated in their utility.

In the preceding table, each row represents a single passenger on the ship, and, for now, we will look at two specific features – the sex of the individual and whether or not they survived the sinking. For example, the first row represents a man who did not survive, while the fourth row (with index 3 – remember how Python indexes lists) represents a female who did survive.

Let’s start with some basics. Let’s start by calculating the probability that any given person on the ship survived, regardless of their gender. To do this, let’s count the number of yeses in the **Survived** column and divide this figure by the total number of rows, as shown here:

```
num_rows = float(titanic.shape[0]) # == 891 rows
p_survived = (titanic.Survived=="yes").sum() / num_rows #
==
.38
p_notsurvived = 1 - p_survived
#
==
.61
```

Note that I only had to calculate $P(\text{Survived})$, and I used the law of conjugate probabilities to calculate $P(\text{Died})$ because those two events are complementary. Now, let’s calculate the probability that any single passenger is male or female:

```
p_male = (titanic.Sex=="male").sum() / num_rows # == .65
p_female = 1 - p_male # == .35
```

Now, let’s ask ourselves a question – did having a certain gender affect the survival rate? For this, we can estimate $P(\text{Survived}|\text{Female})$ or the chance that someone survived given that they were female. For this, we need to divide the number of women who survived by the total number of women, as shown here:

$$P(\text{Survived}|\text{Female}) = \frac{P(\text{Female AND Survives})}{P(\text{Female})}$$

Here’s the code for that calculation:

number_of_women = titanic[titanic.Sex=='female'].shape[0] #==		314
women_who_lived = titanic[(titanic.Sex=='female') &		
(titanic.Survived=='yes')].shape[0]	#==	233

```
p_survived_given_woman = women_who_lived / float(number_of_women)
p_survived_given_woman # == .74
```

That’s a pretty big difference. It seems that gender plays a big part in this dataset.

Example – medical studies

A classic use of Bayes' theorem is the interpretation of medical trials. Routine testing for illegal drug use is increasingly common in workplaces and schools. The companies that perform these tests maintain that the tests have a high sensitivity, which means that they are likely to produce a positive result if there are drugs in their system. They claim that these tests are also highly specific, which means that they are likely to yield a negative result if there are no drugs.

On average, let's assume that the sensitivity of common drug tests is about 60% and the specificity is about 99%. It means that if an employee is using drugs, the test has a 60% chance of being positive, while if an employee is not on drugs, the test has a 99% chance of being negative. Now, suppose these tests are applied to a workforce where the actual rate of drug use is 5%.

The real question here is, of the people who test positive, how many actually use drugs?

In Bayesian terms, we want to compute the probability of drug use, given a positive test:

- Let D = the event that drugs are in use
- Let E = the event that the test is positive
- Let N = the event that drugs are *NOT* in use

We are looking for $P(D|E)$.

By using *Bayes' theorem*, we can extrapolate as follows:

$$P(D|E) = \frac{P(E|D) P(D)}{P(E)}$$

The prior, $P(D)$, is the probability of drug use before we see the outcome of the test, which is 5%. The likelihood, $P(E|D)$, is the probability of a positive test assuming drug use, which is the same thing as the sensitivity of the test. The normalizing constant, $P(E)$, is a little bit trickier.

We have to consider two things – $P(E \text{ and } D)$ as well as $P(E \text{ and } N)$. Basically, we must assume that the test is capable of being incorrect when the user is not using drugs. Check out the following equations:

$$P(E) = P(E \text{ and } D) \text{ or } P(E \text{ and } N)$$

$$P(E) = P(D) P(E|D) + P(N) P(E|N)$$

$$P(E) = .05 * .6 + .95 * .01$$

$$P(E) = 0.0395$$

So, our original equation becomes as follows:

$$P(D|E) = \frac{.6 * .05}{0.0395}$$

$$P(D|E) = .76$$

This means that of the people who test positive for drug use, about a quarter are innocent!

Random variables

A **random variable** uses real numerical values to describe a probabilistic event. In our previous work with variables (both in math and programming), we were used to the fact that a variable takes on a certain value. For example, we might have a right-angled triangle in which we are given the variable n for the hypotenuse, and we must figure out the length of the hypotenuse. We also might have the following, in Python:

```
 $x = 5$ 
```

Both of these variables are equal to one value at a time. In a random variable, we are subject to randomness, which means that our variables' values are, well, just that – variable! They might take on multiple values depending on the environment.

A random variable still, as shown previously, holds a value. The main distinction between variables as we have seen them and a random variable is the fact that a random variable's value may change, depending on the situation.

However, if a random variable can have many values, how do we keep track of them all? Each value that a random variable might take on is associated with a percentage, and for each value, there is a single probability that the variable will be that value.

With a random variable, we can also obtain the probability distribution of a random variable, which gives the variable's possible values and their probabilities.

Written out, we generally use single capital letters (mostly the specific letter X) to denote random variables. For example, we might have the following:

- X : The outcome of a dice roll
- Y : The revenue earned by a company this year
- Z : The score of an applicant on an interview coding quiz (0–100%)

Effectively, a random variable is a function that maps values from the sample space of an event (the set of all possible outcomes) to a probability value (between 0 and 1). Think about the event as being expressed as follows:

$$f(event) = probability$$

The function assigns a probability to each individual option. There are two main types of random variables – **discrete** and **continuous**.

Discrete random variables

A discrete random variable only takes on a countable number of possible values, such as the outcome of a dice roll, as shown here:

X = the outcome of a single dice roll						
Value	X = 1	X = 2	X = 3	X = 4	X = 5	X = 6
Probability	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

Figure 6.2 – Representing a discrete random variable

Note how I use a capital X to define the random variable. This is a common practice. Also, note how the random variable maps a probability to each individual outcome. Random variables have many properties, two of which are their *expected value* and the *variance*. We will use a **probability mass function (PMF)** to describe a discrete random variable.

They take on the following appearance:

$$P(X = x) = PMF$$

So, for a dice roll,

$$P(X = 1) = 1/6 \text{ and } P(X = 5) = 1/6$$

Consider the following examples of discrete variables:

- The likely result of a survey question (for example, on a scale of 1–10)
- Whether the CEO will resign within the year (either true or false)

The expected value of a random variable defines the mean value of a long run of repeated samples of the random variable. This is sometimes called the *mean* of the variable.

For example, refer to the following Python code, which defines the random variable of a dice roll:

```
import random
def random_variable_of_dice_roll():
    return random.randint(1, 7) # a range of (1,7) # includes 1, 2, 3, 4, 5, 6, but NOT 7
```

This function will invoke a random variable and come out with a response. Let's roll 100 dice and average the result, as follows:

```
trials = []
num_trials = 100
for trial in range(num_trials):
    trials.append( random_variable_of_dice_roll() )
print(sum(trials)/float(num_trials))
# == 3.77
```

So, taking 100 dice rolls and averaging them gives us a value of 3.77! Let's try this with a wide variety of trial numbers, as illustrated here:

```
num_trials = range(100,10000, 10)
avgs = []
for num_trial in num_trials:
    trials = []
    for trial in range(1,num_trial):
        trials.append( random_variable_of_dice_roll() )
    avgs.append(sum(trials)/float(num_trial))
plt.plot(num_trials, avgs)
plt.xlabel('Number of Trials')
plt.ylabel("Average")
```

We get the following graph:

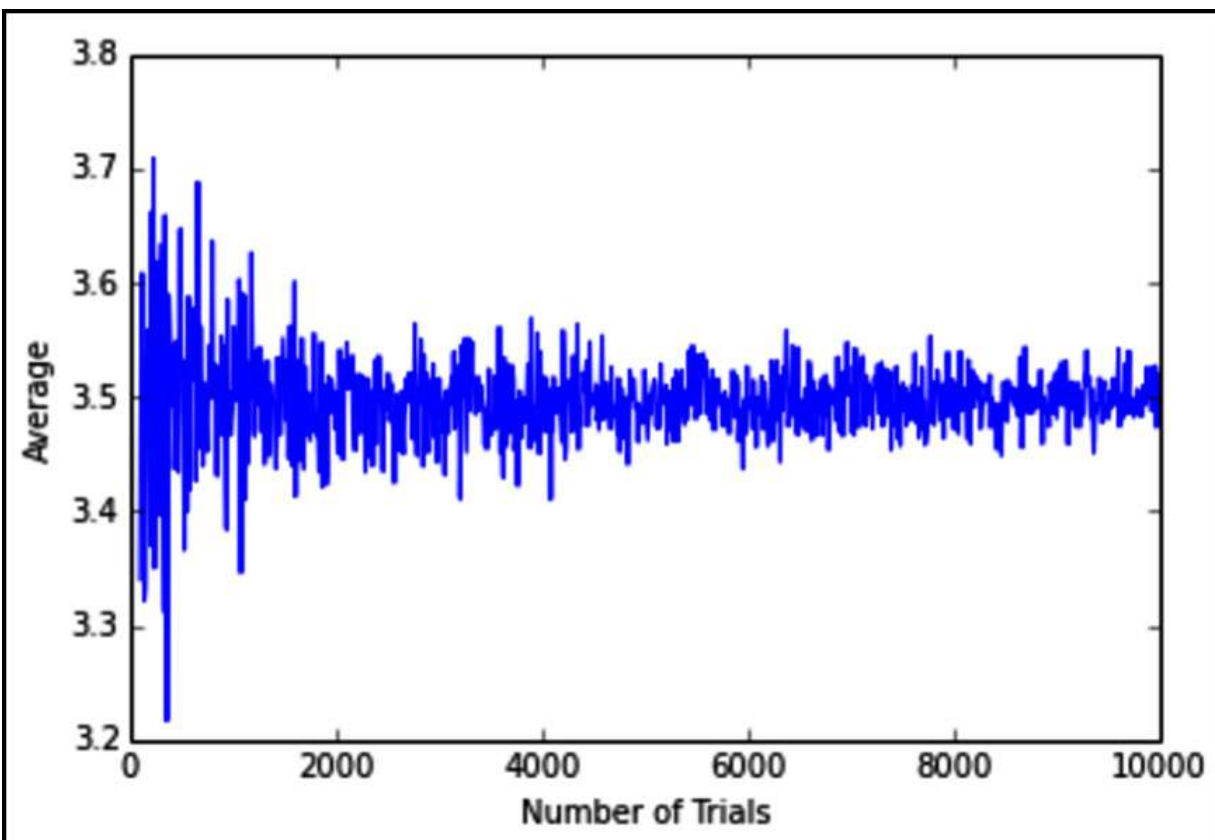


Figure 6.3 – Representing the average of 100 dice rolls

The preceding graph represents the average dice roll as we look at more and more dice rolls. We can see that the average dice roll rapidly approaches 3.5. If we look at the left of the graph, we see that if we only roll a die about 100 times, then we are not guaranteed to get an average dice roll of 3.5. However, if we roll 10,000 dice one after another, we see that we would very likely expect the average dice roll to be about 3.5.

For a discrete random variable, we can also use a simple formula, shown as follows, to calculate the expected value:

$$\text{Expected value} = E[X] = \mu_x = \sum x_i p_i$$

Here, x_i is the i th outcome and p_i is the i th probability.

So, for our dice roll, we can find the exact expected value as follows:

$$\frac{1}{6}(1) + \frac{1}{6}(2) + \frac{1}{6}(3) + \frac{1}{6}(4) + \frac{1}{6}(5) + \frac{1}{6}(6) = 3.5$$

The preceding result shows us that for any given dice roll, we can “expect” a dice roll of 3.5. Now, obviously, that doesn’t make sense because we can’t get a 3.5 on a dice roll, but it does make sense when put in the context of many dice rolls. If you roll 10,000 dice, your average dice roll should approach 3.5, as shown in the graph and code previously.

The average of the expected value of a random variable is generally not enough to grasp the full idea behind the variable. For this reason, we will introduce a new concept, called variance.

The variance of a random variable represents the spread of the variable. It quantifies the variability of the expected value.

The formula for the variance of a discrete random variable is expressed as follows:

$$\text{Variance} = V[X] = \sigma^2_x = \sum (x_i - \mu_x)^2 p_i$$

x_i and p_i represent the same values as before, and μ represents the expected value of the variable. In this formula, I also mentioned the sigma of X . Sigma, in this case, is the standard deviation, which is defined simply as the square root of the variance. Let’s look at a more complicated example of a discrete random variable.

Variance can be thought of as a *give or take* metric. If I say you can expect to win \$100 from a poker hand, you might be very happy. If I append that statement with the additional detail that you might win \$100, give \$80, or take \$80, you now have a wide range of expectations to deal with, which can be frustrating and might make a risk-averse player more wary of joining the game. We can usually say that we have an expected value, give or take the standard deviation.

Consider that your team measures the success of a new product on a **Likert scale** – that is, as being in one of five categories, where a value of 0 represents a complete failure and 4 represents a great success. They estimate that a new project has the following chances of success (shown in *Figure 6.4*), based on user testing and the preliminary results of the performance of the product.

We first have to define our random variable. Let the X random variable represent the success of our product. X is indeed a discrete random variable because the X variable can only take on one of five options – 0, 1, 2, 3, or 4.

The following is the probability distribution of our random variable, X . Note how we have a column for each potential outcome of X , and following each outcome, we have the probability that that particular outcome will be achieved:

Value	$X = 0$	$X = 1$	$X = 2$	$X = 3$	$X = 4$
Probability	0.02	0.07	0.25	0.4	0.26

Figure 6.4 – The probability distribution of our random variable

For example, the project has a 2% chance of failing completely and a 26% chance of being a great success! We can calculate our expected value as follows:

$$E[X] = 0(0.02) + 1(0.07) + 2(0.25) + 3(0.4) + 4(0.26) = 2.81$$

This number means that the manager can expect a success of about 2.81 with this project. Now, by itself, that number is not very useful. Perhaps, if given several products to choose from, an expected value might be a way to compare the potential successes of several products. However, in this case, when we have just one product to evaluate, we will need more. Now, let's check the variance, as shown here:

$$\text{Variance} = V[X] = \sum (x_i - \mu)^2 p_i = (0 - 2.81)^2(0.02) + (1 - 2.81)^2(0.07) + (2 - 2.81)^2(0.25) + (3 - 2.81)^2(0.4) + (4 - 2.81)^2(0.26) = .93$$

Now that we have both the standard deviation and the expected value of the score of the project, let's try to summarize our results. We could say that our project will have an expected score of 2.81, plus or minus 0.96, meaning that we can expect something between 1.85 and 3.77.

So, one way we can address this project is that it is probably going to have a success rating of 2.81, give or take about a point.

You might be thinking, wow, Sinan – so at best, the project will be a 3.8, and at worst it will be a 1.8? Not quite.

It might be better than a 4, and it might also be worse than a 1.8. To take this one step further, let's calculate the following:

$$P(X \geq 3)$$

First, take a minute and convince yourself that you can read that formula to yourself. What am I asking when I ask for $P(X \geq 3)$? Honestly, take a minute and figure it out.

$P(X \geq 3)$ is the probability that our random variable will take on a value at least as big as 3. In other words, what is the chance that our product will have a success rating of 3 or higher? To calculate this, we can calculate the following:

$$P(X \geq 3) = P(X = 3) + P(X = 4) = .66 = 66\%$$

This means that we have a 66% chance that our product will rate as either a 3 or a 4.

Another way to calculate this would be the conjugate way, as shown here:

$$P(X \geq 3) = 1 - P(X < 3)$$

Again, take a moment to convince yourself that this formula holds up. I am claiming that to find the probability that the product will be rated at least a 3 is the same as 1, minus the probability that the product will receive a rating below 3. If this is true, then the two events ($X \geq 3$ and $X < 3$) must complement one another.

This is obviously true! The product can be either of the following two options:

- Be rated 3 or above
- Be rated below a 3

Let's check our math:

$$P(X < 3) = P(X = 0) + P(X = 1) + P(X = 2)$$

$$= 0.02 + 0.07 + 0.25$$

$$= .0341 - P(X < 3)$$

$$= 1 - .34$$

$$= .66$$

$$= P(x \geq 3)$$

It checks out!

Types of discrete random variables

We can get a better idea of how random variables work in practice by looking at specific types of random variables. These specific types of random variables model different types of situations and end up revealing much simpler calculations for very complex event modeling.

Binomial random variables

The first type of discrete random variable we will look at is called a **binomial random variable**. With a binomial random variable, we look at a setting in which a single event happens over and over, and we try to count the number of times the result is positive.

Before we can understand the random variable itself, we must look at the conditions in which it is even appropriate.

A binomial setting has the following four conditions:

- The possible outcomes are either success or failure
- The outcomes of trials cannot affect the outcome of another trial
- The number of trials was set (a fixed sample size)

- The chance of success of each trial must always be p

A binomial random variable is a discrete random variable, X , that counts the number of successes in a binomial setting. The parameters are $n = \text{the number of trials}$ and $p = \text{the chance of success of each trial}$.

Example – fundraising meetings

In this example, a start-up is taking 20 **Venture Capital (VC)** meetings to fund and count the number of offers they receive.

The **probability mass function (PMF)** for a binomial random variable is as follows:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Here,

$$\binom{n}{k} = \text{the binomial coefficient} = \frac{n!}{(n-k)!k!}$$

Example – restaurant openings

In this example, a new restaurant in a town has a 20% chance of surviving its first year. If 14 restaurants open this year, find the probability that exactly four restaurants survive their first year of being open to the public.

First, we should prove that this is a binomial setting:

- The possible outcomes are either success or failure (the restaurants either survive or not)
- The outcomes of trials cannot affect the outcome of another trial (assume that the opening of one restaurant doesn't affect another restaurant's opening and survival)
- The number of trials was set (14 restaurants opened)
- The chance of success of each trial must always be p (we assume that it is always 20%)

Here, we have our two parameters of $n = 14$ and $p = 0.2$. So, we can now plug these numbers into our binomial formula, as shown here:

$$P(X = 4) = \binom{14}{4} \cdot 0.2^4 \cdot 0.8^{10} = .17$$

So, we have a 17% chance that exactly four of these restaurants will be open after a year.

Example – blood types

In this example, a couple has a 25% chance of having a child with type O blood. What is the chance that three of their five kids have type O blood?

Let X = the number of children with type O blood with $n = 5$ and $p = 0.25$, as shown here:

$$P(X = 3) = 10(0.25)^3(0.75)^{5-3} = 10(.25)^3(0.75)^2 = 0.087$$

We can calculate this probability for the values of 0, 1, 2, 3, 4, and 5 to get a sense of the probability distribution:

value x_i	0	1	2	3	4	5
Probability	0.23730	0.39551	0.26367	0.08789	0.01465	0.00098

Figure 6.5 – The probability for the values of 0, 1, 2, 3, 4, and 5

From here, we can calculate an expected value and the variance of this variable:

$$\text{Expected Value} = E[X] = \mu_x = \sum x_i p_i = 1.25$$

$$\text{Variance} = V[X] = \sigma_x^2 = \sum (x_i - \mu_x)^2 p_i = 0.9375$$

So, this family can expect to have probably one or two kids with type O blood!

What if we want to know the probability that at least three of their kids have type O blood? To know the probability that at least three of their kids have type O blood, we can use the following formula for discrete random variables:

$$P(x \geq 3) = P(X = 3) + P(X = 4) + P(X = 5) = .00098 + .01465 + .08789 = 0.103$$

So, there is about a 10% chance that three of their kids have type O blood.

SHORTCUTS TO BINOMIAL EXPECTED VALUES AND VARIANCE

Binomial random variables have special calculations for the exact values of the expected values and variance. If X is a binomial random variable, then we get the following:

$$E(X) = np$$

$$V(X) = np(1 - p)$$

For our preceding example, we can use the following formulas to calculate an exact expected value and variance:

$$E(X) = .25(5) = 1.25$$

$$V(X) = 1.25(.75) = 0.9375$$

A binomial random variable is a discrete random variable that counts the number of successes in a binomial setting. It is used in a wide variety of data-driven experiments, such as counting the number of people who will sign up for a website given a chance of conversion, or even, at a simple level, predicting stock price movements given a chance of decline (don't worry – we will apply much more sophisticated models to predict the stock market later).

Geometric random variables

The second discrete random variable we will take a look at is called a **geometric random variable**. It is actually quite similar to the binomial random variable in the way that we are concerned with a setting, in which a single event occurs over and over. However, in the case of a geometric setting, the major difference is that we are not fixing the sample size.

We are not going into exactly 20 VC meetings as a start-up, nor are we having exactly five kids. Instead, in a geometric setting, we are modeling the number of trials we will need to see before we obtain even a single success. Specifically, a geometric setting has the following four conditions:

- The possible outcomes are either a success or failure
- The outcomes of trials cannot affect the outcome of another trial
- The number of trials was not set
- The chance of success of each trial must always be p

Note that these are the exact same conditions as a binomial variable, except for the third condition.

A **geometric random variable** is a discrete random variable, X , that counts the number of trials needed to obtain one success. The parameters are $p = \text{the chance of success of each trial}$ and $(1 - p) = \text{the chance of failure of each trial}$.

To transform the previous binomial examples into geometric examples, we might do the following:

- Count the number of VC meetings that a start-up must take in order to get their first yes
- Count the number of coin flips needed in order to get a head (yes, I know it's boring, but it's a solid example!)

The formula for the PMF is as follows:

$$P(X = x) = (1 - p)^{x-1}$$

Both the binomial and geometric settings involve outcomes that are either successes or failures. The big difference is that binomial random variables have a fixed number of trials, denoted as n . Geometric random variables do not have a fixed number of trials. Instead, geometric random variables model the number of samples needed in order to obtain the first successful trial, whatever success might mean in those experimental conditions.

Example – weather

In this example, there is a 34% chance that it will rain on any day in April. Find the probability that the first day of rain in April will occur on April 4.

Let X = the number of days until it rains (success) with $p = 0.34$ and $(1 - p) = 0.66$. So then, the probability that it will rain by April 4 is as follows:

$$P(X \leq 4) = P(1) + P(2) + P(3) + P(4) = .34 + .22 + .14 + .09 = .79$$

So, there is an 80% chance that the first rain of the month will happen within the first four days.

SHORTCUTS TO GEOMETRIC EXPECTED VALUES AND VARIANCE

Geometric random variables also have special calculations for the exact values of the expected values and variance. If X is a geometric random variable, then we get the following:

$$E(X) = 1/p$$

$$V(X) = (1 - p)/p^2$$

Poisson random variable

The third and last specific example of a discrete random variable is a Poisson random variable.

To understand why we would need this random variable, imagine that an event that we wish to model has a small probability of happening and that we wish to count the number of times that the event occurs in a certain time frame. If we have an idea of the average number of occurrences, μ , over a specific period of time, given from past instances, then the Poisson random variable, denoted by $X = Poi(\mu)$, counts the total number of occurrences of the event during that given time period.

In other words, the Poisson distribution is a discrete probability distribution that counts the number of events that occur in a given interval of time.

Consider the following examples of Poisson random variables:

- Finding the probability of having a certain number of visitors on your site within an hour, knowing the past performance of the site
- Estimating the number of car crashes at an intersection, based on past police reports

If we let $X = \text{the number of events in a given interval}$, and the average number of events per interval is the λ number, then the probability of observing X events in a given interval is given by the following formula:

$$P(X = x) = \frac{e^{-\lambda} \lambda^x}{x!}$$

Here, e = Euler's constant (2.718....).

Example – a call center

The number of calls arriving at your call center follows a Poisson distribution at the rate of five calls per hour. What is the probability that exactly six calls will come in between 10 and 11 p.m.?

To set up this example, let's write out our given information. Let X be the number of calls that arrive between 10 and 11 p.m. This is our Poisson random variable, with the mean $\lambda = 5$. The mean is 5 because we are using 5 as the expected value of the number of calls to come in at this time. This number could have come from the previous work on estimating the number of calls that come in every

hour, or that specifically come in after 10 p.m. The main point is that we do have some idea of how many calls should be coming in, and then we use that information to create our *Poisson random variable*, using it to make predictions.

Continuing with our example, we have the following:

$$P(X = 6) = 0.146$$

This means that there is about a 14.6% chance that exactly six calls will come in between 10 and 11 p.m.

SHORTCUTS TO POISSON EXPECTED VALUES AND VARIANCE

Poisson random variables also have special calculations for the exact values of the expected values and variance. If X is a Poisson random variable with the mean, then we get the following:

$$E(X) = \lambda$$

$$V(X) = \lambda$$

This is actually interesting because both the expected value and the variance are the same number, and that number is simply the given parameter! Now that we've seen three examples of discrete random variables, we must take a look at the other type of random variable, called the continuous random variable.

Continuous random variables

Switching gears entirely, unlike a discrete random variable, a continuous random variable can take on an *infinite* number of possible values, not just a few countable ones. We call the functions that describe the distribution density curves instead of probability mass functions.

Consider the following examples of continuous variables:

- The length of a sales representative's phone call (not the number of calls)
- The actual amount of oil in a drum marked 20 gallons (not the number of oil drums)

If X is a continuous random variable, then there is a function, $f(x)$, for any constants, a and b :

$$P(a \leq X \leq b) = \int_a^b f(x) dx$$

The preceding $f(x)$ function is known as the **probability density function (PDF)**. The PDF is the continuous random variable version of the PMF for discrete random variables.

The most important continuous distribution is the **standard normal distribution**. You have, no doubt, either heard of the normal distribution or dealt with it. The idea behind it is quite simple. The PDF of this distribution is as follows:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Here, μ is the mean of the variable, and σ is the standard deviation. This might look confusing, but let's graph it in Python, with a mean of 0 and a standard deviation of 1, as shown here:

```
import numpy as np
import matplotlib.pyplot as plt
def normal_pdf(x, mu = 0, sigma = 1):
    return (1./np.sqrt(2*3.14 * sigma**2)) * np.exp(-(x-mu)**2 / (2.* sigma**2))
x_values = np.linspace(-5,5,100)
y_values = [normal_pdf(x) for x in x_values] plt.plot(x_values, y_values)
```

We get this graph:

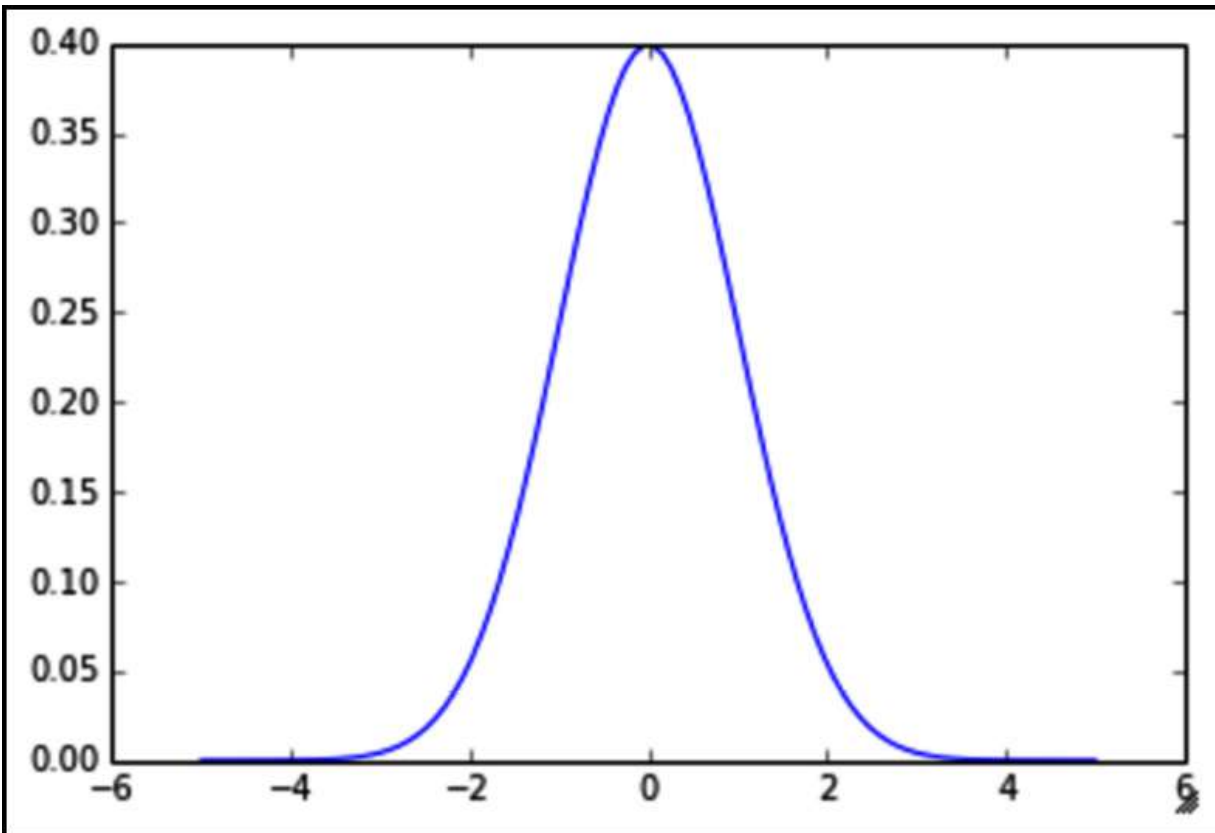


Figure 6.6 – Representing a mean of 0 and a standard deviation of 1

This reveals the all-too-familiar bell curve. Note that the graph is symmetrical around the $x = 0$ line. Let's try changing some of the parameters. First, let's try with $\mu = 5$:

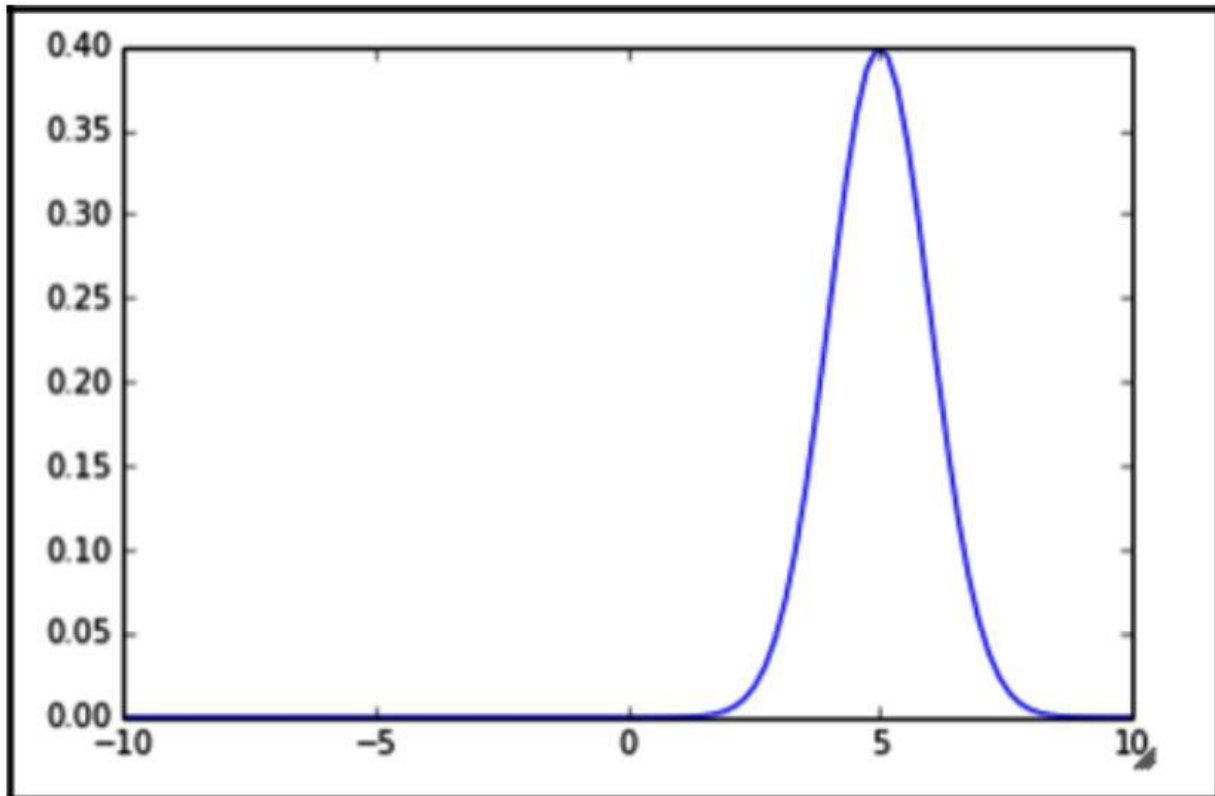


Figure 6.7 – Representing the all-too-familiar bell curve

Next, let's try with the value $\sigma = 5$:

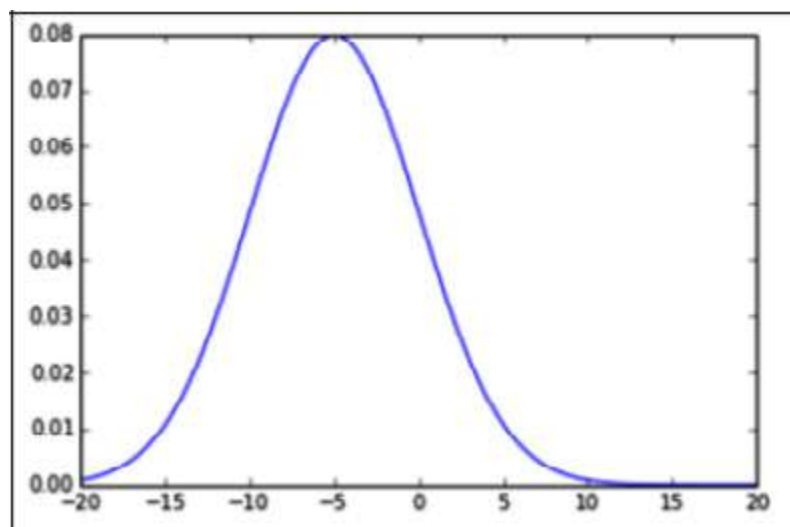


Figure 6.8 – Representing the value $\sigma = 5$

Lastly, we will try with the values $\mu = 5$, $\sigma = 5$:

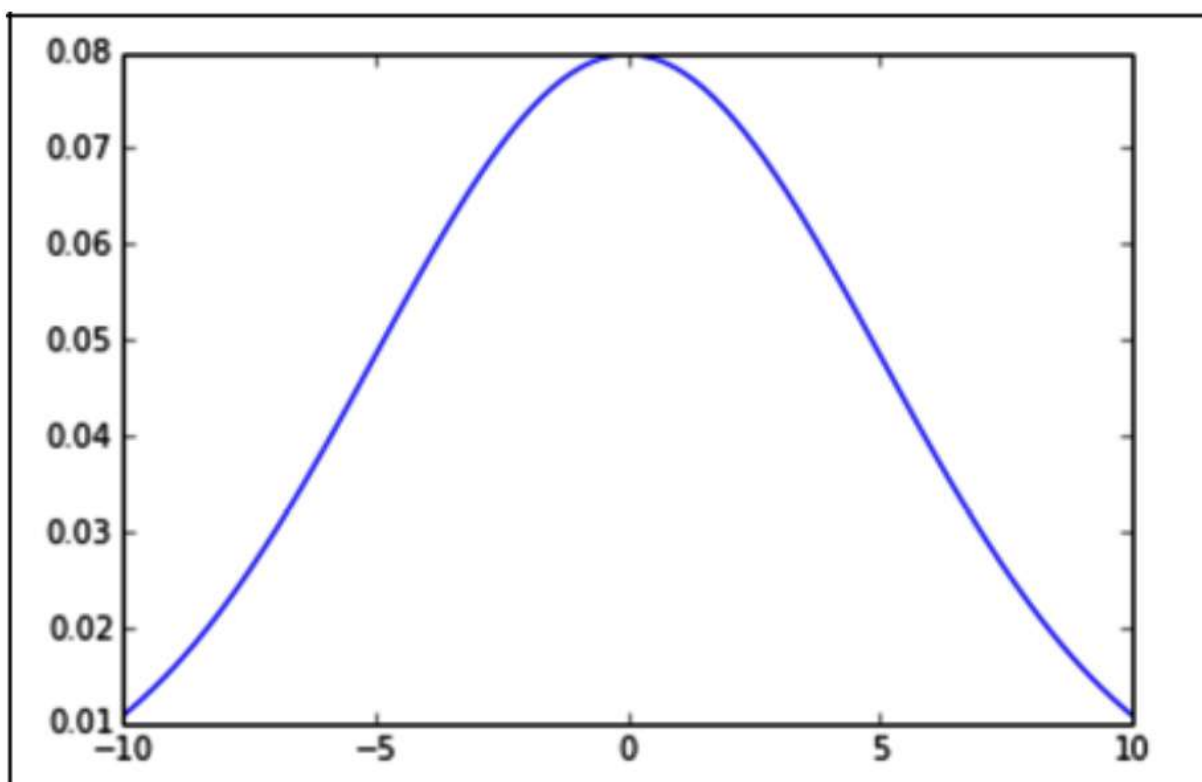


Figure 6.9 – A graph representing the values $\mu = 5$, $\sigma = 5$

In all the graphs, we have the standard bell shape that we are all familiar with, but as we change our parameters, we can see that the bell might get skinnier, thicker, or move from left to right.

In the following chapters, which focus on statistics, we will make much more use of the normal distribution as it applies to statistical thinking.

Summary

Probability as a field works to explain our random and chaotic world. Using the basic laws of probability, we can model real-life events that involve randomness. We can use random variables to represent values that may take on several values, and we can use the probability mass or density functions to compare product lines or look at the test results.

We have seen some of the more complicated uses of probability in prediction. Using random variables and Bayes' theorem are excellent ways to assign probabilities to real-life situations.

The next two chapters focus on statistical thinking. Like probability, these chapters will use mathematical formulas to model real-world events. The main difference, however, will be the terminology we use to describe the world and the way we model different types of events. In these upcoming chapters, we will attempt to model entire populations of data points based solely on a sample.

We will revisit many concepts in probability to make sense of statistical theorems, as they are closely linked, and both are important mathematical concepts in the realm of data science.

What Are the Chances? An Introduction to Statistics

This chapter will focus on the statistical knowledge required by any aspiring data scientist.

We will explore ways of sampling and obtaining data without being affected by bias and then use measures of statistics to quantify and visualize our data. Using the z-score and the empirical rule, we will see how we can standardize data for the purposes of both graphing and interpretability.

In this chapter, we will look at the following topics:

- How to obtain and sample data
- The measures of center, variance, and relative standing
- Normalization of data using the z-score
- The empirical rule

What are statistics?

This might seem like an odd question to ask, but I am frequently surprised by the number of people who cannot answer this simple and yet powerful question: what are statistics? Statistics are the numbers you always see on the news and in the paper. Statistics are useful when trying to prove a point or trying to scare someone, but what are they?

To answer this question, we need to back up for a minute and talk about why we even measure them in the first place. The goal of this field is to try to explain and model the world around us. To do that, we have to take a look at the population.

We can define a **population** as the entire pool of subjects of an experiment or a model.

Essentially, your population is who you care about. Who are you trying to talk about? If you are trying to test whether smoking leads to heart disease, your population would be the smokers of the world. If you are trying to study teenage drinking problems, your population would be all teenagers.

Now, imagine that you want to ask a question about your population. For example, if your population is all of your employees (assume that you have over 1,000 employees), perhaps you want to know what percentage of them enjoy traveling. The question is called a **parameter** – a numerical measurement describing a characteristic of a population. For example, if you ask all 1,000 employees and 100 of them enjoy traveling, the rate of travel enjoyment is 10%. The parameter here is 10%.

You probably can't ask every single employee whether they enjoy traveling. What if you have over 10,000 employees? It would be very difficult to track everyone down in order to get your answer. When this happens, it's impossible to figure out this parameter. In this case, we can *estimate* the parameter.

First, we will take a *sample* of the population. We can define a sample of a population as a subset (not necessarily random) of the population. Perhaps ask 200 of the 1,000 employees you have. Of these 200, suppose 26 enjoy traveling, making the rate 13%. Here, 13% is not a parameter because we didn't get a chance to ask everyone. This 13% is an estimate of a parameter. Do you know what that's called?

That's right, a **statistic**!

We can define a statistic as a numerical measurement describing a characteristic of a sample of a population. A statistic is just an estimation of a parameter. It is a number that attempts to describe an entire population by describing a subset of that population. This is necessary because you can never hope to give a survey to every single teenager or to every single smoker in the world. That's what the field of statistics is all about: taking samples of populations and running tests on these samples.

So, the next time you are given a statistic, just remember that number only represents a sample of that population, not the entire pool of subjects.

How do we obtain and sample data?

If statistics is about taking samples of populations, it must be very important to know how we obtain these samples, and you'd be correct. Let's focus on just a few of the many ways of obtaining and sampling data.

Obtaining data

There are two main ways of collecting data for our analysis: **observational** and **experimentation**. Both these ways have their pros and cons, of course. They each produce different types of behavior and, therefore, warrant different types of analysis.

Observational

We might obtain data through observational means, which consists of measuring specific characteristics but not attempting to modify the subjects being studied. For example, if you had tracking software on your website that observes users' behavior on the website, such as length of time spent on certain pages and the rate of clicking on ads, all the while not affecting the user's experience, then that would be an observational study.

This is one of the most common ways to get data because it's just plain easy. All you have to do is observe and collect data. Observational studies are also limited in the types of data that can be collected. This is because the observer (you) is not in control of the environment. You may only watch and collect natural behavior. If you are looking to induce a certain type of behavior, an observational study would not be useful.

Experimental

An **experiment** consists of a treatment and the observation of its effect on the subjects. Subjects in an experiment are called **experimental units**. This is usually how most scientific labs collect data. They will put people into two or more groups (usually just two) and call them the control and the experimental groups.

The control group is exposed to a certain environment and then observed. The experimental group is exposed to a different environment and then observed. The experimenter then aggregates data from both groups and makes a decision about which environment was more favorable (favorable is a quality that the experimenter gets to decide).

In a marketing example, imagine that we expose half of our users to a certain landing page with certain images and a certain style (*website A*), and we measure whether or not they sign up for the service. Then, we expose the other half to a different landing page, different images, and different styles (*website B*) and again measure whether or not they sign up. We can then decide which of the two sites performed better and should be used going further. This, specifically, is called an *A/B test*. Let's see an example in Python!

Let's suppose we run the preceding test and obtain the following results as a list of lists:

```
results = [ ['A', 1], ['B', 1], ['A', 0], ['A', 0] ... ]
```

Here, each object in the list result represents a subject (person). Each person then has the following two attributes:

- Which website they were exposed to, represented by a single character
- Whether or not they converted (0 for no and 1 for yes)

We can then aggregate and come up with the following results table:

```
users_exposed_to_A = []
users_exposed_to_B = []
# create two lists to hold the results of each individual website
```

Once we create these two lists that will eventually hold each individual conversion value as Booleans (0 or 1), we will iterate all of our results of the test and add them to the appropriate list, as shown:

```
for website, converted in results: # iterate through the results
    will look something like website == 'A' and converted == 0 if website == 'A':
    users_exposed_to_A.append(converted) elif website == 'B':
    users_exposed_to_B.append(converted)
```

Now, each list contains a series of 1 and 0 values.

IMPORTANT NOTE

Remember that 1 represents a user actually converting to the site after seeing that web page, and 0 represents a user seeing the page and leaving before signing up/converting.

To get the total number of people exposed to website A, we can use the `len()` feature.

Let's use Python to illustrate the elements of our two lists:

```
len(users_exposed_to_A) == 188 #number of people exposed to website A
len(users_exposed_to_B) == 158 #number of people exposed to website B
```

To count the number of people who converted, we can use the sum of the list, as shown:

```
sum(users_exposed_to_A) == 54 # people converted from website A
sum(users_exposed_to_B) == 48 # people converted from website B
```

If we subtract the length of the lists and the sum of the list, we are left with the number of people who did *not* convert for each site, as illustrated:

```
len(users_exposed_to_A) - sum(users_exposed_to_A) == 134 # did not convert from
website A
len(users_exposed_to_B) - sum(users_exposed_to_B) == 110 # did not convert from
website B
```

We can aggregate and summarize our results in the following table, which represents our experiment on website conversion testing:

	Did not sign up	Signed up
Website A	134	54
Website B	110	48

Table 7.1 – The results of our A/B test

We can quickly drum up some descriptive statistics. We can say that the website conversion rates for the two websites are as follows:

- Conversion for website A: $54 / (154 + 34) = .288$
- Conversion for website B: $48 / (110 + 48) = .3$

Not much difference, but different nonetheless. Even though B has a higher conversion rate, can we really say that version B significantly converts better? Not yet. To test the *statistical significance* of

such a result, a hypothesis test should be used. These tests will be covered in depth in the next chapter, where we will revisit this exact same example and finish it using a proper statistical test.

Sampling data

Remember how statistics are the result of measuring a sample of a population. Well, we should talk about two very common ways to decide who gets the honor of being in the sample that we measure. We will discuss the main type of sampling, called random sampling, which is the most common way to decide our sample sizes and our sample members.

Probability sampling

Probability sampling is a way of sampling from a population, in which every person has a known probability of being chosen, but that probability *might* be a different value than another user has. The simplest (and probably the most common) probability sampling method is **random sampling**.

Random sampling

Suppose that we are running an A/B test and we need to figure out who will be in Group A and who will be in Group B. The following are three suggestions from our data team:

- *Separate users based on location*: Users on the West Coast are placed in Group A, while users on the East Coast are placed in Group B
- *Separate users based on the time of day they visit the site*: Users who visit between 7 p.m. and 4 a.m. are group A, while the rest are placed in group B
- *Make it completely random*: Every new user has a 50/50 chance of being placed in either group

The first two are valid options for choosing samples and are fairly simple to implement, but they both have one fundamental flaw: they are both at risk of introducing a sampling bias.

A sampling bias occurs when the way the sample is obtained systematically favors some other outcome over the target outcome. This can occur in various ways, such as using an unrepresentative sample, selecting participants based on certain criteria, or using biased sampling methods. When a sample is biased, it can lead to incorrect or misleading conclusions about the population being studied, and therefore it is important to ensure that sampling methods are appropriate and unbiased in order to obtain accurate and reliable results.

It is not difficult to see why choosing the first or second option from the preceding list might introduce bias. If we choose our groups based on where they live or what time they log in, we are priming our experiment incorrectly and, now, we have much less control over the results.

Specifically, we are at risk of introducing a **confounding factor** into our analysis, which is bad news.

A **confounding factor** is a variable that we are not directly measuring but that connects the variables that are being measured. Basically, a confounding factor is like the missing element in our analysis that

is invisible but affects our results.

In this case, the first option is not taking into account the potential confounding factor of *geographical taste*. For example, if website A is unappealing, in general, to West Coast users, it will affect your results drastically.

Similarly, the second option might introduce a temporal (time-based) confounding factor. What if website B is better viewed in a night-time environment (which was reserved for A), and users react negatively to the style purely because the time of day when they view it? These are both factors that we want to avoid, so we should go with the third option, which is a random sample.

IMPORTANT NOTE

While sampling bias can cause confounding, it is a different concept than confounding. The first and second options were both sampling biases because we chose the samples incorrectly, and were also examples of confounding factors because there was a third variable in each case that affected our decision.

A random sample is chosen such that every single member of a population has an equal chance of being chosen as any other member.

This is probably one of the easiest and most convenient ways to decide who will be a part of your sample. Everyone has the exact same chance of being in any particular group. Random sampling is an effective way of reducing the impact of confounding factors.

Unequal probability sampling

Recall that I previously said that a probability sampling might have different probabilities for different potential sample members. But what if this actually introduced problems? Suppose we are interested in measuring the happiness level of our employees. We already know that we can't ask every single member of staff because that would be silly and exhausting. So, we need to take a sample. Our data team suggests random sampling, and at first, everyone high-fives because they feel very smart and statistical. But then someone asks a seemingly harmless question: does anyone know the percentage of men/women who work here?

The high-fives stop and the room goes silent.

This question is extremely important because sex is likely to be a confounding factor. The team looks into it and discovers a split of 75% men and 25% women in the company.

This means that if we introduce a random sample, our sample will likely have a similar split and thus favor the results for men and not women. To combat this, we can opt to include more women than men in our survey in order to make the split of our sample less biased toward men.

At first glance, introducing a favoring system in our random sampling seems like a bad idea; however, alleviating unequal sampling and, therefore, working to remove systematic bias among gender, race, disability, and so on is much more pertinent. A simple random sample, where everyone has the same

chance as everyone else, is very likely to drown out the voices and opinions of minority population members. Therefore, it can be okay to introduce such a favoring system in your sampling techniques.

How do we measure statistics?

Once we have our sample, it's time to quantify our results. Suppose we wish to generalize the happiness of our employees or we want to figure out whether salaries in the company are very different from person to person.

These are some common ways of measuring our results.

Measures of center

Measures of the center are how we define the middle, or center, of a dataset. We do this because sometimes we wish to make generalizations about data values. For example, perhaps we're curious about what the average rainfall in Seattle is or what the median height of European males is. It's a way to generalize a large set of data so that it's easier to convey to someone.

A measure of center is a value in the *middle* of a dataset. This can mean different things to different people. Who's to say where the middle of a dataset is? There are so many different ways of defining the center of data. Let's take a look at a few.

The **arithmetic mean** of a dataset is found by adding up all of the values and then dividing it by the number of data values. This is likely the most common way to define the center of data, but can be flawed! Suppose we wish to find the mean of the following numbers:

```
import numpy as np
np.mean([11, 15, 17, 14]) == 14.25
```

Simple enough; our average is 14.25 and all of our values are fairly close to it. But what if we introduce a new value: 31?

```
np.mean([11, 15, 17, 14, 31]) == 17.6
```

This greatly affects the mean because the arithmetic mean is sensitive to outliers. The new value, 31, is almost twice as large as the rest of the numbers, and therefore skews the mean.

Another, and sometimes better, measure of the center is the median. The **median** is the number found in the middle of the dataset when it is sorted in order, as shown:

```
np.median([11, 15, 17, 14]) == 14.5
np.median([11, 15, 17, 14, 31]) == 15
```

Note how the introduction of 31 using the median did not affect the median of the dataset greatly. This is because the median is less sensitive to outliers.

When working with datasets with many outliers, it is sometimes more useful to use the median of the dataset, while if your data does not have many outliers and the datapoints are mostly close to one another, then the mean is likely a better option.

But how can we tell if the data is spread out? Well, we will have to introduce a new type of statistic.

Measures of variation

Measures of the center are used to quantify the middle of the data, but now we will explore ways of measuring how to *spread out* the data we collect is. This is a useful way to identify whether our data has many outliers lurking inside. Let's start with an example.

Imagine that we take a random sample of 24 of our friends on Facebook and record how many friends each of them has on Facebook. Here's the list:

```
friends = [109, 1017, 1127, 418, 625, 957, 89, 950, 946, 797, 981, 125, 455, 731,
1640, 485, 1309, 472, 1132, 1773, 906, 531, 742, 621]
np.mean(friends) == 789.1
```

The average of the values in this list is just over **789**. So, we could say that according to this sample, the average Facebook friend has 789 friends. But what about the person who only has 89 friends or the person who has over 1,600 friends? In fact, not many of these numbers are really that close to 789.

Well, how about we use the median? The median generally is not as affected by outliers:

```
np.median(friends) == 769.5
```

The median is **769.5**, which is fairly close to the mean. Hmm, good try, but still, it doesn't really account for how drastically different a lot of these datapoints are. This is what statisticians call measuring the variation of data. Let's start by introducing the most basic measure of variation: the range. The range is simply the maximum value minus the minimum value, as illustrated:

```
np.max(friends) - np.min(friends) == 1684
```

The range tells us how far away the two most extreme values are. Now, the range isn't typically widely used but it does have some pertinent applications. Sometimes, we wish to know just how spread apart the outliers are. This is most useful in scientific and safety measurements.

Suppose a car company wants to measure how long it takes for an airbag to deploy. Knowing the average of that time is nice, but they also really want to know how spread apart the slowest time is versus the fastest time. This literally could be the difference between life and death.

Shifting back to the Facebook example, **1684** is our range, but I'm not quite sure it's saying very much about our data. Now, let's take a look at the most commonly used measure of variation, the **standard deviation**.

I'm sure many of you have heard this term thrown around a lot and it might even incite a degree of fear, but what does it really mean? In essence, standard deviation, denoted by s when we are working with a sample of a population, measures by how much data values deviate from the arithmetic mean.

It's basically a way to see how spread out the data is. There is a general formula to calculate the standard deviation, which is as follows:

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

Figure 7.1 – Formula for standard deviation

The formula for standard deviation might seem scary at first, but with time and practice, it will become a familiar friend, helping illustrate how spread out data really is.

Let's look at each of the elements in this formula in turn:

- s is our sample's standard deviation
- x is each individual data point
- \bar{x} is the mean of the data
- n is the number of datapoints

Before you freak out and close this book at the seemingly complicated equation, let's break it down. For each value in the sample, we will take that value, subtract the arithmetic mean from it, square the difference, and, once we've added up every single point this way, we will divide the entire thing by n , the number of points in the sample. Finally, we take a square root of everything.

Without going into an in-depth analysis of the formula, think about it this way: it's basically derived from the distance formula. Essentially, what the standard deviation is calculating is a sort of average distance of the data values from the arithmetic mean.

If you take a closer look at the formula, you will see that it actually makes sense:

- By taking $x - \bar{x}$, you are finding the literal difference between the value and the mean of the sample.
- By squaring the result, $(x - \bar{x})^2$, we are putting a greater penalty on outliers because squaring a large error only makes it much larger.
- By dividing by the number of items in the sample, we are taking (literally) the average squared distance between each point and the mean.

- By taking the square root of the answer, we are putting the number in terms that we can understand. For example, by squaring the number of friends minus the mean, we changed our units to friends squared, which makes no sense. Taking the square root puts our units back to just “friends.”

Let’s go back to our Facebook example for a visualization and further explanation of this.

Let’s begin by calculating the standard deviation – a few of them, in this case.

Recall that the arithmetic mean of the data was around **789**, so we’ll use **789** as the mean.

We start by taking the difference between each data value and the mean, squaring it, adding them all up, dividing it by one less than the number of values, and then taking its square root. This would look as follows:

$$s = \sqrt{\frac{(109 - 789)^2 + (1017 - 789)^2 + \dots + (621 - 789)^2}{24}}$$

Figure 7.2 – Representation of an example of calculating the standard deviation of a series of data

An example of calculating the standard deviation of a series of data has us subtracting each item in the list by the average of the list, squaring that sum, adding it all up, dividing by the number of items, and finally, taking the square root of that whole thing.

On the other hand, we can take the Python approach and do all this programmatically (which is usually preferred):

```
np.std(friends) # == 425.2
```

What the number **425** represents is the spread of data. You could say that 425 is a kind of average distance the data values are from the mean. What this means, in simple words, is that this data is pretty spread out.

So, our standard deviation is about **425**. This means that the number of friends that these people have on Facebook doesn’t seem to be close to a single number and that’s quite evident when we plot the data in a bar graph, and also plot the mean as well as the visualizations of the standard deviation. In the following plot, every person will be represented by a single bar in the bar chart, and the height of the bars represent the number of friends that the individuals have:

```
import matplotlib.pyplot as plt
friends = [109, 1017, 1127, 418, 625, 957, 89, 950, 946, 797, 981, 125, 455, 731,
1640, 485, 1309, 472, 1132, 1773, 906, 531, 742, 621]
y_pos = range(len(friends))
plt.bar(y_pos, friends)
plt.plot((0, 25), (789, 789), 'b-')
```

```
plt.plot((0, 25), (789+425, 789+425), 'g-')
plt.plot((0, 25), (789-425, 789-425), 'r-')
```

Here's the chart that we get:

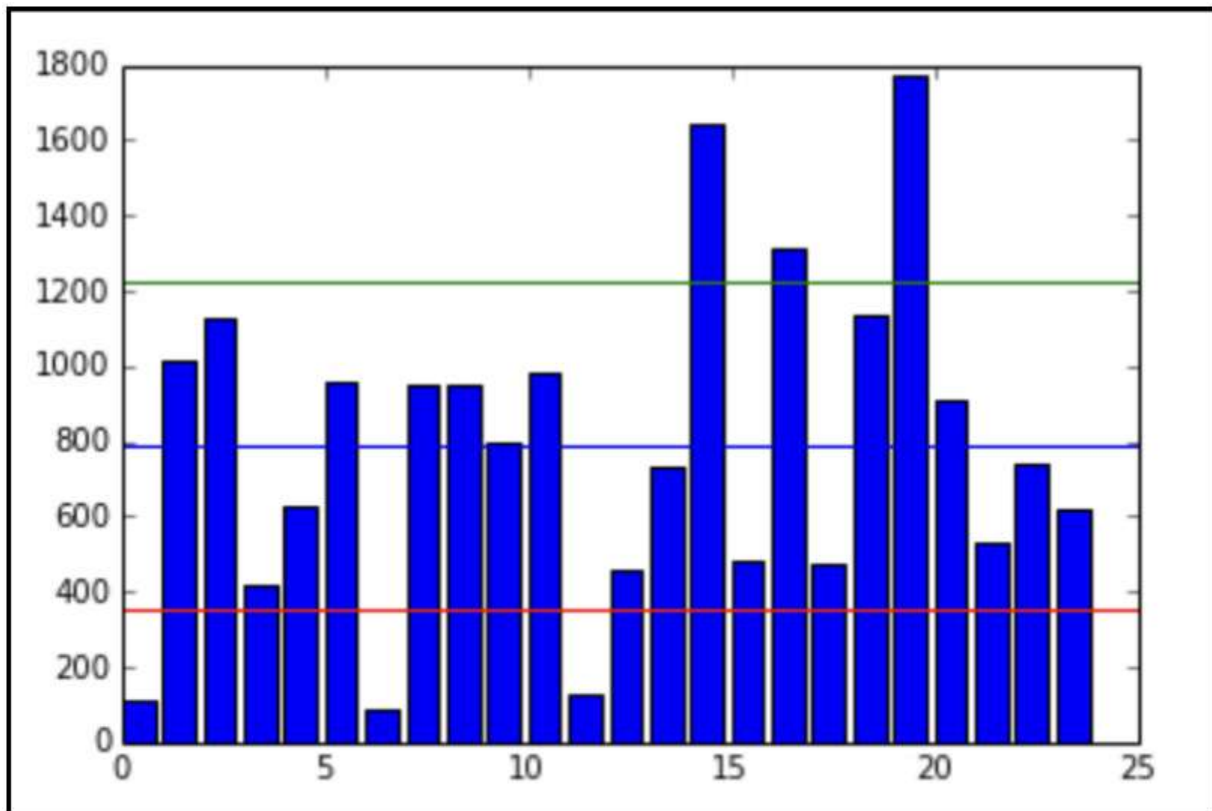


Figure 7.3 – Plotting each of our datapoints as a bar in a bar graph with a line showing the average value (blue, in the middle), the average minus 1 standard deviation (red, below), and the average plus 1 standard deviation (green, the line above)

The blue line in the center is drawn at the mean (789), the red line near the bottom is drawn at the mean minus the standard deviation ($789 - 425 = 364$), and finally, the green line toward the top is drawn at the mean plus the standard deviation ($789 + 425 = 1,214$).

Note how most of the data lives between the green and the red lines while the outliers live outside the lines. There are three people who have friend counts below the red line and three people who have a friend count above the green line. This is a common way to discuss outliers in data – through standard deviations. It's often used as a unit distance. You can say, for example, that in the realm of this "friends" data, the value of 1,214 is "one standard deviation above the mean" or that 1,639 is "two standard deviations above the mean."

It's important to mention that the units for standard deviation are, in fact, the same units as the data's units. So, in this example, we would say that the standard deviation is 425 friends on Facebook.

IMPORTANT NOTE

Another measure of variation is the variance, as described in the previous chapter. The variance is simply the standard deviation squared.

So, now we know that the standard deviation and variance are good for checking how spread out our data is, and that we can use it along with the mean to create a kind of range that a lot of our data lies in. But what if we want to compare the spread of two different datasets, maybe even with completely different units? That's where the coefficient of variation comes into play.

The coefficient of variation

The **coefficient of variation** is defined as the ratio of the data's standard deviation to its mean.

This ratio (which, by the way, is only helpful if we're working in the ratio level of measurement, where the division is allowed and is meaningful) is a way to standardize the standard deviation, which makes it easier to compare across datasets. We use this measure frequently when attempting to compare means, and it spreads across populations that exist at different scales.

Example – employee salaries

If we look at the mean and standard deviation of employees' salaries in the same company but among different departments, we see that, at first glance, it may be tough to compare the standard deviations because they are on such different scales:

Salaries of Company XYZ			
Department	Mean Salary	SD	CoV
Mailroom	\$25,000	\$2,000	8.0%
Human Resources	\$52,000	\$7,000	13.5%
Executive	\$124,000	\$42,000	33.9%

Figure 7.4 – Glancing at the means and standard deviations of salaries across departments

Glancing at the means and standard deviations of salaries across departments can be challenging without scaling them to the same scale as one another using the coefficient of variation (the final column). Only then can we see that the spread of salaries at the executive level is somewhat larger than the spread in the other departments

This is especially true when the mean salary of one department is \$25,000, while another department has a mean salary in the six-figure area.

However, if we look at the last column, which is our coefficient of variation, it becomes clearer that the people in the executive department may be getting paid more but they are also getting wildly different

salaries. This is probably because the CEO is earning way more than an office manager, who is still in the executive department, which makes the data very spread out.

On the other hand, everyone in the mailroom, while not making as much money, is making just about the same as everyone else in the mailroom, which is why their coefficient of variation is only 8%.

With measures of variation, we can begin to answer big questions, such as how to spread out this data or how we can come up with a good range that most of the data falls into.

Measures of relative standing

We can combine both the measures of centers and variations to create measures of relative standing.

Measures of variation measure where particular data values are positioned, relative to the entire dataset.

Let's begin by learning a very important value in statistics, the z-score. The z-score is a way of telling us how far away a single data value is from the mean. Think back to the previous section where I was referring to datapoints being a certain number of standard deviations away from the mean. The z-score of an x data value exactly the calculation of this, the formula for which is as follows:

$$z = \frac{X - \bar{X}}{s}$$

Let's break down this formula:

- X is the data point
- \bar{X} is the mean
- s is the standard deviation

Remember that the standard deviation was (sort of) an average distance of the data from the mean, and the z-score is an individualized value for each particular data point. We can find the z-score of a data value by subtracting it from the mean and dividing it by the standard deviation. The output is the standardized distance a value is from a mean. We use the z-score all over statistics. It is a very effective way of normalizing data that exists on very different scales, and also to put data in the context of its mean.

Let's take our previous data on the numbers of friends on Facebook and standardize the data to the z-score. For each data point, we will find its z-score by applying the preceding formula. We will take each individual, subtract the average-friends number from the value, and divide that by the standard deviation, as follows:

```
z_scores = []
m = np.mean(friends) # average friends on Facebook
s = np.std(friends) # standard deviation friends on Facebook
for friend in friends:
```

```
z = (friend - m)/s# z-score
z_scores.append(z) # make a list of the scores for plotting
```

Now, let's plot these z-scores on a bar chart. The following chart shows the same individuals from our previous example using friends on Facebook, but instead of the bar height revealing the raw number of friends, now each bar is the z-score of the number of friends they have on Facebook. If we plot the z-scores, we'll notice a few things:

```
plt.bar(y_pos, z_scores)
```

We get this chart:

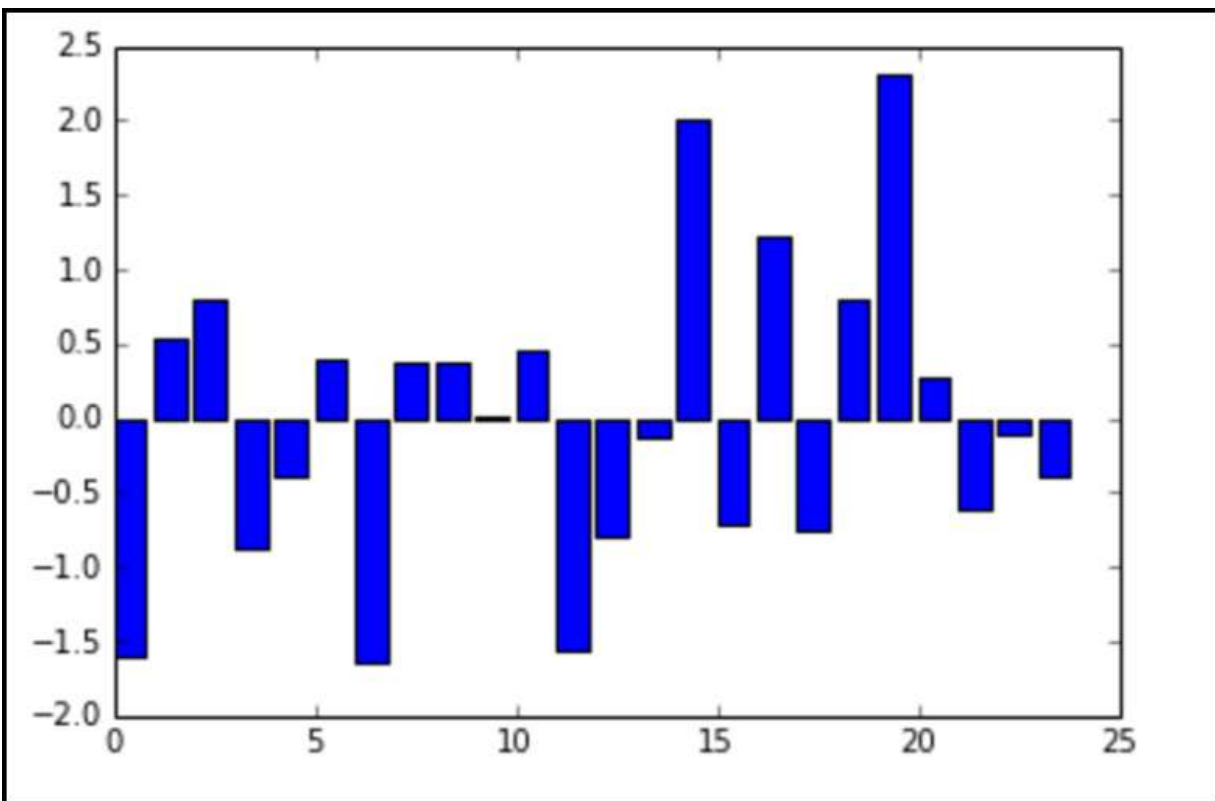


Figure 7.5 – The z scores of our data

The z scores of our data quickly show us which datapoints are below and above the average and by how many standard deviations

We can see that we have negative values (meaning that the data point is below the mean). The bars' lengths no longer represent the raw number of friends, but the degree to which that friend count differs from the mean.

This chart makes it very easy to pick out the individuals with much lower and higher friends on average. For example, the individual at index 0 has fewer friends on average (they had 109 friends where the average was 789).

What if we want to graph the standard deviations? Recall that we earlier plotted three horizontal lines: one at the mean, one at the mean plus the standard deviation ($x+s$), and one at the mean minus the standard deviation ($x-s$).

If we plug these values into the formula for the z-score, we get the following:

$$Z \text{ score of } \bar{x} = \frac{\bar{x} - \bar{x}}{s} = \frac{0}{s} = 0$$

$$Z \text{ score of } (x + s) = \frac{(x + s) - \bar{x}}{s} = \frac{s}{s} = 1$$

$$Z \text{ score of } (x - s) = \frac{(x - s) - \bar{x}}{s} = \frac{-s}{s} = -1$$

This is no coincidence! When we standardize the data using the z-score, our standard deviations become the metric of choice. Let's plot a new graph with the standard deviations added:

```
plt.bar(y_pos, z_scores)
plt.plot((0, 25), (1, 1), 'g-')
plt.plot((0, 25), (0, 0), 'b-')
plt.plot((0, 25), (-1, -1), 'r-')
```

The preceding code is adding the following three lines:

- A blue line at $y = 0$ that represents zero standard deviations away from the mean (which is on the x axis)
- A green line that represents one standard deviation above the mean
- A red line that represents one standard deviation below the mean

Let's look at the graph we get as a result:

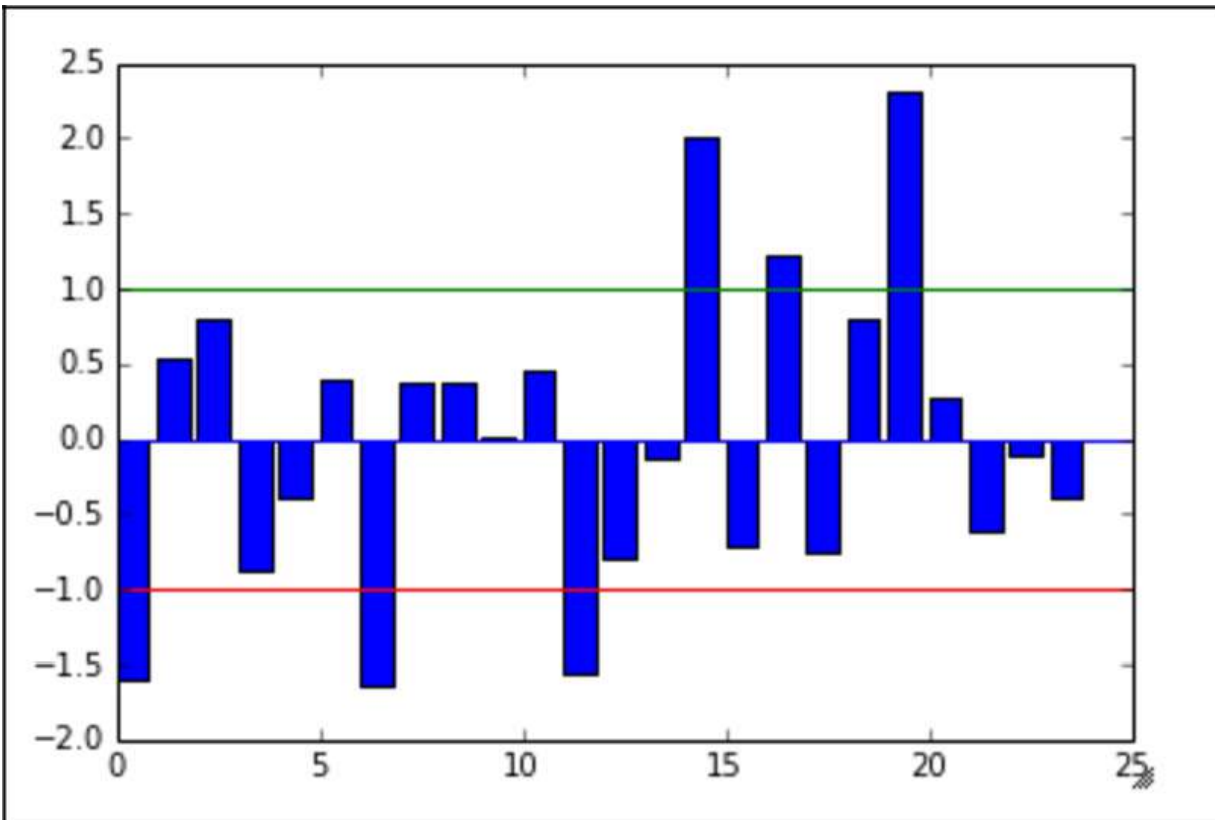


Figure 7.6 – Our z-score graph with lines at 1 and -1

Our z-score graph with lines at 1 and -1 parallels *Figure 7.3*, where we had values within 1 standard deviation of the mean shown between the green and red (top and bottom) lines

The colors of the lines match up with the lines drawn in the earlier graph of the raw friend count. If you look carefully, you'll see the same people still fall outside of the green and the red lines. Namely, the same three people still fall below the red (lower) line, and the same three people fall above the green (upper) line.

Z-scores are an effective way to *standardize* data. This means that we can put the entire set on the same scale. For example, if we also measure each person's general happiness scale (which is between 0 and 1), we might have a dataset similar to the following dataset:

```
friends = [109, 1017, 1127, 418, 625, 957, 89, 950, 946, 797, 981, 125, 455, 731,
1640, 485, 1309, 472, 1132, 1773, 906, 531, 742, 621]
happiness = [.8, .6, .3, .6, .6, .4, .8, .5, .4, .3, .3, .6, .2, .8, 1, .6, .2, .7,
.5, .3, .1, 0, .3, 1]
import pandas as pd
df = pd.DataFrame({'friends':friends, 'happiness':happiness})
df.head()
```

We get this table:

	friends	happiness
0	109	0.8
1	1017	0.6
2	1127	0.3
3	418	0.6
4	625	0.6

Figure 7.7 – Representing data with two columns, one for the number of friends, the other for happiness measures between 0 and 1

These datapoints are on two different dimensions, each with a very different scale. The friend count can be in the thousands while our happiness score is stuck between 0 and 1.

To remedy this (and for some statistical/machine learning modeling, this practice will become essential), we can simply standardize the dataset using a prebuilt standardization package in scikit-learn, as follows:

```
from sklearn import preprocessing
df_scaled = pd.DataFrame(preprocessing.scale(df), columns = ['friends_scaled',
'happiness_scaled'])
df_scaled.head()
```

This code will scale both the friends and happiness columns simultaneously, thus revealing the z-score for each column. It is important to note that when running the preceding code, the preprocessing module in **sklearn** does the following things separately for each column:

- Finding the mean of the column
- Finding the standard deviation of the column
- Applying the z-score function to each element in the column

The result is two columns, as shown, that exist on the same scale as each other even if they were not previously:

	friends_scaled	happiness_scaled
0	-1.599495	1.153223
1	0.536040	0.394939
2	0.794750	-0.742486
3	-0.872755	0.394939
4	-0.385909	0.394939

Figure 7.8 – Using the z-score, standardizing each column to be on the same scale

Each of these numbers now has the same unit – *standard deviations*.

Now, we can plot friends and happiness on the same scale and the graph will at least be readable:

```
df_scaled.plot(kind='scatter', x = 'friends_scaled', y = 'happiness_scaled')
```

The preceding code gives us this graph:

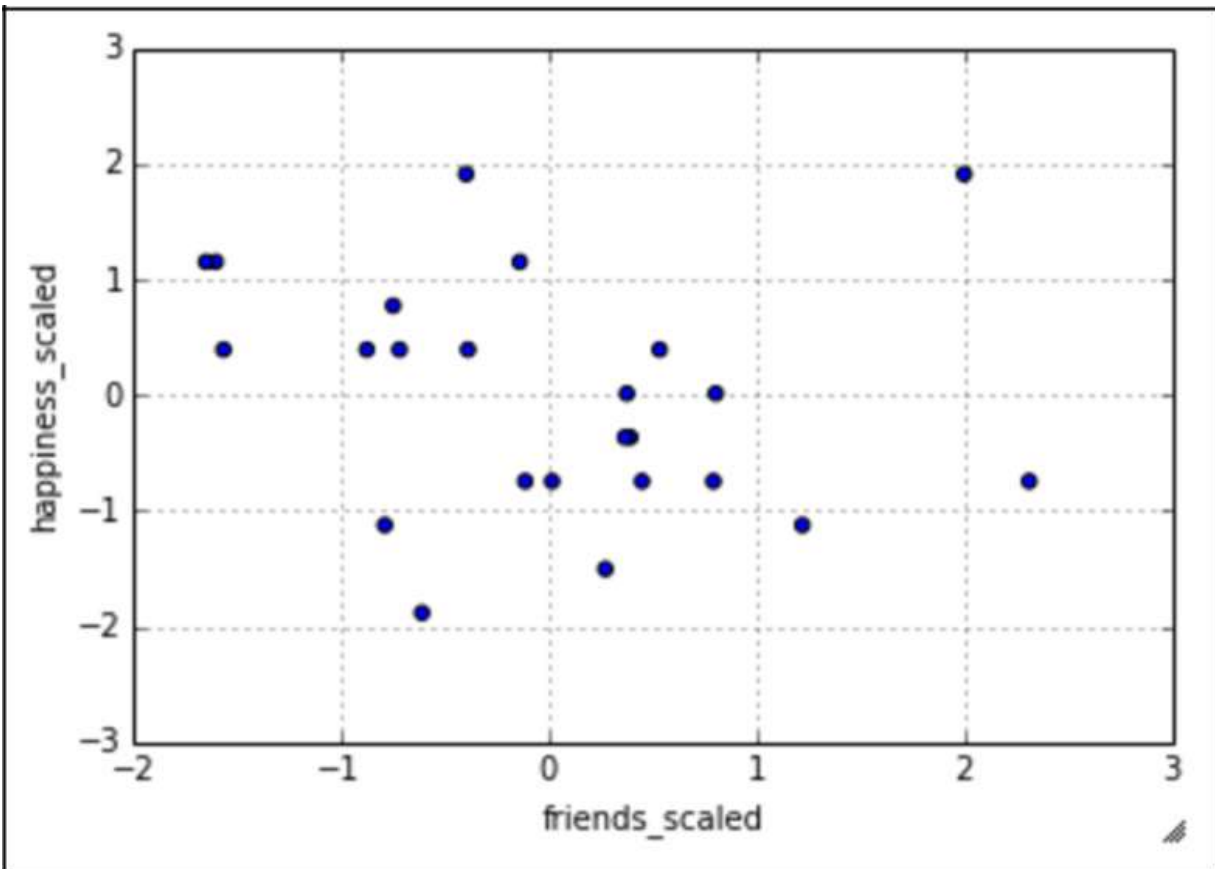


Figure 7.9 – Plotting scaled data

Plotting scaled data is often much easier to read than unscaled data because it's much easier to compare data when it is all in the same unit

Now, our data is standardized to the z-score and this scatter plot is fairly easily interpretable! In later chapters, this practice of standardization will not only make our data more interpretable but will also be an essential part of our model optimization. Many machine learning algorithms require us to have standardized columns as they are reliant on the notion of scale.

The insightful part – correlations in data

Throughout this book, we discuss the difference between having data and having actionable insights about your data. Having data is only one step in achieving a successful data science operation. Being able to obtain, clean, and plot data helps to tell the story that the data has to offer but cannot reveal the moral of the story. In order to take this entire example one step further, we will look at the relationship between having friends on Facebook and happiness.

In subsequent chapters, we will look at a specific machine learning algorithm that attempts to find relationships between quantitative features, called **linear regression**, but we do not have to wait until then to begin to form hypotheses. We have a sample of people, a measure of their online social presence, and their reported happiness. The question of the day here is this: can we find a relationship between the number of friends on Facebook and overall happiness?

Now, obviously, this is a big question and should be treated respectfully. Experiments to answer this question should be conducted in a laboratory setting, but we can begin to form a hypothesis about this question. Given the nature of our data, we really only have the following three options for a hypothesis:

- There is a positive association between the number of online friends and happiness (as one goes up, so does the other)
- There is a negative association between them (as the number of friends goes up, your happiness goes down)
- There is no association between the variables (as one changes, the other doesn't really change that much)

Can we use basic statistics to form a hypothesis about this question? I say we can! But first, we must introduce a concept called **correlation**.

Correlation coefficients are a quantitative measure that describes the strength of association/relationship between two variables.

The correlation between the two sets of data tells us about how they move together. The hope is to understand whether changing one value might help us predict the other. This concept is not only interesting but also one of the core assumptions that many machine learning models make on data. For many prediction algorithms to work, they rely on the fact that there is some sort of relationship between the variables being looked at. The learning algorithms then exploit this relationship in order to make accurate predictions.

A few things to note about the standard correlation coefficient:

- It will lie between -1 and 1
- The greater the absolute value (closer to -1 or 1), the stronger the relationship between the variables:
 - The strongest correlation is -1 or 1
 - The weakest correlation is 0

- A positive correlation means that as one variable increases, the other one tends to increase as well
- A negative correlation means that as one variable increases, the other one tends to decrease

We can use pandas to quickly show us correlation coefficients between every feature and every other feature in the DataFrame, as illustrated here:

```
df.corr(). # correlation between variables
```

We get this table:

	friends	happiness
friends	1.000000	-0.216199
happiness	-0.216199	1.000000

Figure 7.10 – Correlation between friends and happiness

The correlation between friends and happiness is about -0.2, which says that, according to this data, an increase in 1 friend tends to lead to a reduction of happiness by 0.2 units

The preceding table shows the correlation between **friends** and **happiness**. Note the first two things:

- The diagonal of the matrix is filled with positive 1s. This is because they represent the correlation between the variable and itself, which, of course, forms a perfect line, making the correlation perfectly positive!
- The matrix is symmetrical across the diagonal. This is true for any correlation matrix made in pandas.

There are a few caveats to trusting the correlation coefficient. One is that, in general, a correlation will attempt to measure a linear relationship between variables. This means that if there is no visible correlation revealed by this measure, it does not mean that there is no relationship between the variables, only that there is no line of best fit that goes through the lines easily. There might be a *non-linear* relationship that defines the two variables.

It is important to realize that causation is not implied by correlation. Just because there is a weak negative correlation between these two variables does not necessarily mean that your overall happiness decreases as the number of friends you keep on Facebook goes up. This causation must be tested further and, in later chapters, we will attempt to do just that.

To sum up, we can use correlation to make hypotheses about the relationship between variables, but we will need to use more sophisticated statistical methods and machine learning algorithms to solidify these

assumptions and hypotheses.

The empirical rule

Recall that a normal distribution is defined as having a specific probability distribution that resembles a bell curve. In statistics, we love it when our data behaves *normally*. For example, we may have data that resembles a normal distribution, like so:

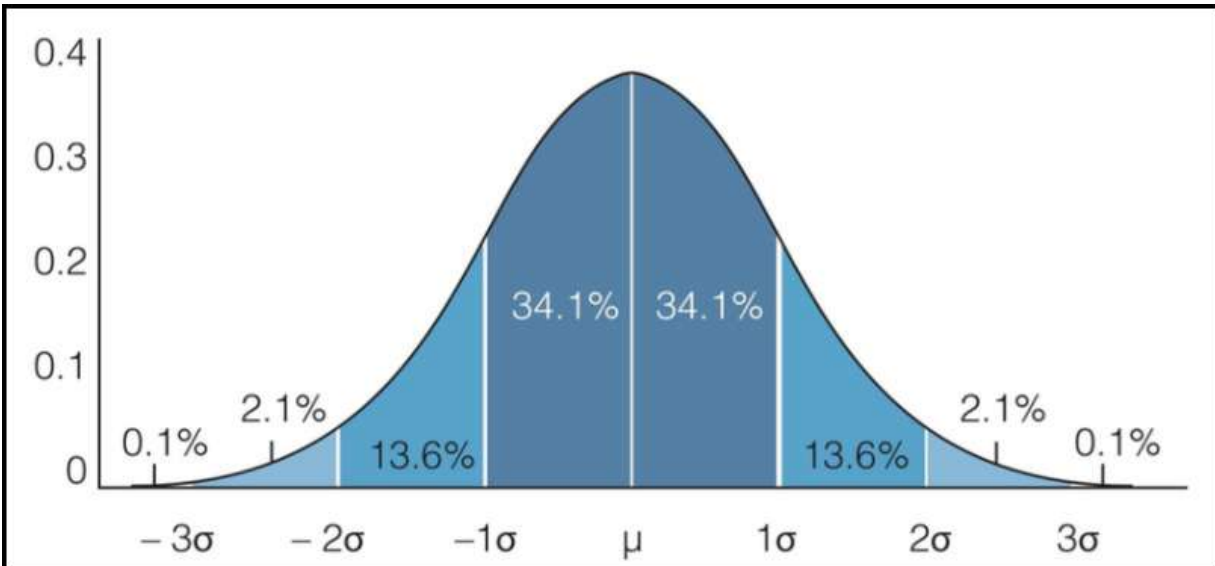


Figure 7.11 – Graphical representation of normal distribution

The normal distribution serves as a guiding line for many branches of statistics and the basis for many statistical tests. Shown here, data that follows this distribution lets us “expect” a certain number of datapoints to live within 1, 2, and 3 standard deviations from the mean.

The **empirical rule** states that we can expect a certain amount of data to live between sets of standard deviations. Specifically, the empirical rule states the following for data that is distributed normally:

- About 68% of the data falls within 1 standard deviation
- About 95% of the data falls within 2 standard deviations
- About 99.7% of the data falls within 3 standard deviations

For example, let’s see whether our Facebook friends’ data holds up to this. Let’s use our DataFrame to find the percentage of people that fall within 1, 2, and 3 standard deviations of the mean, as shown:

```
finding the percentage of people within one standard deviation of the mean
within_1_std = df_scaled[(df_scaled['friends_scaled'] <= 1) &
(df_scaled['friends_scaled'] >= -1)].shape[0] / within_1_std /
float(df_scaled.shape[0])
0.75
finding the percentage of people within two standard deviations of the mean
within_2_std = df_scaled[(df_scaled['friends_scaled'] <= 2) &
```



```

(df_scaled['friends_scaled'] >= -2)].shape[0] within_2_std /
float(df_scaled.shape[0])
0.916
finding the percentage of people within three standard deviations of the mean
within_3_std = df_scaled[(df_scaled['friends_scaled'] <= 3) &
(df_scaled['friends_scaled'] >= -3)].shape[0] within_3_std /
float(df_scaled.shape[0])
1.0

```

We can see that our data does seem to follow the empirical rule. About 75% of the people are within a single standard deviation of the mean. About 92% of the people are within two standard deviations, and all of them are within three standard deviations.

Example – exam scores

Let's say that we're measuring the scores of an exam and the scores generally have a bell-shaped normal distribution. The average result on the exam was 84% and the standard deviation was 6%. We can say the following, with approximate certainty:

- About 68% of the class scored between 78% and 90% because 78 is 6 units below 84, and 90 is 6 units above 84
- If we were asked what percentage of the class scored between 72% and 96%, we would notice that 72 is 2 standard deviations below the mean, and 96 is 2 standard deviations above the mean, so the empirical rule tells us that about 95% of the class scored in that range

However, not all data is normally distributed, so we can't always use the empirical rule. We have another theorem that helps us analyze any kind of distribution. In the next chapter, we will go into depth about when we can assume a normal distribution. This is because many statistical tests and hypotheses require the underlying data to come from a normally distributed population.

IMPORTANT NOTE

Previously, when we standardized our data to the z-score, we did not require an assumption of normal distribution.

Summary

In this chapter, we covered many of the basic statistics required for most data scientists – everything from how we obtain/sample data to how to standardize data according to the z-score and applications of the empirical rule. We also reviewed how to take samples for data analysis. In addition, we reviewed various statistical measures, such as the mean and standard deviation, that help describe data.

In the next chapter, we will look at much more advanced applications of statistics. One thing that we will consider is how to use hypothesis tests on data that we can assume to be normal. As we use these tests, we will also quantify our errors and identify the best practices to solve these errors.

Advanced Statistics

In this chapter, we are concerned with making inferences about entire populations based on certain samples of data. We will be using hypothesis tests along with different estimation tests in order to gain a better understanding of populations, given samples of data.

The key topics that we will cover in this chapter are as follows:

- Point estimates
- Confidence intervals
- The central limit theorem
- Hypothesis testing

Understanding point estimates

Recall that, in the previous chapter, we mentioned how difficult it is to obtain a population parameter; so, we had to use sample data to calculate a statistic that was an estimate of a parameter. When we make these estimates, we call them **point estimates**.

A point estimate is an estimate of a population parameter based on sample data.

We use point estimates to estimate things such as population means, variances, and other statistics. To obtain these estimates, we simply apply the function that we wish to measure for our population to a sample of the data. For example, suppose there is a company of 9,000 employees and we are interested in ascertaining the average length of breaks taken by employees in a single day. As we probably cannot ask every single person, we will take a sample of the 9,000 people and take a mean of the sample. This sample mean will be our point estimate. We will use the probability distribution, known as the Poisson distribution, to randomly generate 9,000 answers to the question *For how many minutes in a day do you usually take breaks?* This will represent our *population*. Remember, from [Chapter 6, Advanced Probability](#), that the Poisson random variable is used when we know the average value of an event and wish to model a distribution around it.

IMPORTANT NOTE

I set a random seed in order to encourage reproducibility (this allows us to get the same random numbers each time).

We will take a sample of 100 employees (using the Python random sample method) and find a point estimate of a mean (called a **sample mean**).

IMPORTANT NOTE

Note that this is just over 1% of our population.

Compare our sample mean (the mean of the sample of 100 employees) to our population mean.

Let's take a look at the following code:

```
import numpy as np
import pandas as pd
from scipy import stats
from scipy.stats import poisson
np.random.seed(1234)
# represents 3000 people who take about a 60 minute break
long_breaks = stats.poisson.rvs(mu=60, size=3000)
```

The `long_breaks` variable represents 3,000 answers to the question, *How many minutes on average do you take breaks for?*, and these answers will be on the longer side. Let's see a visualization of this distribution, as follows:

```
pd.Series(long_breaks).hist()
```

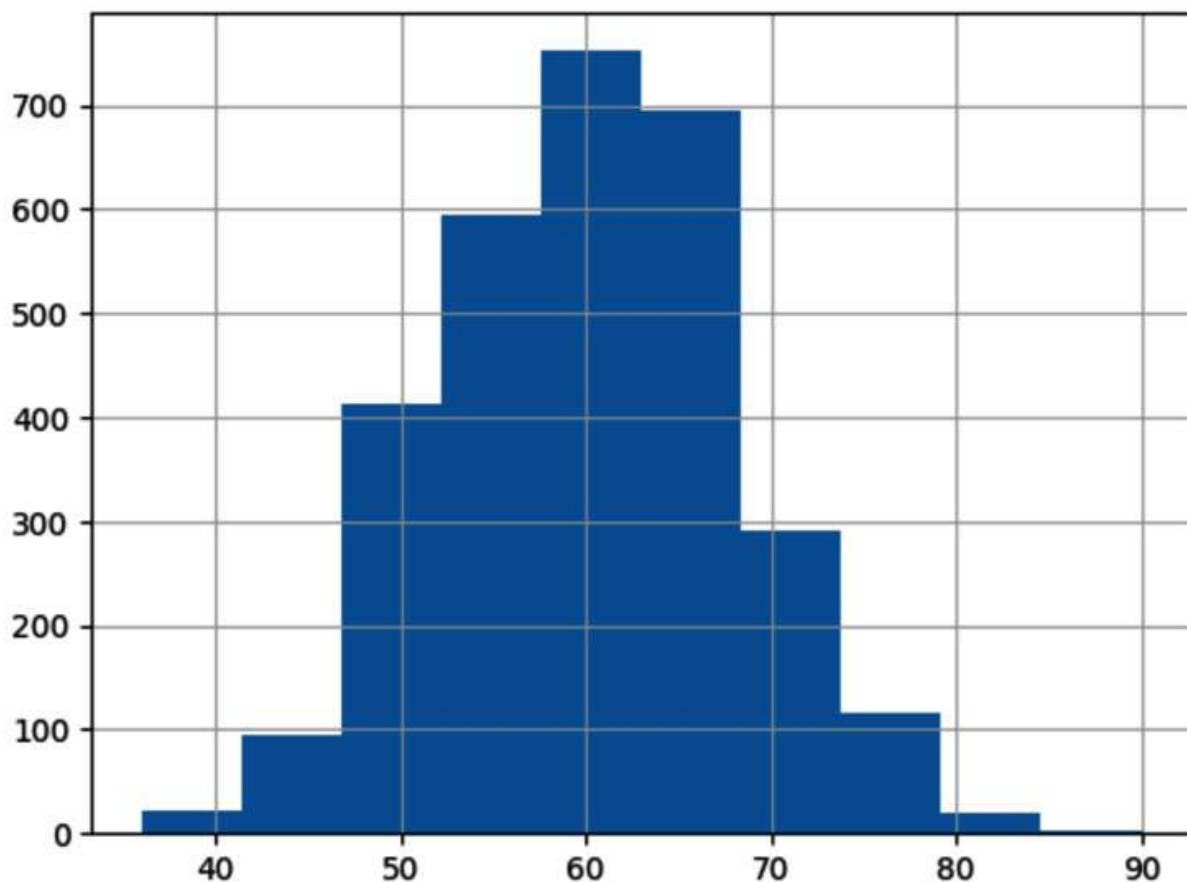


Figure 8.1 – The histogram of our longer break times with a known average of 60 minutes

We can see that our average of 60 minutes is to the left of the distribution. Also, because we only sampled 3,000 people, our bars are at their highest at around 700–800 people.

Now, let's model 6,000 people who take, on average, about 15 minutes' worth of breaks.

Let's again use the Poisson distribution to simulate 6,000 people, as shown:

```
# represents 6000 people who take about a 15 minute break
short_breaks = stats.poisson.rvs(mu=15, size=6000)
pd.Series(short_breaks).hist()
```

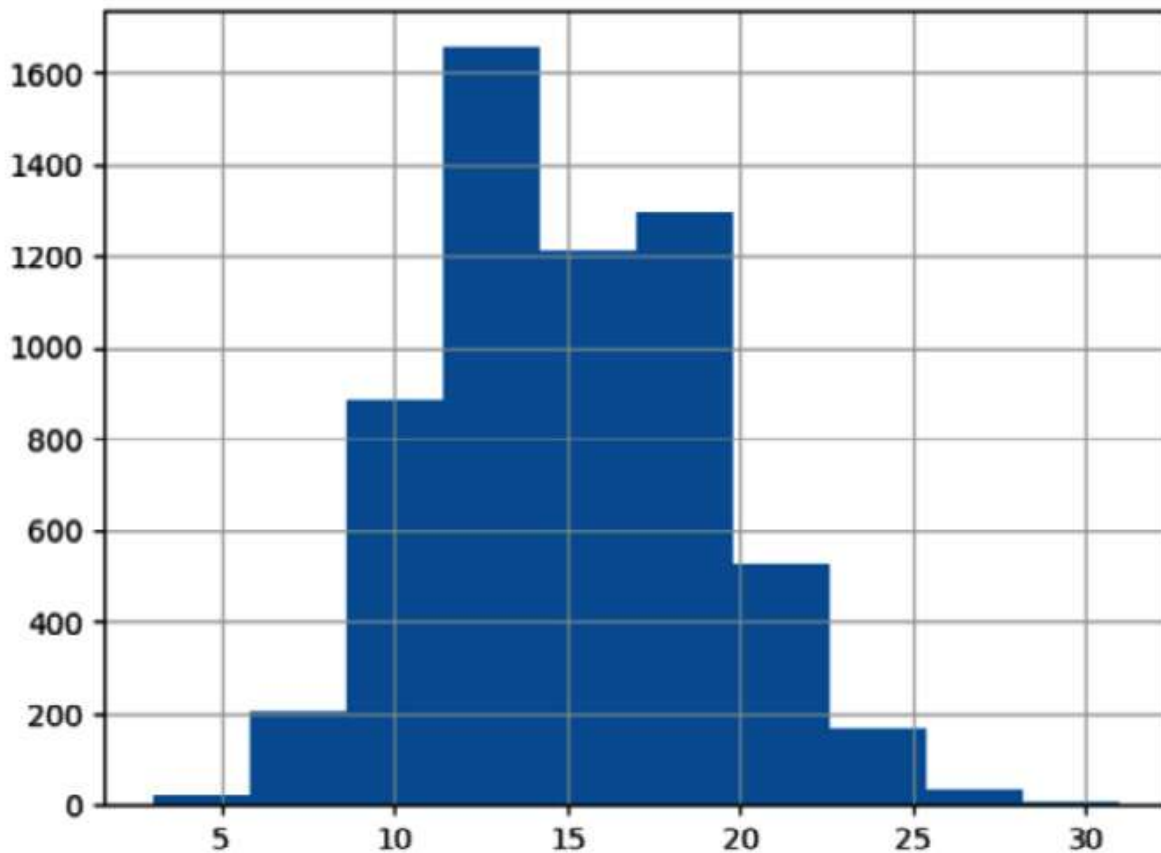


Figure 8.2 – The histogram of our shorter break times with a known average of 15 minutes

Okay, so we have a distribution for the people who take longer breaks and a distribution for the people who take shorter breaks. Again, note how our average break length of 15 minutes falls to the left-hand side of the distribution, and note that the tallest bar is for about 1,600 people:

```
breaks = np.concatenate((long_breaks, short_breaks))
# put the two arrays together to get our "population" of 9000 people
```

The `breaks` variable is the amalgamation of all the 9,000 employees, both long and short break-takers. Let's see the entire distribution of people in a single visualization:

```
pd.Series(breaks).hist()
```

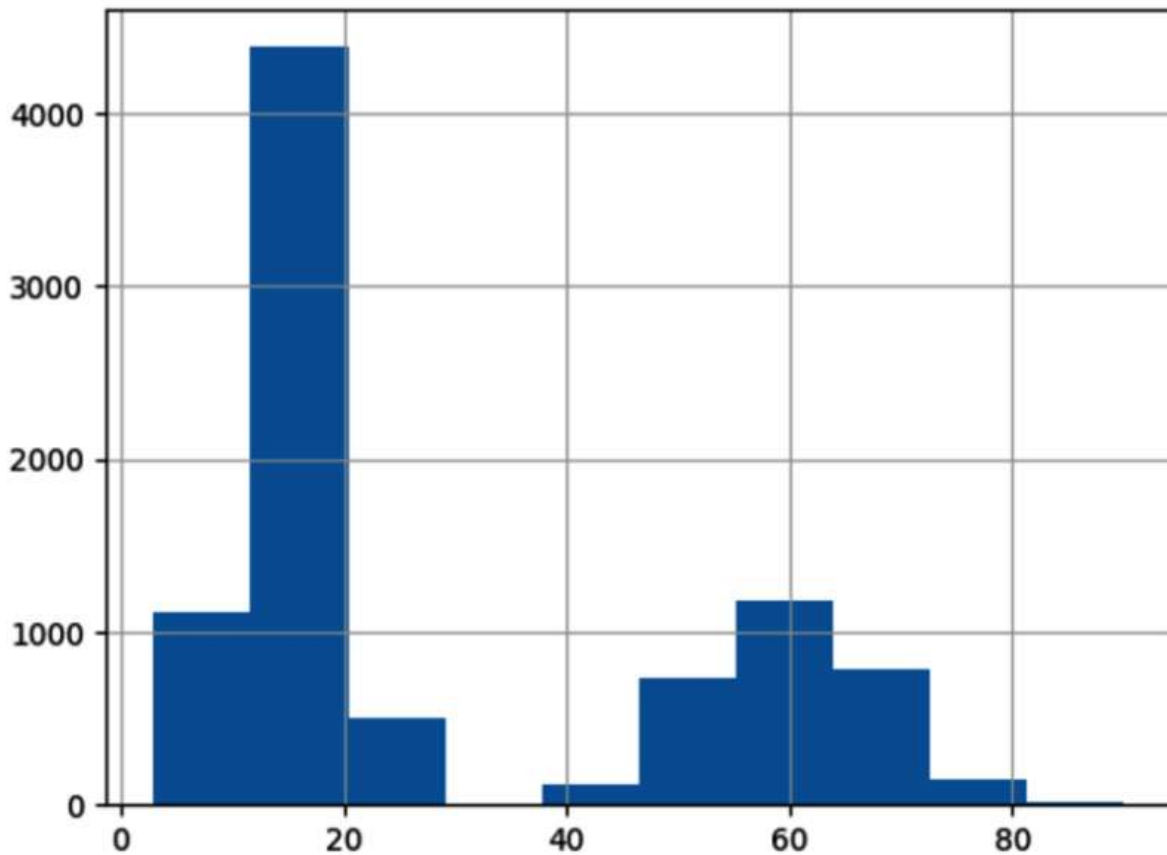


Figure 8.3 – The histogram of our two types of break-takers

We can see we have two humps. On the left, we have our larger hump of people who take about a 15-minute break, and on the right, we have a smaller hump of people who take longer breaks. Later on, we will investigate this graph further.

We can find the total average break length by running the following code:

```
breaks.mean()  
# 29.99 minutes is our parameter.
```

Our average company break length is about 40 minutes. Remember that our population is the entire company's employee size of 9,000 people, and our parameter is 40 minutes. In the real world, our goal would be to estimate the population parameter because we would not have the resources to ask every single employee in a survey their average break length for many reasons. Instead, we will use a point estimate.

So, to make our point, we want to simulate a world where we ask 100 random people about the length of their breaks. To do this, let's take a random sample of 100 employees out of the 9,000 employees we simulated, as shown:

```
sample_breaks = np.random.choice(a = breaks, size=100)
```

```
# taking a sample of 100 employees
```

Now, let's take the mean of the sample and subtract it from the population mean and see how far off we were:

```
breaks.mean() - sample_breaks.mean()
# difference between means is 0.699 minutes, not bad!
```

This is extremely interesting because, with only about 1% of our population (100 out of 9,000), we were able to get within 1 minute of our population parameter and get a very accurate estimate of our population mean. Not bad!

We calculated a point estimate for the mean, but we can also do this for proportion parameters. By proportion, I am referring to a ratio of two quantitative values.

Let's suppose that in a company of 10,000 people, our employees are 20% white, 10% black, 10% Hispanic, 30% Asian, and 30% identify as other. We will take a sample of 1,000 employees and see whether their race proportions are similar:

```
employee_races = (["white"]*2000) + (["black"]*1000) + \
(["hispanic"]*1000) + (["asian"]*3000) + \
(["other"]*3000)
```

employee_races represents our employee population. For example, in our company of 10,000 people, 2,000 people are white (20%) and 3,000 people are Asian (30%).

Let's take a random sample of 1,000 people, as shown:

```
import random
demo_sample = random.sample(employee_races, 1000) # Sample 1000 value
for race in set(demo_sample):print( race + " proportion estimate:" )
print( demo_sample.count(race)/1000. )
```

The output obtained would be as follows:

```
hispanic proportion estimate:
0.103
white proportion estimate:
0.192
other proportion estimate:
0.288
black proportion estimate:
0.1
asian proportion estimate:
0.317
```

We can see that the race proportion estimates are very close to the underlying population's proportions. For example, we got 10.3% for **hispanic** in our sample and the population proportion for **hispanic** was 10%.

Sampling distributions

In [Chapter 7, What Are the Chances? An Introduction to Statistics](#), we mentioned how much we love it when data follows the normal distribution. One of the reasons for this is that many statistical tests (including the ones we will use in this chapter) rely on data that follows a normal pattern, and for the most part, a lot of real-world data is not normal (surprised?). Take our employee break data, for example—you might think I was just being fancy creating data using the Poisson distribution, but I had a reason for this. I specifically wanted non-normal data, as shown:

```
pd.DataFrame(breaks).hist(bins=50, range=(5, 100))
```

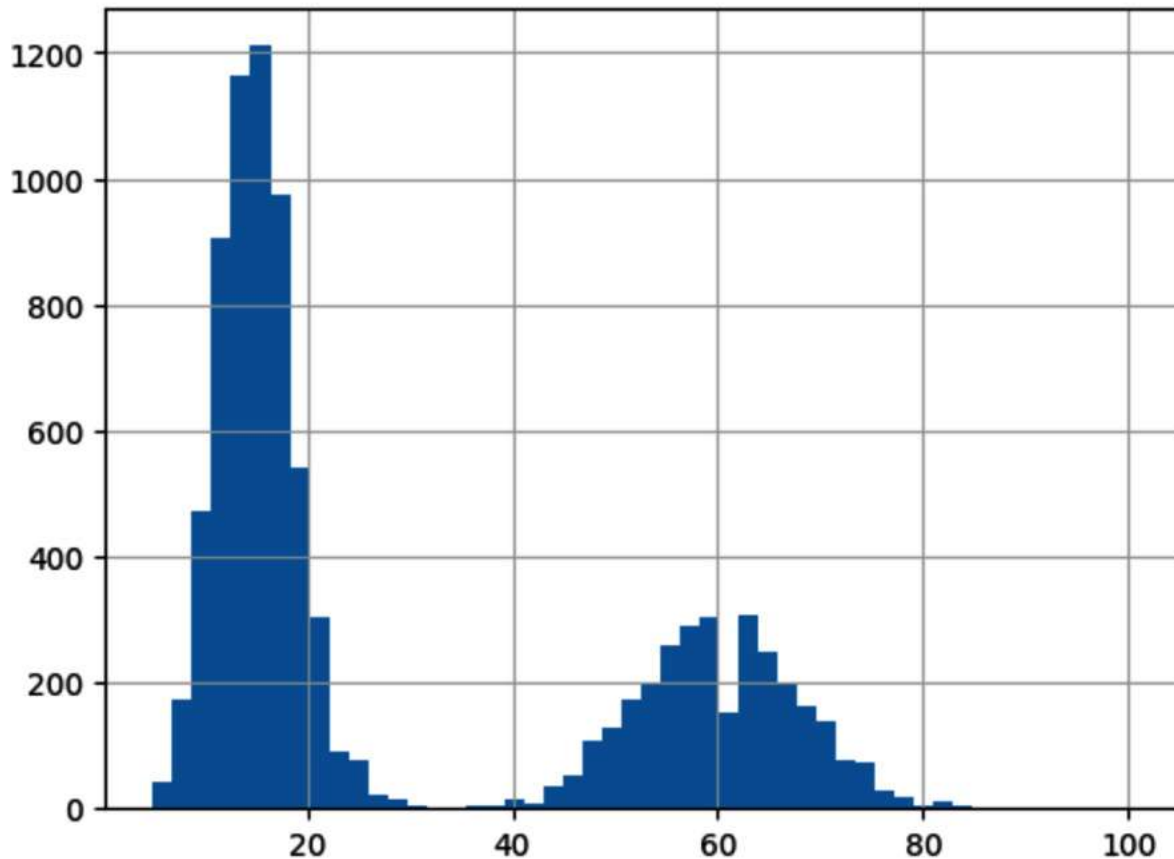


Figure 8.4 – The histogram of our break-takers with a larger number of bins, showing more granularity

As you can see, our data is definitely not following a normal distribution; it appears to be bimodal, which means that there are two peaks of break times, at around 25 and 70 minutes. As our data is not normal, many of the most popular statistics tests may not apply; however, if we follow the given procedure, we can create normal data! Think I'm crazy? Well, see for yourself.

First off, we will need to utilize what is known as a sampling distribution, which is a distribution of point estimates of several samples of the same size. Our procedure for creating a sampling distribution will be the following:

1. Take 500 different samples of the break times of the size of 100 each.

2. Take a histogram of these 500 different point estimates (revealing their distribution).

The number of elements in the sample (100) was arbitrary but large enough to be a representative sample of the population. The number of samples I took (500) was also arbitrary, but large enough to ensure that our data would converge to a normal distribution:

```
point_estimates = []
for x in range(500): # Generate 500 samples
    # take a sample of 100 points
    sample = np.random.choice(a=breaks, size=100)
    # add the sample mean to our list of point estimates
    point_estimates.append( sample.mean() )
    # look at the distribution of our sample means
pd.DataFrame(point_estimates).hist()
```

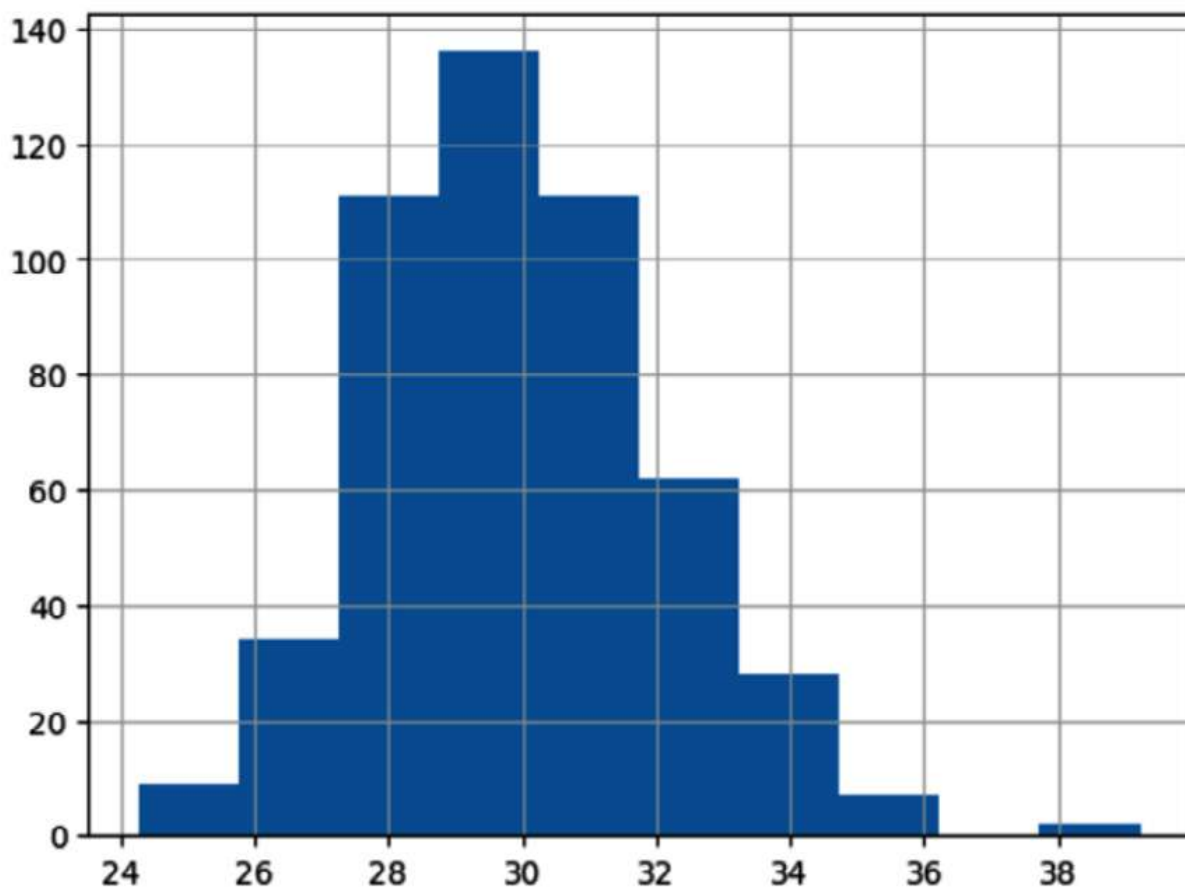


Figure 8.5 – The distribution of sample means becomes much more normally distributed, a blessing of the central limit theorem

Behold! The sampling distribution of the sample mean appears to be normal even though we took data from an underlying bimodal population distribution. It is important to note that the bars in this histogram represent the average break length of 500 samples of employees, where each sample has 100 people in it. In other words, a sampling distribution is a distribution of several point estimates.

Our data converged to a normal distribution because of something called the central limit theorem, which states that the sampling distribution (the distribution of point estimates) will approach a normal distribution as we increase the number of samples taken.

What's more, as we take more and more samples, the mean of the sampling distribution will approach the true population mean, as shown:

```
breaks.mean() - np.array(point_estimates).mean()  
# .042 minutes difference
```

This is actually a very exciting result because it means that we can get even closer than a single point estimate by taking multiple point estimates and utilizing the central limit theorem!

IMPORTANT NOTE

In general, as we increase the number of samples taken, our estimate will get closer to the parameter (actual value).

Confidence intervals

While point estimates are okay, estimates of a population parameter and sampling distributions are even better. There are the following two main issues with these approaches:

- Single point estimates are very prone to error (due to sampling bias among other things)
- Taking multiple samples of a certain size for sampling distributions might not be feasible, and may sometimes be even more infeasible than actually finding the population parameter

For these reasons and more, we may turn to a concept known as the **confidence interval** to find statistics.

A confidence interval is a range of values based on a point estimate that contains the true population parameter at some confidence level.

Confidence is an important concept in advanced statistics. Its meaning is sometimes misconstrued. Informally, a confidence level does not represent a *probability of being correct*; instead, it represents the frequency at which the obtained answer will be accurate. For example, if you want to have a 95% chance of capturing the true population parameter using only a single point estimate, you have to set your confidence level to 95%.

IMPORTANT NOTE

Higher confidence levels result in wider (larger) confidence intervals in order to be more sure.

Calculating a confidence interval involves finding a point estimate and then incorporating a margin of error to create a range. The margin of error is a value that represents our certainty that our point estimate is accurate and is based on our desired confidence level, the variance of the data, and how big our sample is. There are many ways to calculate confidence intervals; for the purpose of brevity and

simplicity, we will look at a single way of taking the confidence interval of a population mean. For this confidence interval, we need the following:

- A point estimate. For this, we will take our sample mean of break lengths from our previous example.
- An estimate of the population standard deviation, which represents the variance in the data. This is calculated by taking the sample standard deviation (the standard deviation of the sample data) and dividing that number by the square root of the population size.
- The degrees of freedom (which is the sample size - 1).

Obtaining these numbers might seem arbitrary but, trust me, there is a reason for all of them. However, again for simplicity, I will use prebuilt Python modules, as shown, to calculate our confidence interval and then demonstrate its value:

```
import math
sample_size = 100
# the size of the sample we wish to take
sample = np.random.choice(a= breaks, size = sample_size)
a sample of sample_size taken from the 9,000 breaks population from before
sample_mean = sample.mean()
# the sample mean of the break lengths sample
sample_stdev = sample.std()
# sample standard deviation
sigma = sample_stdev/math.sqrt(sample_size)
# population standard deviation estimate
stats.t.interval(confidence = 0.95, # Confidence level 95%
df= sample_size - 1, # Degrees of freedom
loc = sample_mean, # Sample mean
scale = sigma) # Standard deviation
# (24.28, 33.14)
```

To reiterate, this range of values (from **24.28** to **33.14**) represents a confidence interval for the average break time with 95% confidence.

We already know that our population parameter is **29.99**, and note that the interval includes the population mean of **29.99**.

As mentioned earlier that the confidence level is not a percentage of the accuracy of our interval but the percent chance that the interval will even contain the population parameter at all.

To better understand the confidence level, let's take 10,000 confidence intervals and see how often our population mean falls in the interval. First, let's make a function, as illustrated, that makes a single confidence interval from our breaks data:

```
# function to make confidence interval
def makeConfidenceInterval():
sample_size = 100
sample = np.random.choice(a= breaks, size = sample_size)
sample_mean = sample.mean() # sample mean
sample_stdev = sample.std()
# sample standard deviation
sigma = sample_stdev/math.sqrt(sample_size)
# population standard deviation estimate
return stats.t.interval(confidence = 0.95, df= sample_size - 1, loc = sample_mean,
scale = sigma)
```

Now that we have a function that will create a single confidence interval, let's create a procedure that will test the probability that a single confidence interval will contain the true population parameter, 29.99:

1. Take **10,000** confidence intervals of the sample mean.
2. Count the number of times that the population parameter falls into our confidence intervals.
3. Output the ratio of the number of times the parameter fell into the interval by 10,000:

```
breaks_mean = breaks.mean()
times_in_interval = 0
n = 10_000
for i in range(n):
    interval = makeConfidenceInterval()
    if breaks_mean >= interval[0] and breaks_mean <= interval[1]: # if 29.99 falls
in the interval
        times_in_interval += 1
print(times_in_interval / n)
# 0.9465
```

Success! We see that about 95% of our confidence intervals contained our actual population mean.

Estimating population parameters through point estimates and confidence intervals is a relatively simple and powerful form of statistical inference.

Let's also take a quick look at how the size of confidence intervals changes as we change our confidence level. Let's calculate confidence intervals for multiple confidence levels and look at how large the intervals are by looking at the difference between the two numbers. Our hypothesis will be that as we make our confidence level larger, we will likely see larger confidence intervals to be sure that we catch the true population parameter:

```
for confidence in (.5, .8, .85, .9, .95, .99):
    confidence_interval = stats.t.interval(confidence = confidence, df= sample_size - 1,
    loc = sample_mean, scale = sigma)
    length_of_interval = round(confidence_interval[1] - confidence_interval[0], 2)
    # the length of the confidence interval
    print( "confidence {0} has a interval of size {1}".format(confidence,
    length_of_interval))
confidence 0.5 has a interval of size 2.95
confidence 0.8 has a interval of size 5.63
confidence 0.85 has a interval of size 6.33
confidence 0.9 has a interval of size 7.24
confidence 0.95 has a interval of size 8.65
confidence 0.99 has a interval of size 11.45
```

We can see that as we wish to be *more confident* in our interval, our interval expands in order to compensate.

Next, we will take our concept of confidence levels and look at statistical hypothesis testing in order to both expand on these topics and create (usually) even more powerful statistical inferences.

Hypothesis tests

Hypothesis tests are one of the most widely used tests in statistics. They come in many forms; however, all of them have the same basic purpose.

A hypothesis test is a statistical test that is used to ascertain whether we are allowed to assume that a certain condition is true for the entire population, given a data sample. Basically, a hypothesis test is a test for a certain hypothesis that we have about an entire population. The result of the test then tells us whether we should believe the hypothesis or reject it for an alternative one.

You can think of the hypothesis test's framework to determine whether the observed sample data deviates from what was to be expected from the population itself. Now, this sounds like a difficult task but, luckily, Python comes to the rescue and includes built-in libraries to conduct these tests easily.

A hypothesis test generally looks at two opposing hypotheses about a population. We call them the **null hypothesis** and the **alternative hypothesis**. The null hypothesis is the statement being tested and is the default correct answer; it is our starting point and our original hypothesis. The alternative hypothesis is the statement that opposes the null hypothesis. Our test will tell us which hypothesis we should trust and which we should reject.

Based on sample data from a population, a hypothesis test determines whether or not to reject the null hypothesis. We usually use a p value (which is based on our significance level) to make this conclusion.

IMPORTANT NOTE

A very common misconception is that statistical hypothesis tests are designed to select the more likely of the two hypotheses. This is incorrect. A hypothesis test will default to the null hypothesis until there is enough data to support the alternative hypothesis.

The following are some examples of questions you can answer with a hypothesis test:

- Does the mean break time of employees differ from 40 minutes?
- Is there a difference between people who interacted with website A and people who interacted with website B (A/B testing)?
- Does a sample of coffee beans vary significantly in taste from the entire population of beans?

Conducting a hypothesis test

There are multiple types of hypothesis tests out there, and among them are dozens of different procedures and metrics. Nonetheless, there are five basic steps that most hypothesis tests follow, which are as follows:

1. Specify the hypotheses:
 - Here, we formulate our two hypotheses: the null and the alternative.
 - We usually use the notation of H_0 to represent the null hypothesis and H_a to represent our alternative hypothesis.
2. Determine the sample size for the test sample:

- This calculation depends on the chosen test. Usually, we have to determine a proper sample size in order to utilize theorems, such as the central limit theorem, and assume the normality of data.
3. Choose a significance level (usually called alpha or α):
 - A significance level of 0.05 is common.
 4. Collect the data:
 - Collect a sample of data to conduct the test.
 5. Decide whether to reject or fail to reject the null hypothesis:
 - This step changes slightly based on the type of test being used. The final result will either yield a rejection of the null hypothesis in favor of the alternative or fail to reject the null hypothesis.

In this chapter, we will look at the following three types of hypothesis tests:

- One-sample *t*-tests
- Chi-square goodness of fit
- Chi-square test for association/independence

There are many more tests. However, these three are a great combination of distinct, simple, and powerful tests. One of the biggest things to consider when choosing which test we should implement is the type of data we are working with—specifically, whether we are dealing with continuous or categorical data. In order to truly see the effects of a hypothesis, I suggest we dive right into an example. First, let's look at the use of *t*-tests to deal with continuous data.

One-sample *t*-tests

The one-sample *t*-test is a statistical test used to determine whether a quantitative (numerical) data sample differs significantly from another dataset (the population or another sample). Suppose, in our previous employee break time example, we look specifically at the engineering department's break times, as shown:

```
long_breaks_in_engineering = stats.poisson.rvs(loc=10, mu=55, size=100)
short_breaks_in_engineering = stats.poisson.rvs(loc=10, mu=15, size=300)
engineering_breaks = np.concatenate((long_breaks_in_engineering,
short_breaks_in_engineering))
print(breaks.mean())
# 29.99
print(engineering_breaks.mean())
# 34.825
```

Note that I took the same approach as making the original break times but with the following two differences:

- I took a smaller sample from the Poisson distribution (to simulate that we took a sample of 400 people from the engineering department)
- Instead of using μ of 60 as before, I used 55 to simulate the fact that the engineering department's break behavior isn't exactly the same as the company's behavior as a whole

It is easy to see that there seems to be a difference (of over 5 minutes) between the engineering department and the company as a whole. We usually don't have the entire population and the population parameters at our disposal, but I have them simulated in order for the example to work. So, even though we (the omniscient readers) can see a difference, we will assume that we know nothing of these population parameters and, instead, rely on a statistical test in order to ascertain these differences.

Example of a one-sample t-test

Our objective here is to ascertain whether there is a difference between the overall population's (company employees) break times and the break times of employees in the engineering department.

Let's now conduct a t -test at a 95% confidence level in order to find a difference (or not!). Technically speaking, this test will tell us whether the sample comes from the same distribution as the population.

Assumptions of the one-sample t-test

Before diving into the five steps, we must first acknowledge that t -tests must satisfy the following two conditions to work properly:

- The population distribution should be normal, or the sample should be large ($n \geq 30$)
- In order to make the assumption that the sample is independently, randomly sampled, it is sufficient to enforce that the population size should be at least 10 times larger than the sample size ($10n < N$)

Note that our test requires that either the underlying data be normal (which we know is not true for us), or that the sample size is at least 30 points large. For the t -test, this condition is sufficient to assume normality. This test also requires independence, which is satisfied by taking a sufficiently small sample. Sounds weird, right? The basic idea is that our sample must be large enough to assume normality (through conclusions similar to the central limit theorem) but small enough to be independent of the population.

Now, let's follow our five steps:

1. Specify the hypotheses.

We will let H_a = the engineering department take breaks the same as the company as a whole.

If we let this be the company average, we may write the following:

H_a : (engineering takes breaks the same length as everyone else)

NOTE

Note how this is our null, or default, hypothesis. It is what we would assume, given no data. What we would like to show is the alternative hypothesis.

Now that we actually have some options for our alternative, we could either say that the engineering mean (let's call it that) is lower than the company average, higher than the company average, or just flat-out different (higher or lower) from the company average:

- If we wish to answer the question, *Is the sample mean different from the company average?*, then this is called a **two-tailed test** and our alternative hypothesis would be as follows:

Ha: (engineering takes breaks of different lengths than the rest of the company)

- If we want to find out whether the sample mean is lower than the company average or the sample mean is higher than the company average, then we are dealing with a **one-tailed test** and our alternative hypothesis would be one of the following hypotheses:
 - $H_a: (\text{engineering takes longer breaks})$
 - $H_a: (\text{engineering takes shorter breaks})$

The difference between one and two tails is the difference of dividing a number later on by two or not. The process remains completely unchanged for both. For this example, let's choose the two-tailed test. So, we are testing for whether or not this sample of the engineering department's average break times is different from the company average.

Our test will end in one of two possible conclusions: we will either reject the null hypothesis, which means that the engineering department's break times are different from the company average, or we will fail to reject the null hypothesis, which means that there wasn't enough evidence in the sample to support rejecting the null.

2. Determine the sample size for the test sample.

As mentioned earlier, most tests (including this one) make the assumption that either the underlying data is normal or that our sample is in the right range:

- The sample is at least 30 points (it is 400)
- The sample is less than 10% of the population (which would be 900 people)

3. Choose a significance level (usually called alpha or α). We will choose a 95% significance level, which means that our alpha would actually be $1 - .95 = .05$.

4. Collect the data. This is generated through the two Poisson distributions.

5. Decide whether to reject or fail to reject the null hypothesis. As mentioned before, this step varies based on the test used. For a one-sample *t*-test, we must calculate two numbers: the test statistic and our *p* value. Luckily, we can do this in one line in Python.

A test statistic is a value that is derived from sample data during a type of hypothesis test. They are used to determine whether or not to reject the null hypothesis.

The test statistic is used to compare the observed data with what is expected under the null hypothesis. The test statistic is used in conjunction with the *p* value.

The p value is the probability that the observed data occurred this way by chance.

When the data shows very strong evidence against the null hypothesis, the test statistic becomes large (either positive or negative) and the p value usually becomes very small, which means that our test is showing powerful results and what is happening is probably not happening by chance.

In the case of a t -test, a t value is our test statistic, as shown:

```
t_statistic, p_value = stats.ttest_1samp(a= engineering_breaks, popmean=
breaks.mean())
```

We input the `engineering_breaks` variable (which holds 400 break times) and the population mean (`popmean`), and we obtain the following numbers:

```
t_statistic == -5.742
p_value == .00000018
```

The test result shows that the t value is -5.742 . This is a standardized metric that reveals the deviation of the sample mean from the null hypothesis. The p value is what gives us our final answer. Our p value tells us how often our result would appear by chance. So, for example, if our p value was $.06$, then that would mean we should expect to observe this data by chance about 6% of the time. This means that about 6% of samples would yield results like this.

We are interested in how our p value compares to our significance level:

- If the p value is less than the significance level, then we can reject the null hypothesis
- If the p value is greater than the significance level, then we failed to reject the null hypothesis

Our p value is way lower than $.05$ (our chosen significance level), which means that we may reject our null hypothesis in favor of the alternative. This means that the engineering department seems to take different break lengths from the company as a whole!

IMPORTANT NOTE

The use of p values is controversial. Many journals have actually banned the use of p values in tests for significance. This is because of the nature of the value. Suppose our p value came out to $.04$. This means that 4% of the time, our data just randomly happened to appear this way and is not significant in any way. 4% is not that small of a percentage! For this reason, many people are switching to different statistical tests. However, that does not mean that p values are useless. It merely means that we must be careful and aware of what the number is telling us.

There are many other types of t -tests, including one-tailed tests (mentioned before) and paired tests as well as two-sample t -tests (both not mentioned yet). These procedures can be readily found in statistics literature; however, we should look at something very important—what happens when we get it wrong.

Type I and Type II errors

In the realm of statistical hypothesis testing, two kinds of errors can occur: Type I and Type II errors. These errors are synonymous with the concepts of false positives and false negatives, respectively. Grasping these concepts is crucial as they underscore the potential limitations and risks associated with inferential statistics, where conclusions about a population are drawn from sample data.

Type I errors explained

A Type I error is often referred to as a **false positive**. Imagine you have an app on your phone designed to identify bird songs from snippets of music it hears. If the app indicates it recognizes a bird from the ambient noise when there is no bird nearby, it has made a Type I error—it has alerted you to a “hit” when there was none. In statistical hypothesis testing, this error occurs when we reject a true null hypothesis. The null hypothesis usually represents a default position or a statement of no effect—for example, the assumption that a new drug has no effect on a disease.

When setting up a hypothesis test, we determine the significance level (denoted as α), which defines the threshold for how much evidence is required to reject the null hypothesis. Commonly, a 5% significance level is used, meaning there is a 5% chance of rejecting the null hypothesis when it is actually true. This 5% is the risk we are willing to take of making a Type I error.

Type II errors explained

Conversely, a Type II error is when we overlook something significant—a false negative. Consider a medical test designed to detect a disease. If the test results come back negative when the person has the disease, the test has made a Type II error. In the context of hypothesis testing, this error occurs when the null hypothesis is not rejected despite being false. For example, we might conclude that the new drug has no effect on a disease when, in fact, it does.

The likelihood of a Type II error is represented by β , and it is inversely related to the significance level α . As we lower the risk of committing a Type I error by choosing a smaller α (for example, setting a higher confidence level such as 99%), we inadvertently increase the risk of a Type II error. This is because requiring stronger evidence to reject the null hypothesis (i.e., a higher confidence level) can make it harder to detect an actual effect.

Balancing these errors is a critical part of designing experiments and interpreting statistical results. Researchers must decide on an acceptable balance between the risks of Type I and Type II errors, often based on the context of the research and the potential consequences of incorrect conclusions.

Hypothesis testing for categorical variables

T-tests (among other tests) are hypothesis tests that work to compare and contrast quantitative variables and underlying population distributions. In this section, we will explore two new tests, both of which

serve to explore qualitative data. They are both a form of test called **chi-square tests**. These two tests will perform the following two tasks for us:

- Determine whether a sample of categorical variables is taken from a specific population (similar to the t -test)
- Determine whether two variables affect each other and are associated with each other

Chi-square goodness of fit test

The one-sample t -test was used to check whether a sample mean differed from the population mean. The chi-square goodness of fit test is very similar to the one-sample t -test in that it tests whether the distribution of the sample data matches an expected distribution, while the big difference is that it tests for categorical variables.

For example, a chi-square goodness of fit test would be used to see whether the race demographics of your company match that of the entire city of the U.S. population. It can also be used to see whether users of your website show similar characteristics to average internet users.

As we are working with categorical data, we have to be careful because categories such as “male,” “female,” or “other” don’t have any mathematical meaning. Therefore, we must consider counts of the variables rather than the actual variables themselves.

In general, we use the chi-square goodness of fit test in the following cases:

- We want to analyze one categorical variable from one population
- We want to determine whether a variable fits a specified or expected distribution

In a chi-square test, we compare what is observed to what we expect.

Assumptions of the chi-square goodness of fit test

There are two usual assumptions of this test, as follows:

- All the expected counts are at least 5
- Individual observations are independent and the population should be at least 10 times as large as the sample ($10n < N$)

The second assumption should look familiar to the t -test; however, the first assumption should look foreign. Expected counts are something we haven’t talked about yet but are about to!

When formulating our null and alternative hypotheses for this test, we consider a default distribution of categorical variables. For example, if we have a die and we are testing whether or not the outcomes are coming from a fair die, our hypothesis might look as follows:

H_a : The specified distribution of the categorical variable is correct.

$$p1 = \frac{1}{6}, p2 = \frac{1}{6}, p3 = \frac{1}{6}, p4 = \frac{1}{6}, p5 = \frac{1}{6}, p6 = \frac{1}{6}$$

Our alternative hypothesis is quite simple, as shown:

H_{0a} : The specified distribution of the categorical variable is not correct. At least one of the π values is not correct.

In the t -test, we used our test statistic (the t value) to find our p value. In a chi-square test, our test statistic is, well, a chi-square:

Test Statistic: $\chi^2 = \text{over } k \text{ categories}$

Degrees of Freedom = $k - 1$

A critical value is when we use χ^2 as well as our degrees of freedom and our significance level, and then reject the null hypothesis if the p value is below our significance level (the same as before).

Let's look at an example to understand this further.

Example of a chi-square test for goodness of fit

The CDC categorizes adult BMIs into four classes: **Under/Normal**, **Over**, **Obesity**, and **Extreme Obesity**. A 2009 survey showed the distribution for adults in the US to be 31.2%, 33.1%, 29.4%, and 6.3%, respectively. A total of 500 adults were randomly sampled and their BMI categories were recorded.

	Under/Normal	Over	Obesity	Extreme Obesity	Total
Observed	102	178	186	34	500

Figure 8.6 – The raw numbers of people who fall under each BMI category

Is there evidence to suggest that BMI trends have changed since 2009? Let's test at the 0.05 significance level:

1. First, let's calculate our expected values. In a sample of 500, we expect 156 to be **Under/Normal** (that's 31.2% of 500), and we fill in the remaining boxes in the same way:

	Under/Normal	Over	Obesity	Extreme Obesity	Total
Observed	102	178	186	34	500
Expected	156	165.5	147	31.5	500

Figure 8.7 – The same raw numbers as the previous figure with an added row representing the "Expected" values in each category given a 2009 survey

2. Now, let's check that the conditions for our test are satisfied:

- All of the expected counts are greater than five
- Each observation is independent and our population is very large (much more than 10 times of 500 people)

3. Next, we will carry out a goodness of fit test. We will set our null and alternative hypotheses:

- H_a : The 2009 BMI distribution is still accurate.
- H_a : The 2009 BMI distribution is no longer accurate (at least one of the proportions is different now). We can calculate our test statistic by hand:

$$\text{Test Statistic: } \chi^2 = \sum \frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}} \text{ for } df = 3$$

$$= \frac{(102 - 156)^2}{156} + \frac{(178 - 165.5)^2}{165.5} + \frac{(186 - 147)^2}{147} + \frac{(34 - 31.5)^2}{31.5} = 30.18$$

Figure 8.8 – Walking through the calculation for our test statistic. Keep reading for the Python code to do it for you!

4. Alternatively, we can use our handy-dandy Python skills, like so:

```
observed = [102, 178, 186, 34]
expected = [156, 165.5, 147, 31.5]

chi_squared, p_value = stats.chisquare(f_obs= observed, f_exp= expected)

chi_squared, p_value

# (30.1817679275599, 1.26374310311106e-06)
```

Our p value is lower than .05; therefore, we may reject the null hypothesis in favor of the fact that the BMI trends today are different from what they were in 2009.

Chi-square test for association/independence

Independence as a concept in probability is when knowing the value of one variable tells you nothing about the value of another. For example, we might expect that the country and the month you were born in are independent. However, knowing which type of phone you use might indicate your creativity levels. Those variables might not be independent.

The chi-square test for association/independence helps us ascertain whether two categorical variables are independent of one another. The test for independence is commonly used to determine whether variables such as education levels or tax brackets vary based on demographic factors, such as gender, race, and religion. Let's look back at an example posed in the preceding chapter, the A/B split test.

Recall that we ran a test and exposed half of our users to a certain landing page (**Website A**), exposed the other half to a different landing page (**Website B**), and then measured the sign-up rates for both sites. We obtained the following results:

	Did not sign up	Signed up

Website A	134	54
Website B	110	48

Figure 8.9 – Results of our A/B test

We calculated website conversions but what we really want to know is whether there is a difference between the two variables: *which website was the user exposed to? And did the user sign up?* For this, we will use our chi-square test.

Assumptions of the chi-square independence test

There are the following two assumptions of this test:

- All expected counts are at least 5
- Individual observations are independent and the population should be at least 10 times as large as the sample ($10n < N$)

Note that they are exactly the same as the last chi-square test.

Let's set up our hypotheses:

- H_0 : There is no association between two categorical variables in the population of interest
- H_0 : Two categorical variables are independent in the population of interest
- H_a : There is an association between two categorical variables in the population of interest
- H_a : Two categorical variables are not independent in the population of interest

You might notice that we are missing something important here. Where are the expected counts?

Earlier, we had a prior distribution to compare our observed results to but now we do not. For this reason, we will have to create some. We can use the following formula to calculate the expected values for each value. In each cell of the table, we can use the following:

Expected Count = to calculate our chi-square test statistic and our degrees of freedom

$$\text{Test Statistic: } \chi^2 = \sum \frac{(\text{Observed}_{r,c} - \text{Expected}_{r,c})^2}{\text{Expected}_{r,c}}$$

Degrees of Freedom: $(r - 1) \cdot (c - 1)$

Here, r is the number of rows and c is the number of columns. Of course, as before, when we calculate our p value, we will reject the null hypothesis if that p value is less than the significance level. Let's use some built-in Python methods, as shown, in order to quickly get our results:

```
observed = np.array([[134, 54], [110, 48]])
# built a 2x2 matrix as seen in the table above
chi_squared, p_value, degrees_of_freedom, matrix = stats.chi2_contingency(observed=
observed)
chi_squared, p_value
# (0.04762692369491045, 0.82724528704422262)
```

We can see that our p value is quite large; therefore, we fail to reject the null hypothesis and we cannot say for sure that seeing a particular website has any effect on whether or not a user signs up. There is no association between these variables.

Summary

In this chapter, we looked at different statistical tests, including chi-square and t -tests, as well as point estimates and confidence intervals, in order to ascertain population parameters based on sample data. We were able to find that even with small samples of data, we can make powerful assumptions about the underlying population as a whole.

Using the concepts reviewed in this chapter, data scientists will be able to make inferences about entire datasets based on certain samples of data. In addition, they will be able to use hypothesis tests to gain a better understanding of full datasets, given samples of data.

Statistics is a very wide and expansive subject that cannot truly be covered in a single chapter; however, our understanding of the subject will allow us to carry on and talk more about how we can use statistics and probability in order to communicate our ideas through data science in the next chapter.

In the next chapter, we will discuss different ways of communicating results from data analysis including various presentation styles as well as visualization techniques.

Communicating Data

This chapter deals with the different ways of communicating results from our analysis. Here, we will look at different presentation styles as well as visualization techniques. The point of this chapter is to take our results and be able to explain them in a coherent, intelligible way so that anyone, whether they are data savvy or not, can understand and use our results.

Much of what we will discuss will be how to create effective graphs through labels, keys, colors, and more. We will also look at more advanced visualization techniques, such as parallel coordinate plots.

In this chapter, we will look into the following topics:

- Identifying effective and ineffective visualizations
- Recognizing when charts are attempting to “trick” the audience
- Being able to identify causation versus correlation
- Constructing appealing visuals that offer valuable insight

Why does communication matter?

Being able to conduct experiments and manipulate data in a coding language is not enough to conduct practical and applied data science. This is because data science is, generally, only as good as how it is used in practice. For instance, a medical data scientist might be able to predict the chance of a tourist contracting malaria in developing countries with >98% accuracy; however, if these results are published in a poorly marketed journal and online mentions of the study are minimal, their groundbreaking results that could potentially prevent deaths would never truly see the light of day.

For this reason, communication of results is arguably as important as the results themselves. A famous example of poor management of the distribution of results is the case of Gregor Mendel. Mendel is widely recognized as one of the founders of modern genetics. However, his results (including data and charts) were not well adopted until after his death. Mendel even sent them to Charles Darwin, who largely ignored Mendel’s papers, which were written in unknown Moravian journals.

Generally, there are two ways of presenting results: verbal and visual. Of course, both verbal and visual forms of communication can be broken down into dozens of subcategories, including slide decks, charts, journal papers, and even university lectures. However, we can find common elements of data presentation that can make anyone in the field more aware and effective in their communication skills.

Let’s dive right into effective (and ineffective) forms of communication, starting with visuals.

Identifying effective visualizations

The main goal of data visualization is to have the reader quickly digest the data, including possible trends, relationships, and more. Ideally, a reader will not have to spend more than 5-6 seconds digesting a single visualization. For this reason, we must take visuals very seriously and ensure that we are making a visual as effective as possible. Let's look at five basic types of graphs: scatter plots, line graphs, bar charts, histograms, and box plots.

Scatter plots

A scatter plot is probably one of the simplest graphs to create. It is made by creating two *quantitative* axes and using data points to represent observations. The main goal of a scatter plot is to highlight relationships between two variables and, if possible, reveal a correlation.

For example, we can look at two variables: the average hours of TV watched in a day and a 0-100 scale of work performance (0 being very poor performance and 100 being excellent performance). The goal here is to find a relationship (if it exists) between watching TV and average work performance.

The following code simulates a survey of a few people, in which they revealed the amount of television they watched, on average, in a day against a company-standard work performance metric. This line of code is creating 14 sample survey results of people answering the question of how many hours of TV they watch in a day:

```
import pandas as pd
hours_tv_watched = [0, 0, 0, 1, 1.3, 1.4, 2, 2.1, 2.6, 3.2, 4.1, 4.4, 4.4, 5]
```

This next line of code is creating 14 new sample survey results of the same people being rated on their work performance on a scale from 0 to 100. For example, the first person watched 0 hours of TV a day and was rated 87/100 on their work, while the last person watched, on average, 5 hours of TV a day and was rated 72/100:

```
work_performance = [87, 89, 92, 90, 82, 80, 77, 80, 76, 85, 80, 75, 73, 72]
```

Here, we are creating a DataFrame in order to simplify our **exploratory data analysis (EDA)** and make it easier to make a scatter plot:

```
df = pd.DataFrame({'hours_tv_watched':hours_tv_watched,
                   'work_performance':work_performance})
```

Now, we are actually making our scatter plot:

```
df.plot(x='hours_tv_watched', y='work_performance', kind='scatter')
```

In the following plot, we can see that our axes represent the number of hours of TV watched in a day and the person's work performance metric:

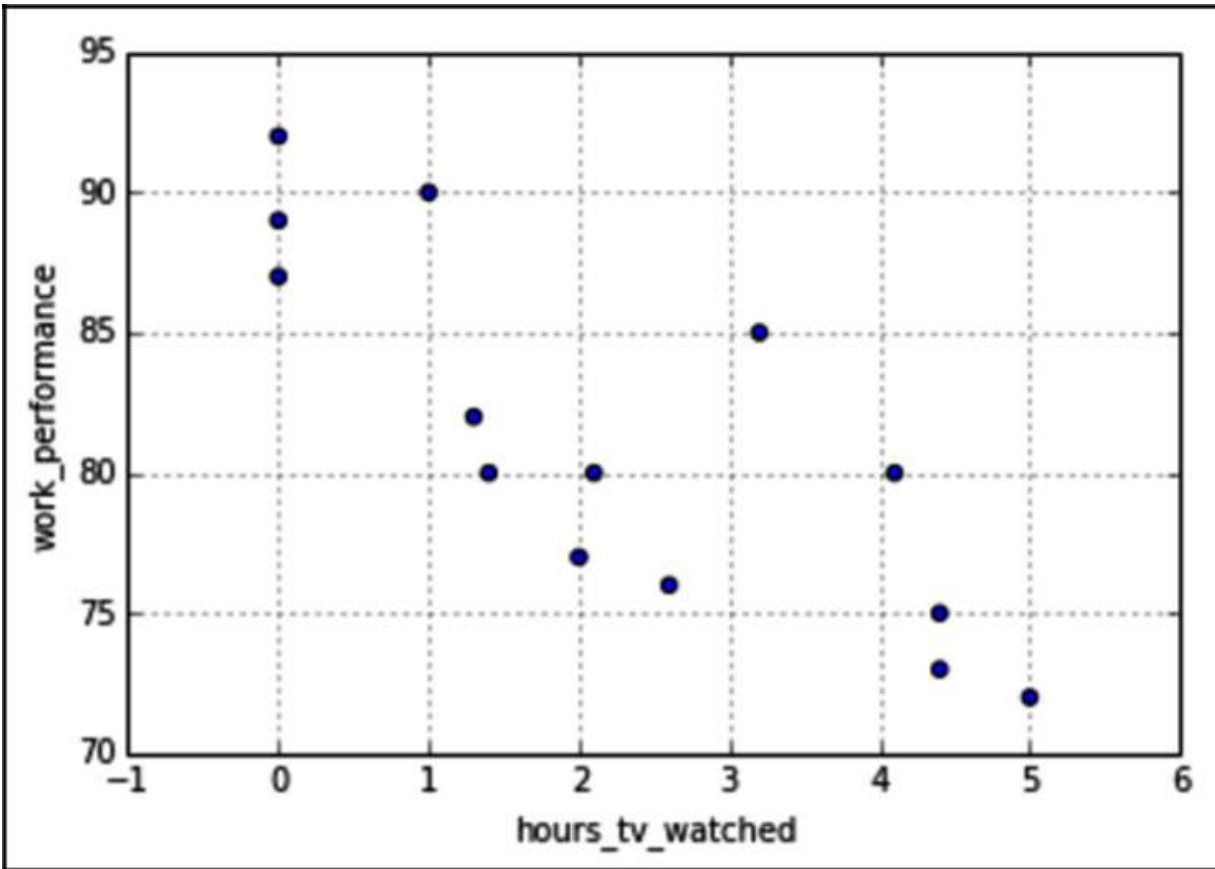


Figure 9.1 – Scatter plot: hours of TV watched versus work performance

Each point on a scatter plot represents a single observation (in this case, a person), and its location is a result of where the observation stands on each variable. This scatter plot does seem to show a relationship, which implies that as we watch more TV during the day, it seems to affect our work performance.

Of course, as we are now experts in statistics from the last two chapters, we know that this might not be causal. A scatter plot may only work to reveal a correlation or an association, but not a causation. Advanced statistical tests, such as the ones we saw in [Chapter 8, Advanced Statistics](#), might work to reveal causation. Later on in this chapter, we will see the damaging effects that trusting correlation might have.

Line graphs

Line graphs are, perhaps, one of the most widely used graphs in data communication. A line graph simply uses lines to connect data points and usually represents time on the x axis. Line graphs are a popular way to show changes in variables over time. A line graph, like a scatter plot, is used to plot **quantitative** variables.

As a great example, many of us wonder about possible links between what we see on TV and our behavior in the world. A friend of mine once took this thought to an extreme: he wondered if he could find a relationship between the TV show *The X-Files* and the amount of UFO sightings in the US. He found the number of sightings of UFOs per year and plotted them over time. He then added a quick graphic to ensure that readers would be able to identify the point in time when *The X-Files* was released:

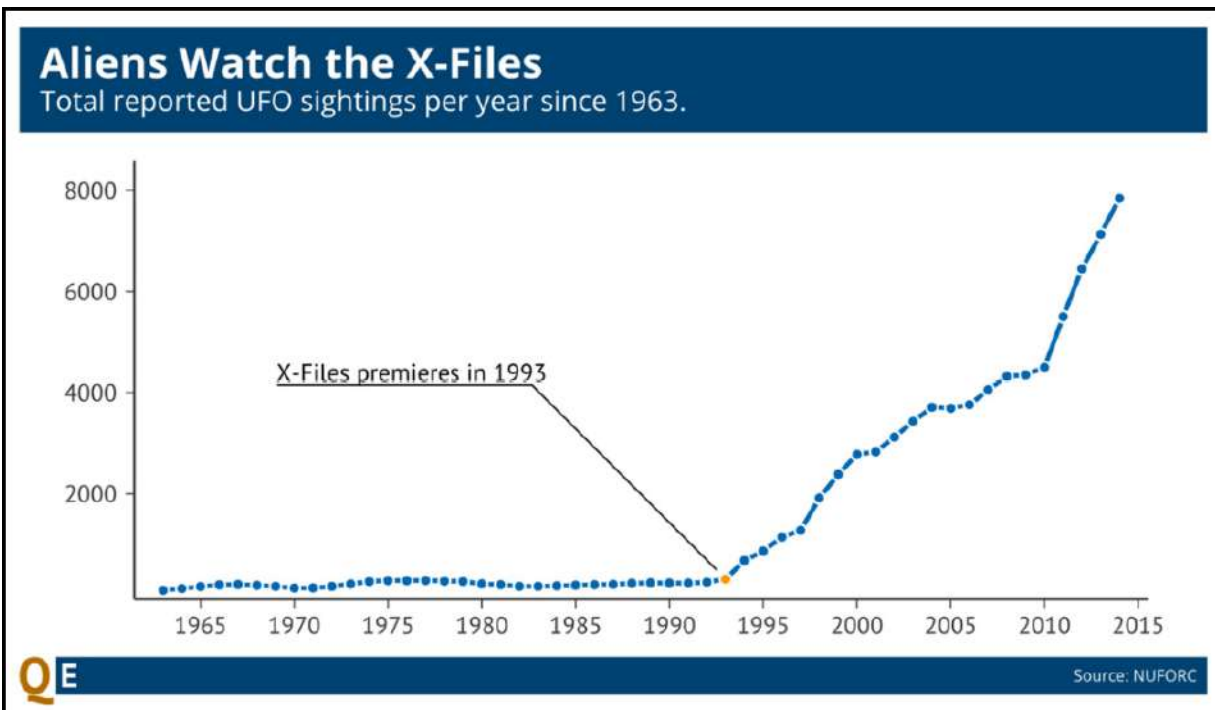


Figure 9.2 – Total reported UFO sightings since 1963

It appears to be clear that right after 1993, the year of *The X-Files*' premiere, the number of UFO sightings started to climb drastically.

This graphic, albeit light-hearted, is an excellent example of a simple line graph. We are told what each axis measures, we can quickly see a general trend in the data, and we can identify the author's intent, which is to show a relationship between the number of UFO sightings and *The X-Files*' premiere.

On the other hand, the following is a less impressive line graph:



Figure 9.3 – Line graph: gas price changes

This line graph attempts to highlight changes in the price of gas by plotting three points in time. At first glance, it is not much different than the previous graph; we have time on the bottom x axis and a quantitative value on the vertical y axis. The (not so) subtle difference here is that the three points are equally spaced out on the x axis; however, if we read their actual time indications, they are not equally spaced out in time. A year separates the first 2 points whereas a mere 7 days separate the last 2 points.

Bar charts

We generally turn to bar charts when trying to compare variables across different groups. For example, we can plot the number of countries per continent using a bar chart. Note how the x axis does not represent a quantitative variable; in fact, when using a bar chart, the x axis is generally a categorical variable, while the y axis is quantitative.

Note that, for this code, I am using the **World Health Organization's (WHO's)** report on alcohol consumption around the world by country:

```
from matplotlib import pyplot as plt
drinks =
pd.read_csv('https://raw.githubusercontent.com/sinanuozdemir/principles_of_
data_science/master/data/chapter_2/drinks.csv')
drinks.continent.value_counts().plot(kind='bar', title='Countries per Continent')
plt.xlabel('Continent')
plt.ylabel('Count')
```

The following graph shows us a count of the number of countries in each continent. We can see the continent code at the bottom of the bars, and the bar height represents the number of countries we have

in each continent. For example, we see that Africa has the most countries represented in our survey, while South America has the fewest:

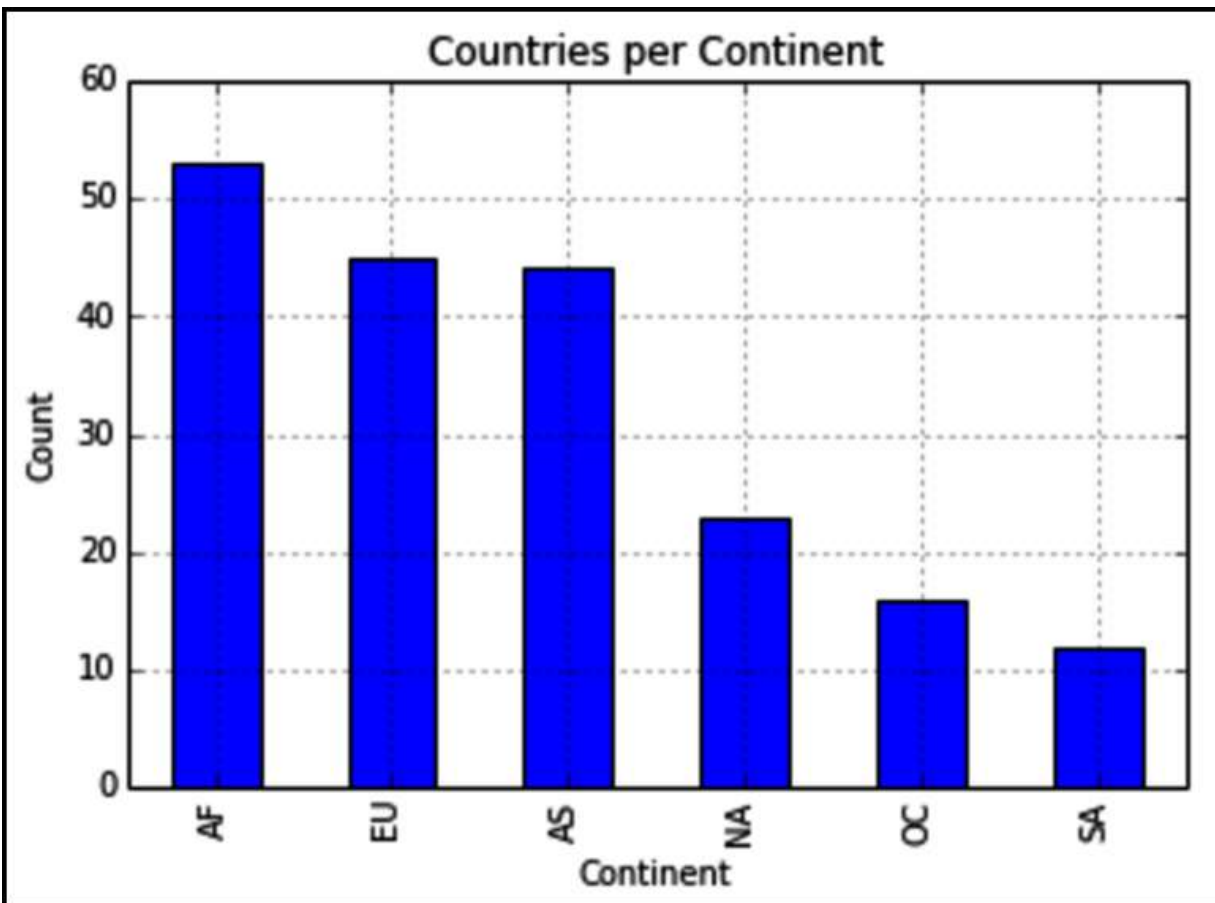


Figure 9.4 – Bar chart: count of countries in each continent

In addition to the count of countries, we can also plot the average beer servings per continent using a bar chart, as shown:

```
drinks.groupby('continent').beer_servings.mean().plot(kind='bar')
```

The preceding code gives us this chart:

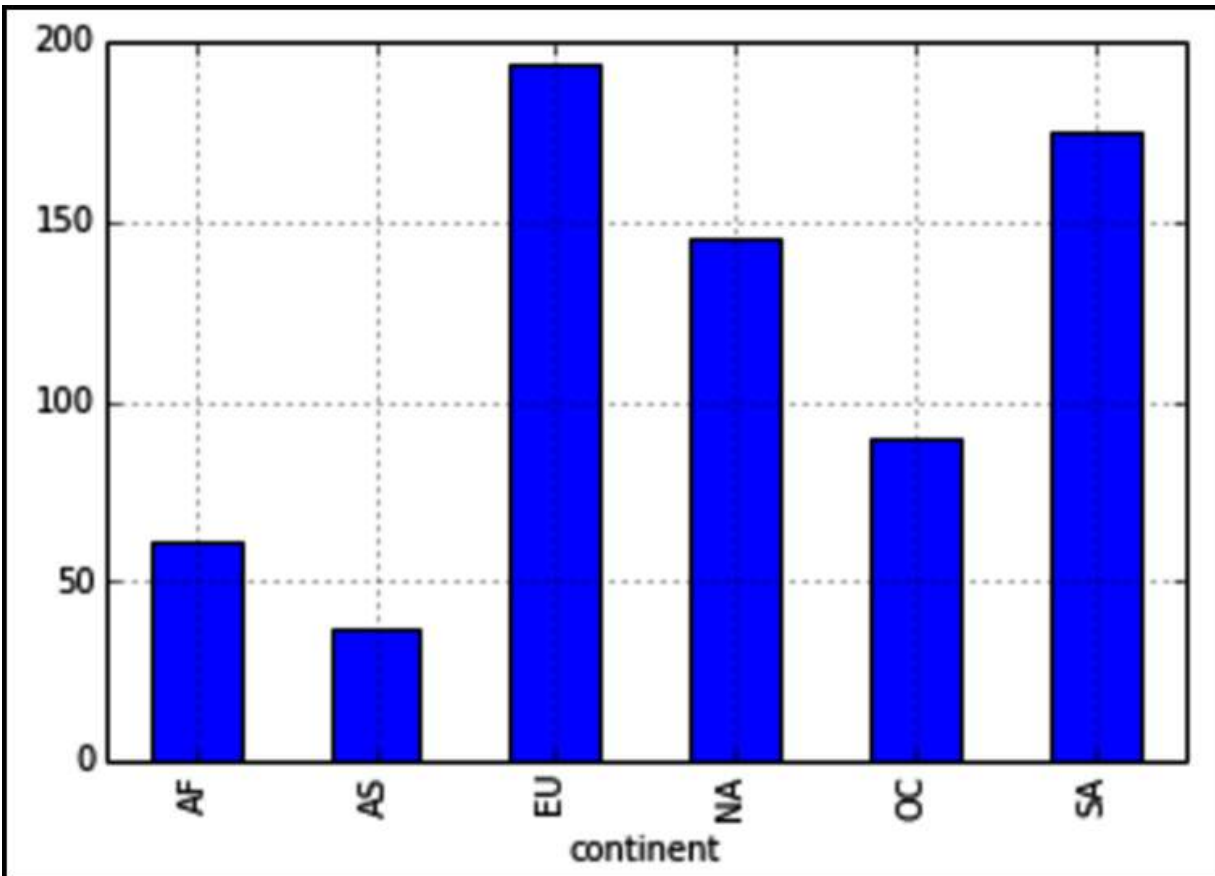


Figure 9.5 – Bar chart: average beer served per country

Note how a scatter plot or a line graph would not be able to support this data because they can only handle quantitative variables; bar graphs have the ability to demonstrate categorical values.

We can also use bar charts to graph variables that change over time, like a line graph.

Histograms

Histograms show the frequency distribution of a single quantitative variable by splitting the data, by range, into equidistant *bins* and plotting the raw count of observations in each bin. A histogram is effectively a bar chart where the *x* axis is a bin (subrange) of values and the *y* axis is a count. As an example, I will import a store's daily number of unique customers, as shown:

```
rossmann_sales = pd.read_csv('data/rossmann.csv')
rossmann_sales.head()
```

We get the following table:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1

Figure 9.6 – Store's daily number of unique customers

Note how we have multiple store data (in the first `store` column). Let's subset this data for only the first store, as shown:

```
first_rossmann_sales = rossmann_sales[rossmann_sales['Store']==1]
```

Now, let's plot a histogram of the first store's customer count:

```
first_rossmann_sales['Customers'].hist(bins=20)
plt.xlabel('Customer Bins')
plt.ylabel('Count')
```

This is what we get:

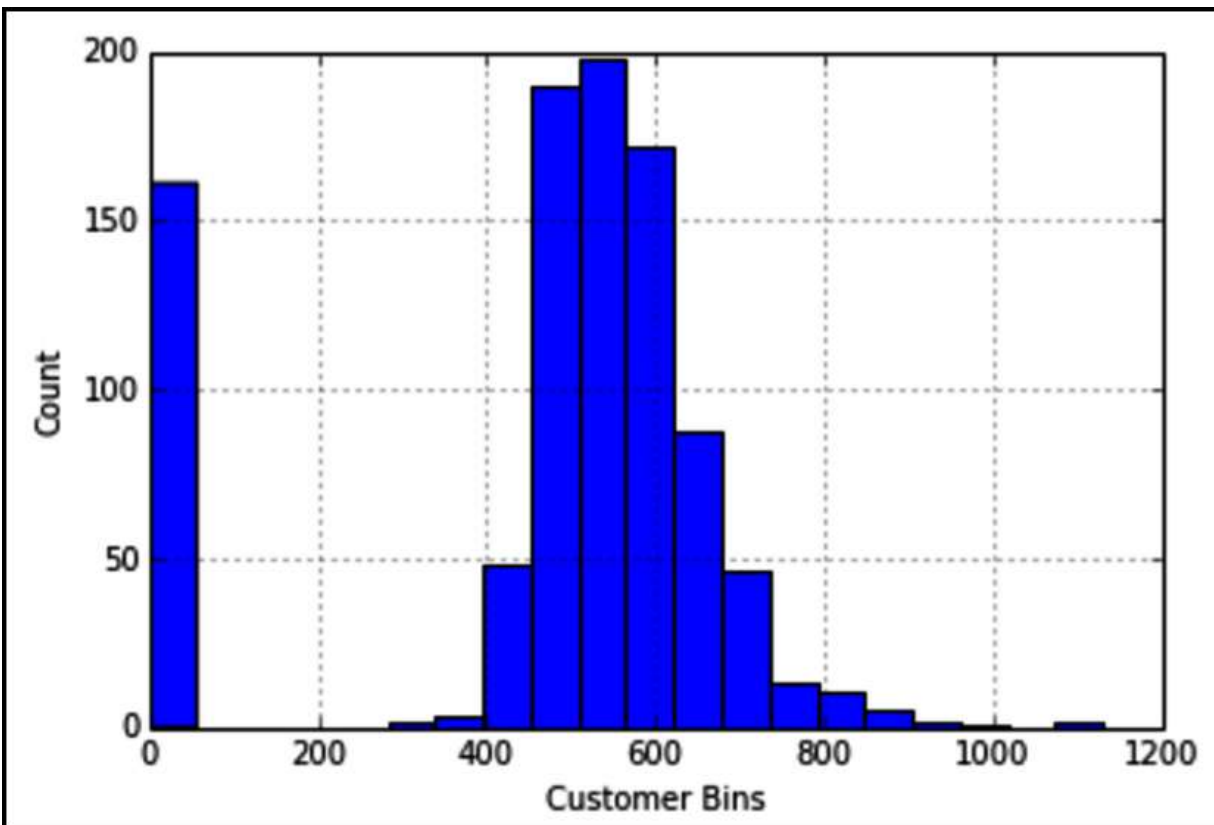


Figure 9.7 – Histogram: customer counts

The x axis is now categorical in that each category is a selected range of values; for example, 600-620 customers would potentially be a category. The y axis, like a bar chart, plots the number of observations in each category. In this graph, for example, one might take away the fact that most of the time, the number of customers on any given day will fall between 500 and 700.

Altogether, histograms are used to visualize the distribution of values that a quantitative variable can take on.

IMPORTANT NOTE

In a histogram, we do not put spaces between bars.

Box plots

Box plots are also used to show a distribution of values. They are created by plotting a five-number summary, as follows:

- The minimum value
- The first quartile (the number that separates the 25% lowest values from the rest)
- The median
- The third quartile (the number that separates the 25% highest values from the rest)
- The maximum value

In **pandas**, when we create box plots, the red line denotes the median, the top of the box (or the right if it is horizontal) is the third quartile, and the bottom (left) part of the box is the first quartile.

The following is a series of box plots showing the distribution of beer consumption according to continent:

```
drinks.boxplot(column='beer_servings', by='continent')
```

We get this graph:

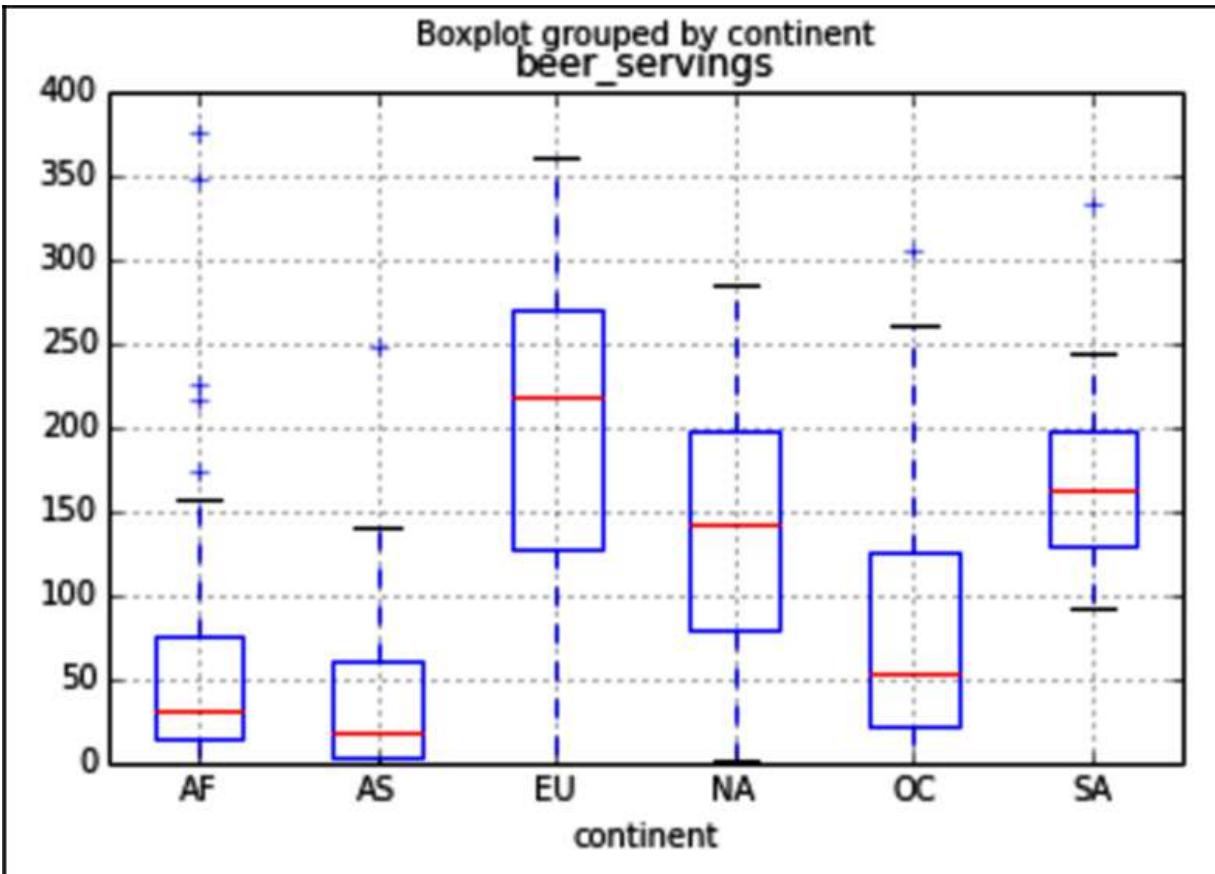


Figure 9.8 – Box plot: beer consumption by continent

Now, we can clearly see the distribution of beer consumption across the seven continents and how they differ. Africa and Asia have a much lower median of beer consumption than Europe or North America.

Box plots also have the added bonus of being able to show outliers much better than a histogram. This is because the minimum and maximum are parts of the box plot.

Getting back to the customer data, let's look at the same store customer numbers, but using a box plot:

```
first_rossmann_sales.boxplot(column='Customers', vert=False)
```

This is the graph we get:

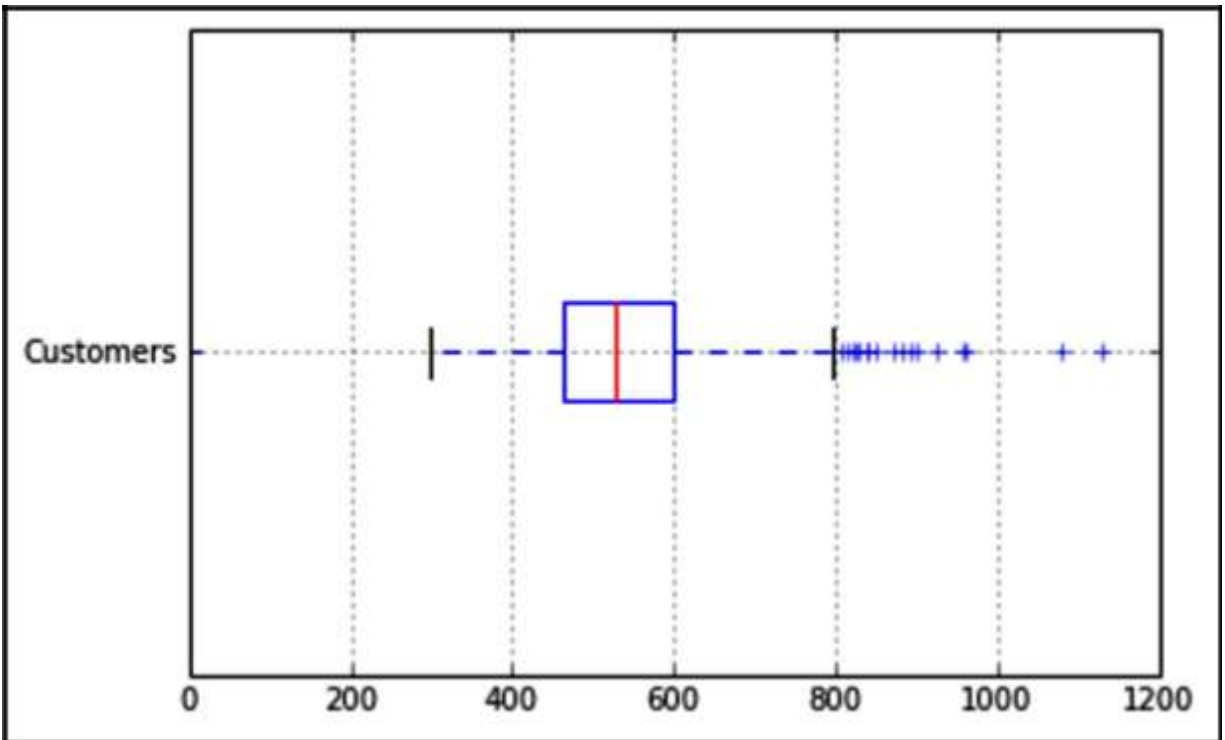


Figure 9.9 – Box plot: customer sales

This is the exact same data as plotted earlier in the histogram; however, now it is shown as a box plot. For the purpose of comparison, I will show you both graphs, one after the other:

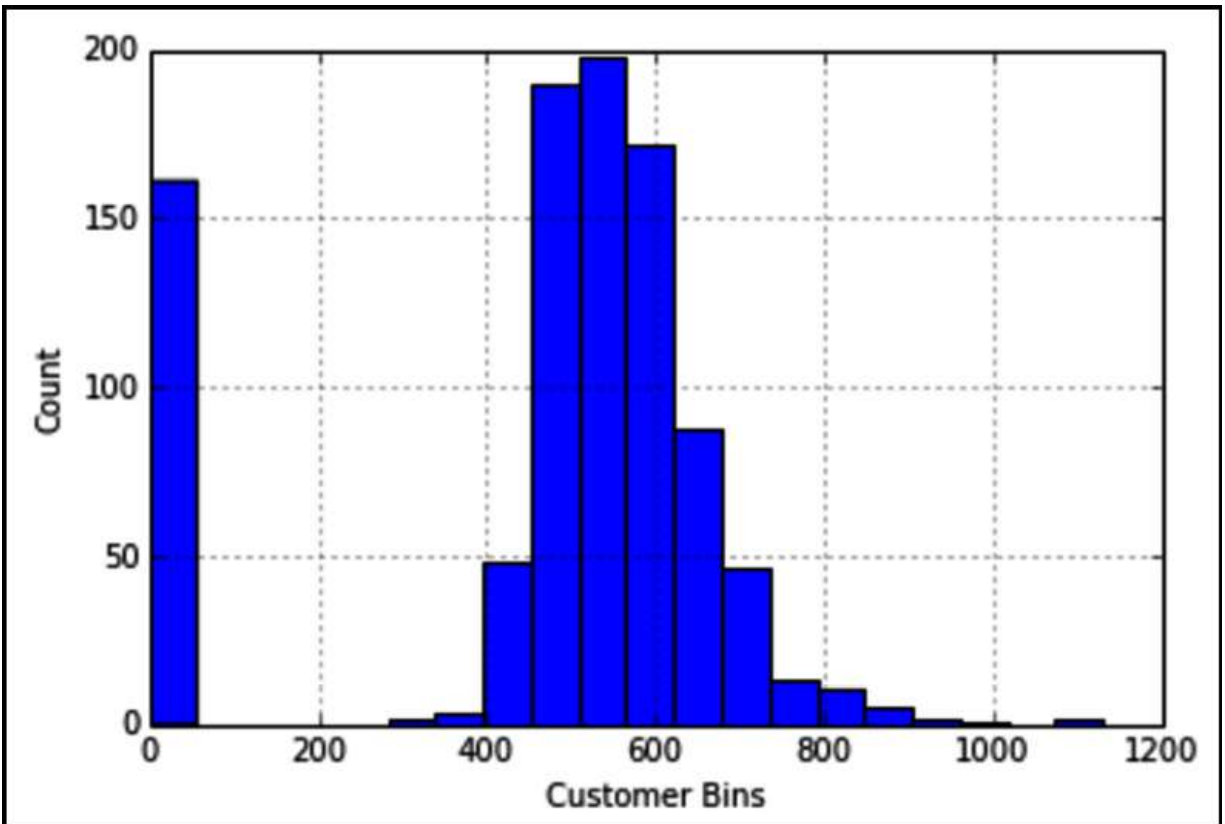


Figure 9.10 – Histogram: customer counts

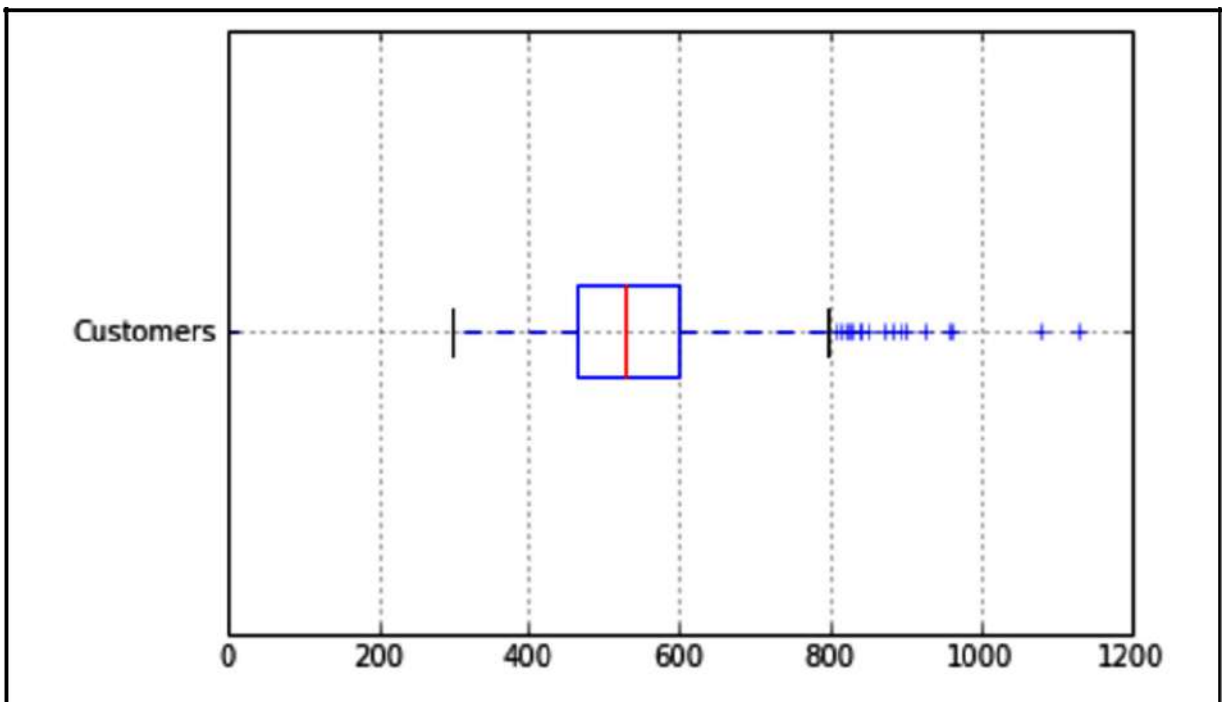


Figure 9.11 – Box plot: customer sales

Note how the x axis for each graph is the same, ranging from 0 to 1200. The box plot is much quicker at giving us a center for the data, the red line is the median, while the histogram works much better in showing us how spread out the data is and where people's biggest bins are. For example, the histogram reveals that there is a very large bin of zero people. This means that for a little over 150 days of data, there were zero customers.

Note that we can get the exact numbers to construct a box plot using the `describe` feature in `pandas`, as shown:

```
first_rossmann_sales['Customers'].describe()
```

min	0.000000
25%	463.000000
50%	529.000000
75%	598.750000
max	1130.000000

When graphs and statistics lie

I should be clear: statistics don't lie; people lie. One of the easiest ways to trick your audience is to confuse correlation with causation.

Correlation versus causation

I don't think I would be allowed to publish this book without taking a deeper dive into the differences between correlation and causation. For this example, I will continue to use my data for TV consumption and work performance.

Correlation is a quantitative metric between -1 and 1 that measures how two variables *move with each other*. If two variables have a correlation close to -1, it means that as one variable increases, the other decreases, and if two variables have a correlation close to +1, it means that those variables move together in the same direction; as one increases, so does the other, and the same when decreasing.

Causation is the idea that one variable affects another. For example, we can look at two variables: the average hours of TV watched in a day and a 0-100 scale of work performance (0 being very poor performance and 100 being excellent performance). One might expect that these two factors are

negatively correlated, which means that as the number of hours of TV watched increases in a 24-hour day, your overall work performance goes down. Recall the code from earlier, which is as follows. Here, I am looking at the same sample of 14 people as before and their answers to the question “*How many hours of TV do you watch on average per night?*”:

```
import pandas as pd
hours_tv_watched = [0, 0, 0, 1, 1.3, 1.4, 2, 2.1, 2.6, 3.2, 4.1, 4.4, 4.4, 5]
```

These are the same 14 people as mentioned earlier, in the same order, but now, instead of the number of hours of TV they watched, we have their work performance as graded by the company or a third-party system:

```
work_performance = [87, 89, 92, 90, 82, 80, 77, 80, 76, 85, 80, 75, 73, 72]
```

Then, we produce a DataFrame:

```
df = pd.DataFrame({'hours_tv_watched':hours_tv_watched,
                   'work_performance':work_performance})
```

Earlier, we looked at a scatter plot of these two variables, and it seemed to clearly show a downward trend between the variables: as TV consumption went up, work performance seemed to go down.

However, a correlation coefficient, a number between -1 and 1, is a great way to identify relationships between variables and, at the same time, quantify them and categorize their strength.

Now, we can introduce a new line of code that shows us the correlation between these two variables:

```
df.corr() # -0.824
```

Recall that a correlation, if close to -1, implies a strong negative correlation, while a correlation close to +1 implies a strong positive correlation.

This number helps support the hypothesis because a correlation coefficient close to -1 implies not only a negative correlation but a strong one at that. Again, we can see this via a scatter plot between the two variables. So, both our visuals and our numbers are aligned with each other. This is an important concept that should be true when communicating results. If your visuals and your numbers are off, people are less likely to take your analysis seriously:

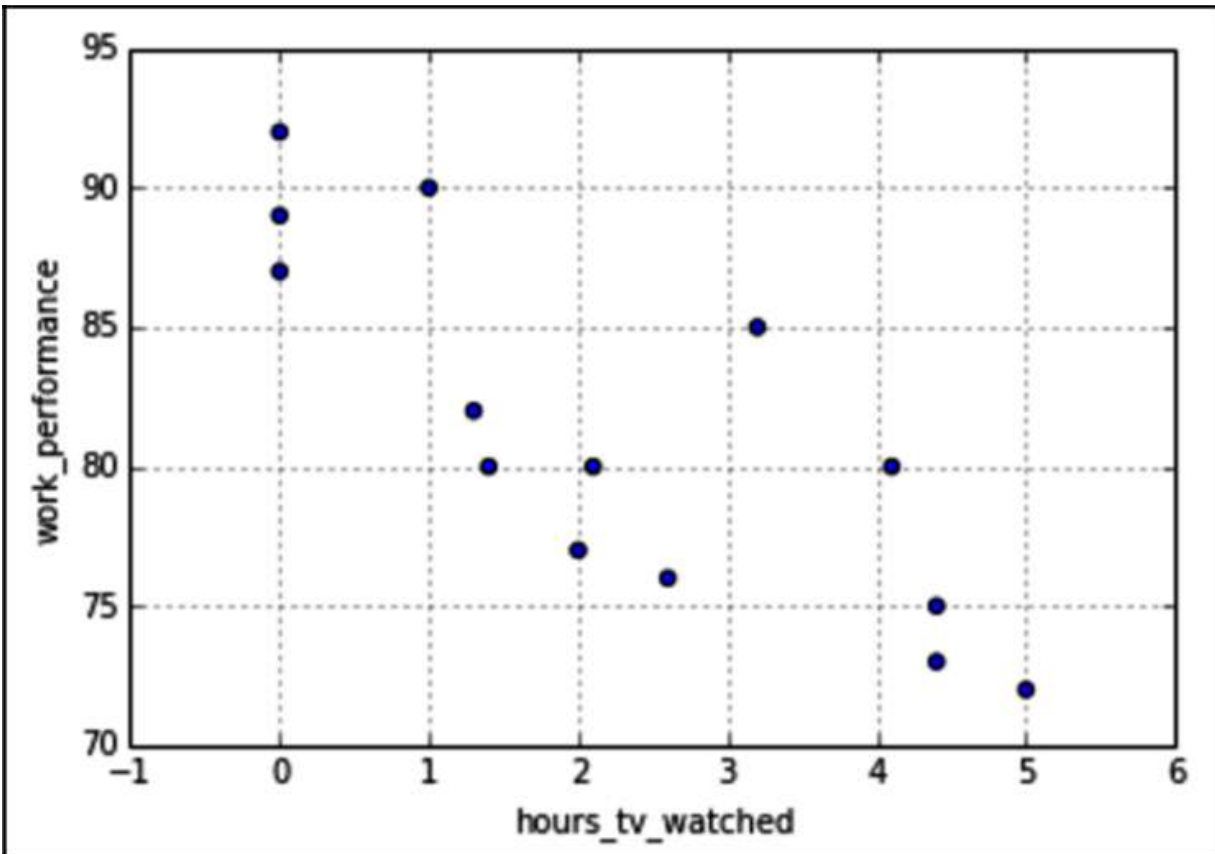


Figure 9.12 – Correlation: hours of TV watched and work performance

I cannot stress enough that correlation and causation are *different* from each other. Correlation simply quantifies the degree to which variables change together, whereas causation is where one variable actually determines the value of another. If you wish to share the results of the findings of your correlational work, you might be met with challengers in the audience asking for more work to be done. What is more terrifying is that no one might know that the analysis is incomplete, and you may make actionable decisions based on simple correlational work.

It is very often the case that two variables might be correlated with each other but they do not have any causation between them. This can be for a variety of reasons, some of which are as follows:

- There might be a *confounding factor* between them. This means that there is a third variable lurking in the background that is not being factored in that acts as a bridge between the two variables. For example, we previously showed that you might find that the amount of TV you watch is negatively correlated with work performance; that is, as the number of hours of TV you watch increases, your overall work performance may decrease. That is a correlation. It doesn't seem quite right to suggest that watching TV is the actual cause of a decrease in the quality of work performed. It might seem more plausible to suggest that there is a third factor, perhaps hours of sleep every night, that might answer this question. Perhaps, watching more TV decreases the amount of time you have for sleep, which in turn limits your work performance. The number of hours of sleep per night is the confounding factor.
- They might not have anything to do with each other! It might simply be a coincidence. There are many variables that are correlated but simply do not cause each other. Consider the following example:

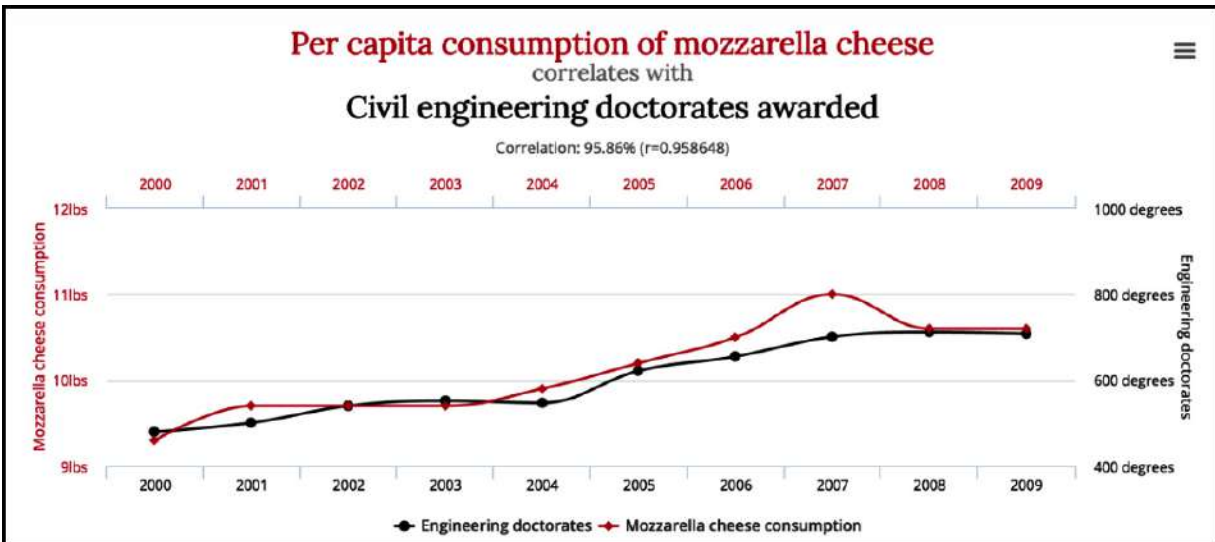


Figure 9.13 – Correlation analysis: cheese consumption and civil engineering doctorates

It is much more likely that these two variables only happen to correlate (more strongly than our previous example, I may add) than cheese consumption determining the number of civil engineering doctorates in the world.

You have likely heard the statement *correlation does not imply causation*, and the last graph is exactly the reason why data scientists must believe that. Just because there exists a mathematical correlation between variables does not mean they have causation between them. There might be confounding factors between them or they might just not have anything to do with each other!

Let's see what happens when we ignore confounding variables and correlations become extremely misleading.

Simpson's paradox

Simpson's paradox is a formal reason for why we need to take confounding variables seriously. The paradox states that a correlation between two variables can be completely reversed when we take different factors into account. This means that even if a graph shows a positive correlation, these variables can become *anti-correlated* when another factor (most likely a confounding one) is taken into consideration. This can be very troublesome to statisticians.

Suppose we wish to explore the relationship between two different landing pages (recall our previous A/B testing in [Chapter 7](#), *What Are the Chances? An Introduction to Statistics*). We will call these pages *Page A* and *Page B* once again. We have two splash pages that we wish to compare and contrast, and our main metric for choosing will be our conversion rates, just as earlier.

Suppose we run a preliminary test and find the following conversion result:

--	--

Page A	Page B
75% (263/350)	83% (248/300)

Table 9.1 – Running a preliminary test

This means that *Page B* has almost a 10% higher conversion rate than *Page A*. So, right off the bat, it seems like *Page B* is the better choice because it has a higher rate of conversion. If we were going to communicate this data to our colleagues, it would seem that we are in the clear!

However, let's see what happens when we also take into account which coast in the US the user was closer to, as shown:

	Page A	Page B
West Coast	95% (76/80)	93% (231/250)
East Coast	72% (193/270)	34% (17/50)
Both	75% (263/350)	83% (248/300)

Table 9.2 – Usage of our websites broken down by which coast the user is on

Thus, the paradox! When we break the sample down by location, it seems that *Page A* was better in *both* categories but was worse overall. That's the beauty and, also, the horrifying nature of the paradox. This happens because of the unbalanced classes between the four groups.

The *Page A/East Coast* group and the *Page B/West Coast* group are providing most of the people in the sample, therefore skewing the results to not be as expected. The confounding variable here might be the fact that the pages were made available at different hours of the day and the West Coast people were more likely to see *Page B*, while the East Coast people were more likely to see *Page A*.

There is a resolution to Simpson's paradox (and therefore an answer); however, the proof lies in a complex system of Bayesian networks and is a bit out of the scope of this book.

The main takeaway from Simpson's paradox is that we should not unduly give causal power to correlated variables. There might be confounding variables that have to be examined. Therefore, if you are able to reveal a correlation between two variables (such as website category and conversion rate or TV consumption and work performance), then you should absolutely try to isolate as many variables as possible that might be the reason for the correlation, or can at least help explain your case further.

If correlation doesn't imply causation, then what does?

As a data scientist, it is often quite frustrating to work with correlation and not be able to draw conclusive causality. The best way to confidently obtain causality is, usually, through randomized experiments, such as the ones we saw in [Chapter 8, Advanced Statistics](#). One would have to split up the population groups into randomly sampled groups and run hypothesis tests to conclude, with a degree of certainty, that there is a true causation between variables.

Verbal communication

Apart from visual demonstrations of data, verbal communication is just as important when presenting results. If you are not merely uploading results or publishing, you are usually presenting data to a room of data scientists and executives or to a conference hall.

In any case, there are key areas to focus on when giving a verbal presentation, especially when the presentation regards findings in data.

There are generally two styles of oral presentation: one meant for more professional settings, including corporate offices where the problem at hand is usually tied directly to company performance or some other **key performance indicator (KPI)**, and another meant more for a room of your peers where the key idea is to motivate the audience to care about your work.

It's about telling a story

Whether it is a formal or casual presentation, people like to hear stories. When you are presenting results, you are not just spitting out facts and metrics; you are attempting to frame the minds of your audience to believe in and care about what you have to say.

When giving a presentation, always be aware of your audience and try to gauge their reactions/interest in what you are saying. If they seem unengaged, try to relate the problem to them. For example, you can begin with a punchy statement that is designed to grab attention and hint at the kind of content in the remainder of your speech:

Just think, when popular TV shows like Game of Thrones come back, your employees will all spend more time watching TV and therefore will have lower work performance.

Now, you have their attention. It's about relating to your audience; whether it's your boss or your mom's friend, you have to find a way to make it relevant.

On the more formal side of things

When presenting data findings to a more formal audience, I like to stick to the following six steps:

1. **Outline the state of the problem:** In this step, we go over the current state of the problem, including what the problem is and how the problem came to the attention of the team of data scientists.
2. **Define the nature of the data:** Here, we go into more depth about who this problem affects, how the solution would change the situation, and previous work done on the problem, if any.
3. **Divulge an initial hypothesis:** Here, we state what we believe to be the solution before doing any work. This might seem like a more novice approach to presentations; however, this can be a good time to outline not just your initial hypothesis but, perhaps, the hypothesis of the entire company. For example, “We took a poll and 61% of the company believes there is no correlation between hours of TV watched and work performance.”
4. **Describe the solution and, possibly, the tools that led to the solution:** Get into how you solved the problem, any statistical tests used, and any assumptions that were made during the course of the problem.
5. **Share the impact that your solution will have on the problem:** Talk about whether your solution was different from the initial hypothesis. What will this mean for the future? How can we take action on this solution to improve ourselves and our company?
6. **Future steps:** Share what future steps can be taken to address the problem, such as how to implement a solution and what further work this research has sparked.

By following these steps, we can hit on all of the major areas of the data science method. The first thing you want to hit on during a formal presentation is action. You want your words and solutions to be actionable. There must be a clear path to take upon the completion of the project, and future steps should be defined.

The why/how/what strategy for presenting

When speaking on a less formal level, the *why/how/what* strategy is a quick and easy way to create a presentation worthy of praise. It is quite simple, as follows.

This model is borrowed from famous advertisements – the kind where they would not even tell you what the product was until there were 3 seconds left. They want to catch your attention and then, finally, reveal what it was that was so exciting. Consider the following example:

Hello everyone. I am here to tell you about why we seem to have a hard time focusing on our jobs when the Olympics are being aired. After mining survey results and merging this data with company-standard work performance data, I was able to find a correlation between the number of hours of TV watched per day and average work performance. Knowing this, we can all be a bit more aware of our TV-watching habits and make sure we don't let it affect our work. Thank you.

This chapter was actually formatted in this way! We started with *why* we should care about data communication, then we talked about *how* to accomplish it (through correlation, visuals, and so on), and finally, I am telling you the *what*, which is the why/how/what strategy (insert mind-blowing sound effect here).

Summary

Data communication is not an easy task. It is one thing to understand the mathematics of how data science works, but it is a completely different thing to try to convince a room of data scientists and non-data scientists alike of your results and their value to them. In this chapter, we went over basic chart making, how to identify faulty causation, and how to hone our oral presentation skills.

Our next few chapters will really begin to hit at one of the biggest talking points of data science. In the last nine chapters, we spoke about everything related to how to obtain data, clean data, and visualize data in order to gain a better understanding of the environment that the data represents.

We then turned to look at basic and advanced probability/statistics laws in order to use quantifiable theorems and tests on our data to get actionable results and answers.

In subsequent chapters, we will take a look into **machine learning (ML)** and the situations in which ML performs well and doesn't perform well. As we take a journey into this material, I urge you, the reader, to keep an open mind and truly understand not just how ML works, but also why we need to use it.

How to Tell if Your Toaster is Learning – Machine Learning Essentials

It seems as though every time we hear about the next great start-up or turn on the news, we hear something about a revolutionary piece of **machine learning (ML)** or **artificial intelligence (AI)** technology and how it will change the way we live. This chapter focuses on ML as a practical part of data science. We will cover the following topics in this chapter:

- Defining different types of ML, along with examples of each kind
- Regression and classification
- What is ML, and how is it used in data science?
- The differences between ML and statistical modeling and how ML is a broad category of the latter
- An Introduction to Linear Regression

Our aim in this chapter will be to utilize statistics, probability, and algorithmic thinking in order to understand and apply essential ML skills to practical industries, such as marketing. Examples will include predicting star ratings of restaurant reviews, predicting the presence of a disease, spam email detection, and much more. This chapter focuses on ML as a whole and as a single statistical model. The subsequent chapters will deal with many more models, some of which are much more complex.

We will also turn our focus on metrics, which tell us how effective our models are. We will use metrics in order to conclude results and make predictions using ML.

Introducing ML

In [Chapter 1](#), *Data Science Terminology*, we defined ML as giving computers the ability to learn from data without being given explicit rules by a programmer. This definition still holds true. ML is concerned with the ability to ascertain certain patterns (signals) out of data, even if the data has inherent errors in it (noise).

ML models are able to learn from data without the explicit direction of a human. That is the main difference between ML models and classical non-ML algorithms.

Classical algorithms are told directly by a human how to find the best answer in a complex system, and the algorithm then achieves these best solutions, often working faster and more efficiently than a human. However, the bottleneck here is that the human has to first come up with the best solution in

order to tell the algorithm what to do. In ML, the model is not told the best solution and, instead, is given several examples of the problem and told to figure out the best solution for itself.

ML is just another tool in the belt of a data scientist. It is on the same level as statistical tests (chi-square or t-tests) or uses basic probability or statistics to estimate population parameters. ML is often regarded as the only thing data scientists know how to do, and this is simply untrue. A true data scientist is able to recognize when ML is applicable and, more importantly, when it is not.

ML is a game of correlations and relationships. Most ML algorithms in existence are concerned with finding and/or exploiting relationships between datasets (often represented as columns in a `pandas DataFrame`). Once ML algorithms can pinpoint certain correlations, the model can either use these relationships to predict future observations or generalize the data to reveal interesting patterns.

Perhaps a great way to explain ML is to offer an example of a problem coupled with two possible solutions: one using an ML algorithm and the other utilizing a non-ML algorithm.

Example – facial recognition

This problem is, on its face (pun intended), quite simple: given a picture of a face, who does it belong to? However, let's consider a slightly simpler task. Suppose you wish to implement a home security system that recognizes who is entering your house. Most likely, during the day, your house will be empty most of the time, and facial recognition will kick in only if there is a person in the shot. This is exactly the question I propose we try to solve – given a photo, is there a face in it to even recognize?

Given this task definition, I propose the following two solutions:

- A non-ML algorithm that will define a face as having a roundish structure, two eyes, hair, nose, and so on. The algorithm then looks for these hardcoded features in the photo and returns whether or not it was able to find any of these features.
- An ML algorithm that will work a bit differently. The model will only be given several pictures of faces and non-faces that are labeled as such. From the examples (called training sets), it would figure out its own definition of a face.

The ML version of the solution is never told what a face is; it is merely given several examples – some with faces, and some without. It is then up to the ML model to figure out the difference between the two. Once it figures this out, it uses this information to take in a picture and predict whether or not there is a face in the new picture. For example, to train the system, we might have the following images denoted in *Figure 10.1*:



Face



Face



No Face

Figure 10.1 – Input images for training an ML model

The model will then figure out the difference between the pictures labeled as *Face* and the pictures labeled as *No Face* and be able to use that difference to find faces in future photos. Because the promise of ML – learning simply from data and without explicit human intervention – is so alluring, many people might believe that ML is perfect, but it simply isn't.

ML isn't perfect

There are many caveats of ML. Many are specific to different models being implemented, but some assumptions are universal for any ML model:

- The data used, for the most part, is preprocessed and cleaned using the methods outlined in the earlier chapters. Almost no ML model will tolerate extremely dirty/incomplete data with missing values or categorical values.
- Each row of a cleaned dataset represents a single observation of the environment we are trying to model.
- The data as a whole should be representative of the task we are solving. This might sound obvious, but in so many cases, people use data to train an ML model that is close to but not exactly related to the task. This is often seen in criminal justice examples where people might use arrest data to train a model to predict criminality but, of course, arrests are not the same as convicting someone of a crime.
- If our goal is to find relationships between variables, then there is an assumption that there is some kind of relationship between these variables. Again, this seems obvious, but if a human putting the data together is biased and “believes” there is a relationship between the data, then they might incorrectly judge an ML model to be more powerful than it actually is.

This assumption is particularly important. Many ML models take this assumption very seriously. These models are not able to communicate that there might not be a relationship.

- ML models are generally considered semi-automatic, which means that intelligent decisions by humans are still needed.
- The machine is very smart but has a hard time putting things into context. The output of most models is a series of numbers and metrics attempting to quantify how well the model did. It is up to a human to put these metrics into perspective and communicate the results to an audience.
- Most ML models are sensitive to noisy data. This means that the models get confused when you include data that doesn't make sense. For example, if you are attempting to find relationships between economic data around the world and one of your columns relates to puppy adoption rates in the capital city, that information is likely not relevant and will confuse the model.

These assumptions will come up again and again when dealing with ML. They are all too important and are often ignored by novice data scientists.

How does ML work?

Each flavor of ML and each individual model works in very different ways, exploiting different parts of mathematics and data science. However, in general, ML works by taking in data, finding relationships within the data, and giving as output what the model learned, as illustrated in *Figure 10.2*.

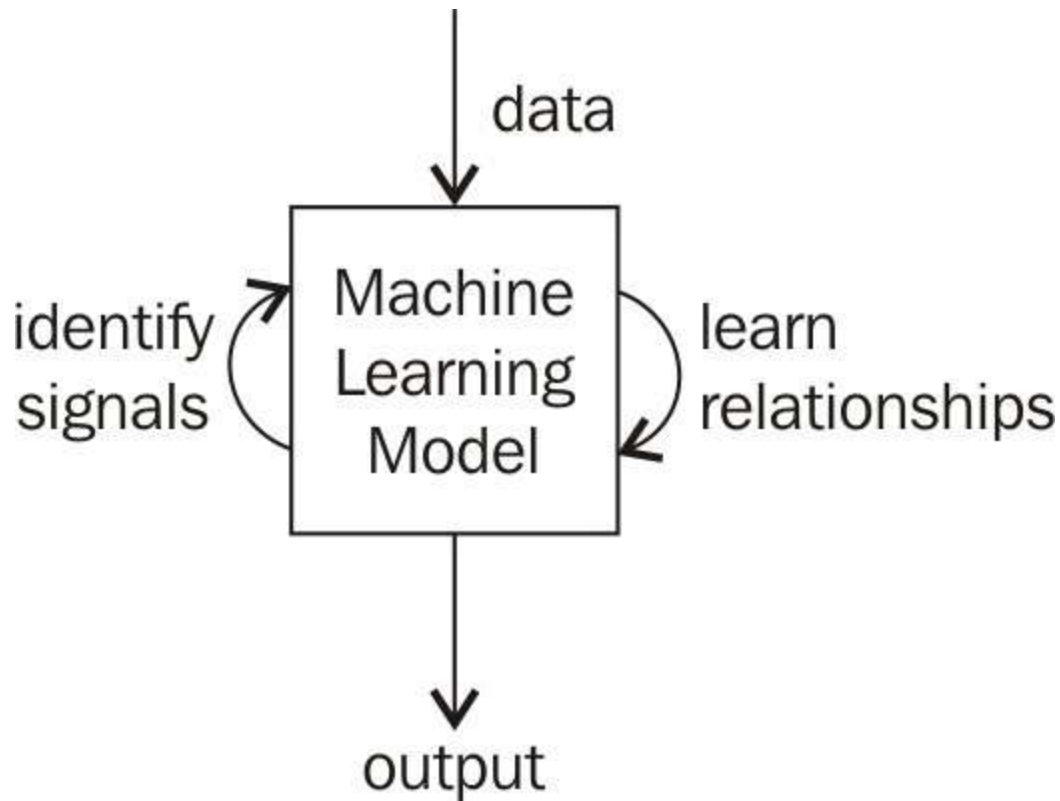


Figure 10.2 – An overview of ML models taking in input data, learning signals, and identifying patterns in order to produce a meaningful and interpretable output

As we explore different types of ML models, we will see how they manipulate data differently and come up with different outputs for different applications.

Types of ML

There are many ways to segment ML and dive deeper. In [Chapter 1, Data Science Terminology](#), I mentioned statistical and probabilistic models. These models utilize statistics and probability, which we've seen in the previous chapters, in order to find relationships between data and make predictions. In this chapter, we will implement both types of models. In the following chapter, we will see ML outside the rigid mathematical world of statistics/probability. You can segment ML models by different characteristics, including the following:

- The types of data organic structures they utilize (tree, graph, or **neural network (NN)**)
- The field of mathematics they are most related to (statistical or probabilistic)
- The level of computation required to train (**deep learning (DL)**)

Branching off from the top level of ML, there are the following three subsets:

- **Supervised learning (SL)**
- **Unsupervised learning (UL)**

- **Reinforcement learning (RL)**

Let's go into each one of these one by one. Our next chapter will include multiple examples of the first two, with the third one being slightly out of the scope of our introductory book. You can always find more resources in our code base!

SL

Simply put, **SL** finds associations between features of a dataset (independent variables) and a target (dependent) variable. For example, SL models might try to find the association between a person's health features (heart rate, weight, and so on) and that person's risk of having a heart attack (the target variable). These associations allow supervised models to make predictions based on past examples.

Supervised ML (SML) models are often called predictive analytics models, named for their ability to predict the future based on the past. This is often the first thing that comes to people's minds when they hear the term *ML*, but it in no way encompasses the realm of ML.

SML requires a certain type of data called **labeled data** – data that acts as full, correct, and complete examples of the features and target variable. *Figure 10.1* shows a snippet of labeled data. The goal is to let our model learn by giving it historical examples that are labeled with the correct answer.

Recall the facial recognition example. That is an SL model because we are training our model with the previous pictures labeled as either *face* or *not face*, and then asking the model to predict whether or not a new picture has a face in it.

First, let us separate the data into two distinct parts, as follows:

- The features, which are the columns that will be used to make our prediction. These are sometimes called predictors, input values, variables, and independent variables.
- The response, which is the column that we wish to predict. This is sometimes called outcome, label, target, and dependent variable.

SL attempts to find a relationship between features and responses in order to make a prediction. The idea is that, in the future, a data observation will present itself, and we will only know the predictors. The model will then have to use the features to make an accurate prediction of the response value.

Figure 10.3 shows a visualization of how we generally use supervised models: we train (fit) them using labeled training data and use the result to predict unseen cases (features without the response) to make final predictions:

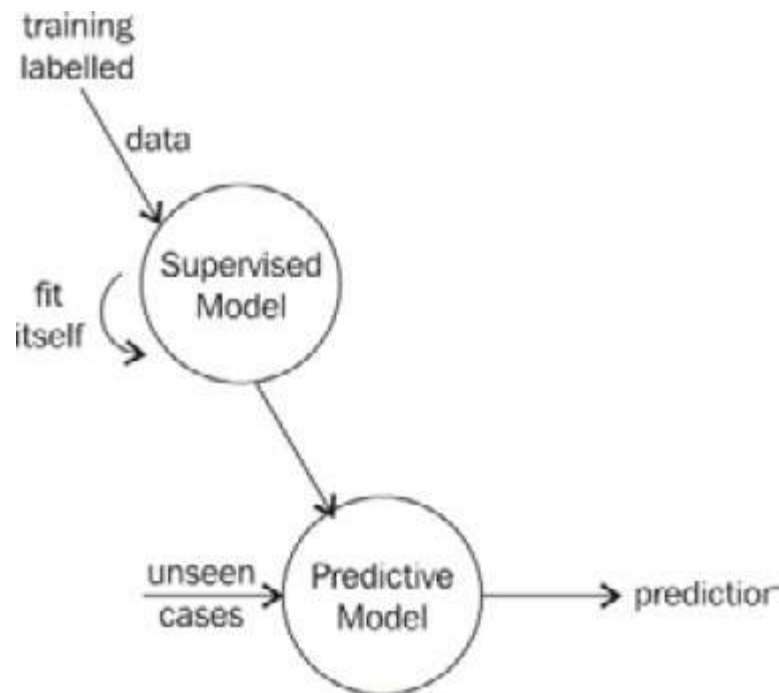


Figure 10.3 – Supervised models are fit using labeled training data and are then used to make predictions from unseen cases

Example – heart attack prediction

Suppose we wish to predict whether someone will have a heart attack within a year. To predict this, we are given that person's cholesterol level, blood pressure, height, smoking habits, and perhaps more. From this data, we must ascertain the likelihood of a heart attack. Suppose, to make this prediction, we look at previous patients and their medical history. As these are previous patients, we know not only their predictors (cholesterol, blood pressure, and so on) but also if they actually had a heart attack (because it already happened!).

This is an SML problem because we are doing the following:

- We are making a prediction about someone
- We are using historical training data to find relationships between medical variables and heart attacks

Figure 10.4 shows a basic outline of how SML models use data:

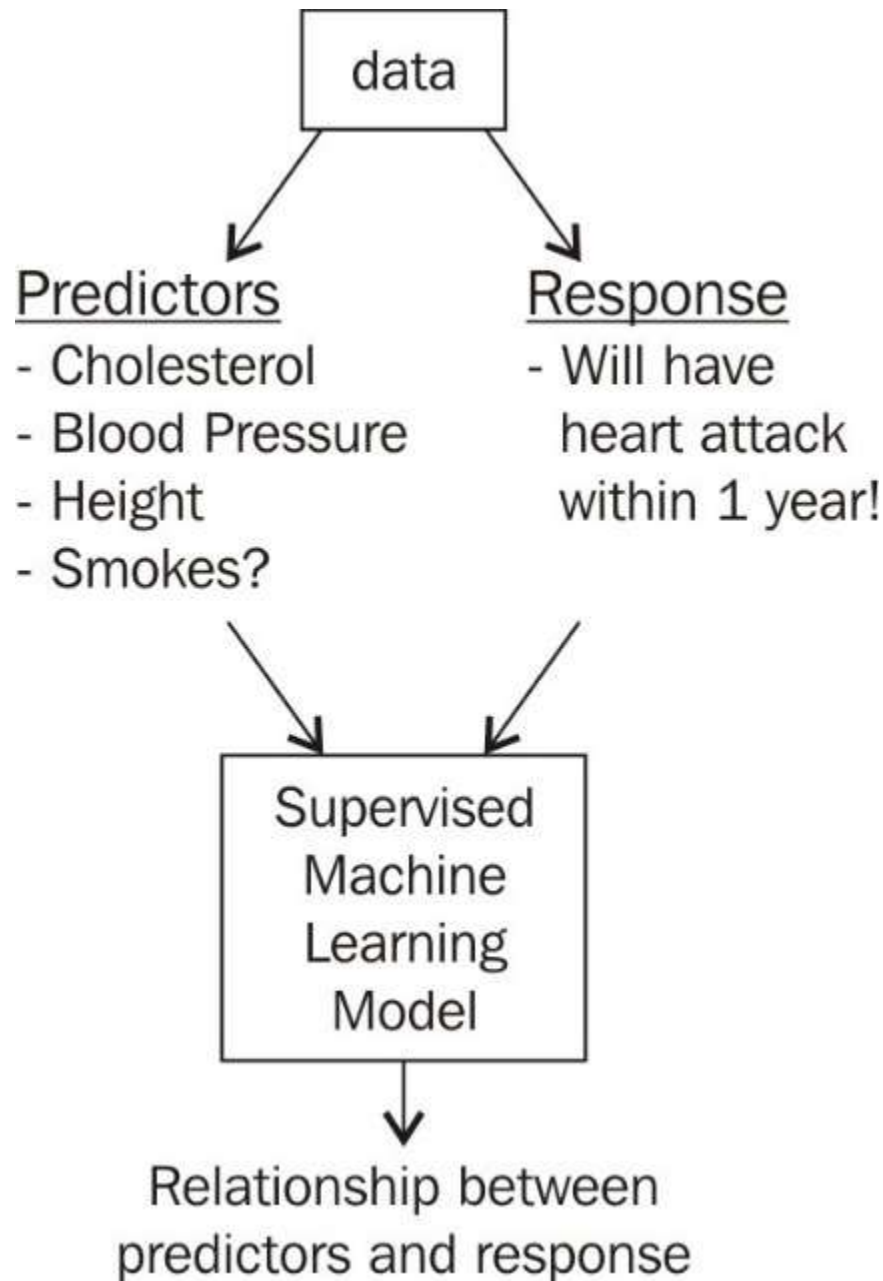


Figure 10.4 – An SML model uses predictors and a response from data in order to learn relationships between them, usually in order to make future predictions given predictors without the response

The hope here is that a patient will walk in sometime in the future and our model will be able to identify whether or not the patient is at risk for a heart attack based on their conditions (just like a doctor would!).

As the model sees more and more diverse and representative labeled data, it should adjust itself in order to match the correct labels outlined in the training data. We can then use different metrics (explained more in the next chapter) to pinpoint exactly how well our SML model is doing and how it can better adjust itself. One of the largest obstacles associated with SML is obtaining diverse and representative

labeled data, which can be very difficult to get hold of. Suppose we wish to predict heart attacks; we might need thousands of patients along with all of their medical information and years' worth of follow-up records for each person, which could be a nightmare to obtain. In short, supervised models use historical labeled data in order to make predictions about the future from predefined features. Some possible applications for SL include the following:

- **Stock price predictions:** Historical trading volume and movements could be features along with social media sentiment, while the future price could be a target
- **Weather predictions:** Using past meteorological data, such as temperature, humidity, and wind speed, to forecast future weather conditions
- **Disease diagnosis:** Medical imaging and patient history can be used to predict the presence or absence of a disease
- **Facial recognition:** Features extracted from images of faces to identify individuals
- **Email filtering:** Using characteristics of emails to classify them as spam or not spam
- **Credit scoring:** Historical financial behavior data to predict creditworthiness

Each of these applications relies on a labeled dataset that includes historical data points and a target variable that the model is trying to predict. The quality and quantity of the labeled data are crucial in SL as they directly impact the model's ability to learn and generalize to new, unseen data. When designing an SL model, it is important to consider the features that will be used. Features should be relevant, informative, and non-redundant to ensure the model performs effectively. Additionally, the choice of algorithm depends on the nature of the task (regression, classification), the size and dimensionality of the dataset, and the computational efficiency required.

By carefully preparing the dataset and selecting the right features and model, SL can provide powerful predictive insights across various fields, from finance to healthcare.

Types of SL models

There are, in general, two types of SL models: **regression** and **classification** models. The difference between the two is quite simple and lies in the nature of the response variable.

Regression models attempt to predict a continuous response. This means that the response can take on a range of infinite values. Consider the following examples:

- **House pricing:** Where the value to be predicted is the cost of a house based on features such as square footage, number of bedrooms, and location
- **Temperature forecasting:** Where the model predicts the temperature for future days or hours based on historical weather data
- **Stock market price prediction:** Where the continuous response could be the future price of a stock based on its historical performance and other economic indicators

Classification models, on the other hand, predict categorical responses. These responses are discrete and have a finite number of values, often referred to as classes or categories. Here are some examples:

- **Email spam detection:** The model classifies emails as either “spam” or “not spam” based on content, sender information, and other attributes
- **Medical diagnosis:** A model might classify patient outcomes based on test results, predicting categories such as “disease” or “no disease”
- **Image recognition:** Classifying images into predefined categories such as “cat,” “dog,” “car,” and so on, based on pixel data and patterns

Both regression and classification tasks use a similar process of learning from historical data, but their applications and evaluation metrics differ due to the nature of their output. Our earlier heart attack example is classification because the question was: Will this person have a heart attack within a year? This has only two possible answers: *Yes* or *No*. We will see a full example of regression later in this chapter and full examples of both classification and regression in the next chapter.

Deciding between classification and regression

Sometimes, it can be tricky to decide whether or not you should use classification or regression. Consider that we are interested in the weather outside. We could ask the question, *How hot is it outside?* In this case, your answer is on a continuous scale, and some possible answers are 60.7 degrees or 98 degrees. However, as an exercise, go and ask 10 people what the temperature is outside. I guarantee you that someone (if not most people) will not answer in some exact degrees but will bucket their answer and say something like *It's in the 60s*.

We might wish to consider this problem as a classification problem, where the response variable is no longer in exact degrees but is in a bucket. There would only be a finite number of buckets in theory, making the model perhaps learn the differences between 60s and 70s a bit better.

What if we aren't predicting anything, but instead we wanted to use ML to simply better understand and interpret our data? That's where UL comes in very handy.

UL

The second type of ML on our list does not deal with making predictions but has a much more open objective. **UL** takes in a set of predictors and utilizes relationships between the predictors in order to accomplish tasks such as the following:

- It reduces the dimension of the data by condensing variables together. An example of this would be file compression. Compression works by utilizing patterns in the data and representing the data in a smaller format. This is referred to as **dimension reduction**.
- It finds groups of observations that behave similarly and groups them together. This is called **clustering**.

Both of these are examples of UL because they do not attempt to find a relationship between predictors and a specific response and therefore are not used to make predictions of any kind. Unsupervised

models, instead, are utilized to find organizations and representations of the data that were previously unknown.

Figure 10.5 gives a representation of a cluster analysis:

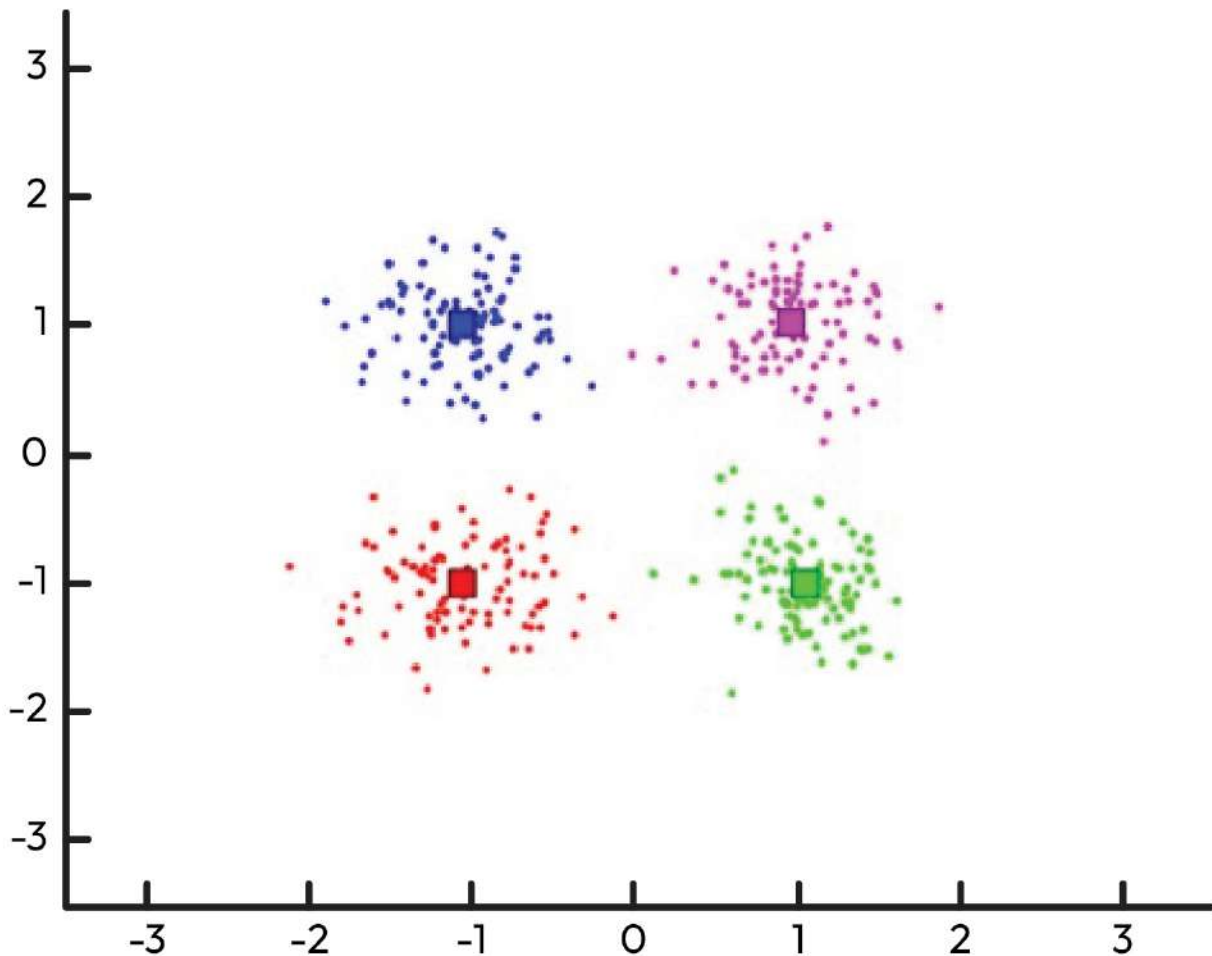


Figure 10.5 – Cluster analysis groups together similar data points to add a layer of interpretation on top of raw data

The model will recognize that each uniquely colored cluster of observations is similar to another but different from the other clusters.

A big advantage of UL is that it does not require labeled data, which means that it is much easier to get data that complies with UL models. Of course, a drawback to this is that we lose all predictive power because the response variable holds the information to make predictions and, without it, our model will be hopeless in making any sort of predictions.

A big drawback is that it is difficult to see how well we are doing. In a regression or classification problem, we can easily tell how well our models are predicting by comparing our models' answers to the actual answers. For example, if our supervised model predicts rain and it is sunny outside, the prediction is incorrect. If our supervised model predicts the price will go up by 1 dollar and it goes up by 99 cents,

our prediction is very close! In unsupervised modeling, this concept is foreign because we have no answer to compare our models to. Unsupervised models merely suggest differences and similarities that then require a human's interpretation, as visualized in *Figure 10.6*:

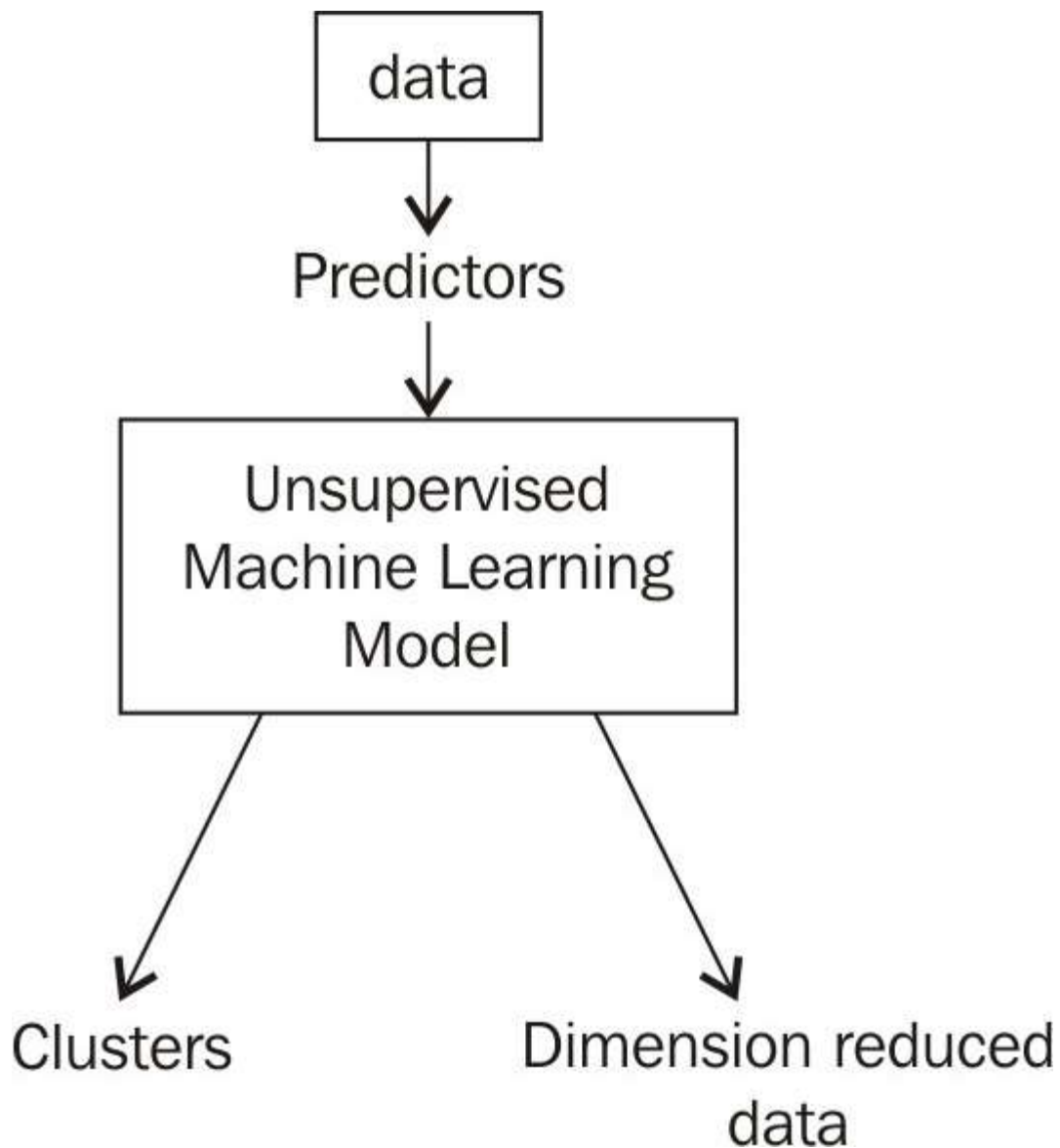


Figure 10.6 – Unsupervised models don't have a notion of a “target” and instead focus on adding a layer of structure on top of raw data

In short, the main goal of unsupervised models is to find similarities and differences between data observations and use these comparisons to add structure on top of otherwise raw and unstructured data.

Moving in a very different direction is our final type of ML. To be honest with you, we don't have the time nor the pages in this book to cover our next topic with the respect it deserves, but it merits a place in our text regardless.

RL

In **RL**, algorithms, referred to as agents, learn to make decisions by interacting with an environment. The agent selects an action to take based on its current state and then receives a reward or penalty based on the outcome of that action. The goal is to learn a policy—a mapping from states to actions—that maximizes the cumulative reward over time.

This type of ML is quite distinct from SL as it does not rely on labeled input/output pairs and does not require explicit correction of suboptimal actions. Instead, it focuses on finding a balance between exploration (trying new actions) and exploitation (using known information to maximize the reward).

RL has been successfully applied in various domains, including the following:

- **Game playing:** AI agents are trained to play and excel at complex games, such as Go, chess, and various video games, often surpassing human expert performance
- **Robotics:** Robots learn to perform tasks such as walking, picking up objects, or navigating through challenging terrain through trial and error
- **Autonomous vehicles:** RL is used to develop systems that can make real-time driving decisions in dynamic and unpredictable environments

OpenAI's pioneering work in using **RL with human feedback (RLHF)** has been instrumental in developing AI models such as ChatGPT. By incorporating human preferences, these models are trained to generate responses that are not only relevant but also aligned with human values, enhancing their helpfulness and minimizing potential harm.

A typical flow for an RL problem would look something like this:

1. The agent receives state S from the environment.
2. The agent takes action A based on policy π .
3. The environment presents a new state S and reward R to the agent.
4. The reward informs the agent of the action's effectiveness.
5. The agent updates policy π to increase future rewards.

Overview of the types of ML

Of the three types of ML – SL, UL, and RL – we can imagine the world of ML as something like the depiction in *Figure 10.7*:

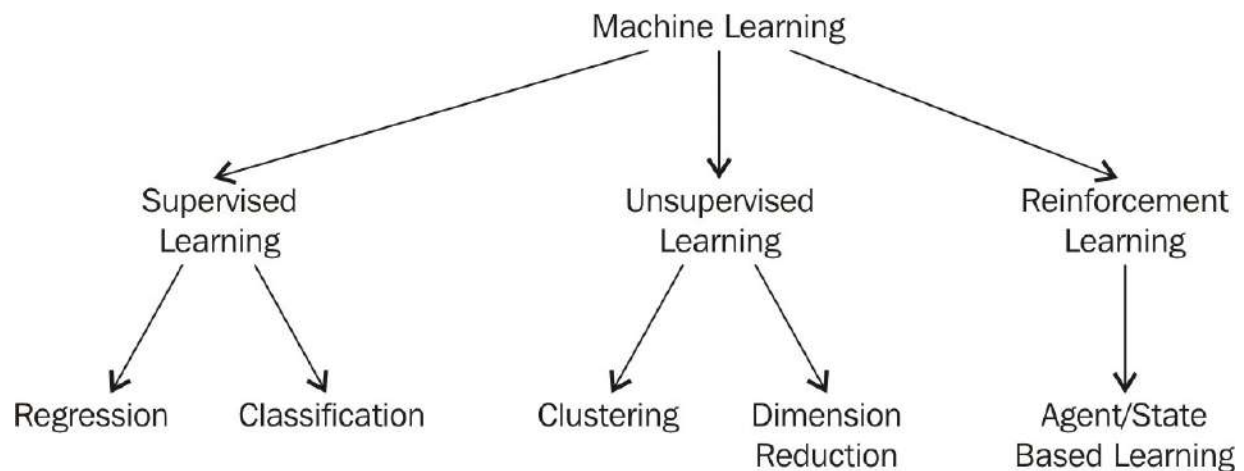


Figure 10.7 – Our family tree of ML has three main branches: SL, UL, and RL

ML paradigms – pros and cons

As we now know, ML can be broadly classified into three categories, each with its own set of advantages and disadvantages.

SML

This method leverages the relationships between input predictors and the output response variable to predict future data observations.

The advantages of it are as follows:

- Enables predictive analysis for future events
- Quantifies the relationships and effects between variables
- Provides insights into how variables interact and influence each other

Let's see the disadvantage:

- Dependent on the availability of labeled data, which can be scarce and expensive to procure

Unsupervised ML (UML)

This approach discovers patterns by finding similarities and differences between data points without using labeled responses.

The advantages of it are as follows:

- Identifies subtle correlations that may not be evident to human analysts
- Serves as a valuable preprocessing step for SL, transforming raw data into structured clusters
- Utilizes unlabeled data, which is generally more abundant and accessible

Here are the disadvantages:

- Lacks direct predictive capabilities
- Validation of the model's efficacy is challenging and highly reliant on human judgment

RL

RL employs a system of rewards to train agents to take optimal actions within their environments.

The advantages of it are as follows:

- Capable of developing complex AI behaviors through intricate reward systems
- Adaptable to a wide range of environments, including real-world scenarios

The disadvantages are as follows:

- Initial behavior can be unpredictable as the agent learns from its mistakes
- Learning can be slow, as the agent may take time to discern beneficial actions from detrimental ones
- There is a risk of the agent becoming overly cautious, limiting its actions to avoid negative outcomes

Enough talk – let's look at our first ML code!

Predicting continuous variables with linear regression

We will finally explore our first true ML model! Linear regression is a form of regression, which means that it is an ML model that attempts to find a relationship between predictors and a response variable, and that response variable is – you guessed it – continuous! This notion is synonymous with making a *line of best fit*. While linear regressions are no longer a state-of-the-art ML algorithm, the path behind it can be a bit tricky and it will serve as an excellent entry point for us.

In the case of linear regression, we will attempt to find a linear relationship between our predictors and our response variable. Formally, we wish to solve a formula of the following format:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n$$

Let's look at the constituents of this formula:

- y is our response variable
- x_i is our i^{th} variable (i^{th} column or i^{th} predictor)
- B_0 is the intercept
- B_i is the coefficient for the x_i term

Let's take a look at some data before we go in depth. This dataset is publicly available and attempts to predict the number of bikes needed on a particular day for a bike-sharing program:

```
# read the data and set the datetime as the index
# taken from Kaggle: https://www.kaggle.com/c/bike-sharing-demand/data
import pandas as pd
import matplotlib.pyplot as plt
```

```
url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/bikeshare.csv'
bikes = pd.read_csv(url) bikes.head()
```

Our data can be seen in *Figure 10.8*:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0	0	1	1

Figure 10.8 – The first five rows (the head) of our bike-share data

We can see that every row represents a single hour of bike usage. In this case, we are interested in predicting the `count` value, which represents the total number of bikes rented in the period of that hour.

Let's use the `seaborn` module to draw ourselves a line of best fit using only the `temp` feature, as follows:

```
import seaborn as sns #using seaborn to get a line of best fit
sns.lmplot(x='temp', y='count', data=bikes, aspect=1.5, scatter_kws={'alpha':0.2})
```

The output of this code can be seen in *Figure 10.9*:

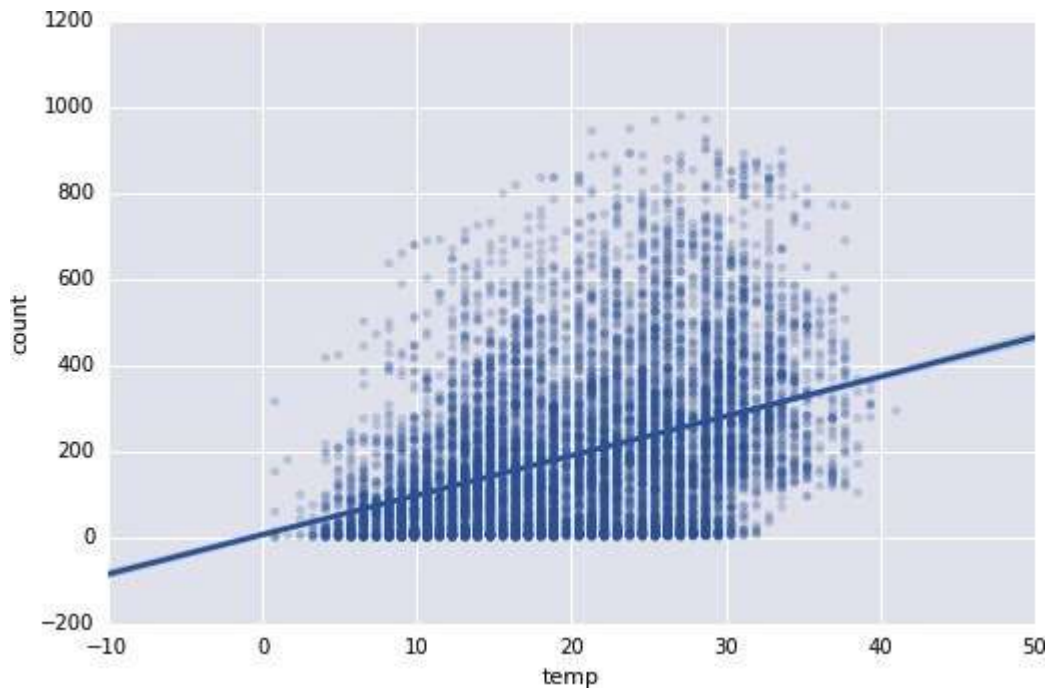


Figure 10.9 – Our first line of best fit showing us the relationship between the temperature and the number of bike shares

This line in the graph attempts to visualize and quantify the relationship between `temp` and `count`. To make a prediction, we simply find a given temperature and then see where the line would predict the count. For example, if the temperature is 20 degrees (Celsius, mind you), then our line would predict that

about 200 bikes will be rented. If the temperature is above 40°C, then more than 400 bikes will be needed!

It appears that as `temp` goes up, our `count` value also goes up. Let's see if our correlation value, which quantifies a linear relationship between variables, also matches this notion:

```
bikes[['count', 'temp']].corr() # 0.3944
```

There is a (weak) positive correlation between the two variables, which makes sense considering our line of best fit! Let's now use `pandas` to create a variable for our features (`x`) and another one for our target (`y`):

```
# create X and y
feature_cols = ['temp'] # a list of the predictors
X = bikes[feature_cols] # subsetting our data to only the predictors
y = bikes['count'] # our response variable
```

Our `x` and `y` variables represent our predictors and our response variable. Then, we will import our ML module, `scikit-learn`, as shown:

```
# import scikit-learn, our machine learning module from sklearn.linear_model import
LinearRegression
```

Finally, we will fit our model to the predictors and the response variable, as follows:

```
linreg = LinearRegression() #instantiate a new model linreg.fit(X, y) #fit the model
to our data
# print the coefficients print(linreg.intercept_) print(linreg.coef_) 6.04621295962 #
our Beta_0
[ 9.17054048] # our beta parameters
```

Let's attempt to interpret this:

- B_0 (6.04) is the value of `y` when `x` = 0
- It is the estimation of bikes that will be rented when the temperature is 0°C
- So, at 0°C, six bikes are predicted to be in use (it's cold!)

Sometimes, it might not make sense to interpret the intercept at all because there might not be a concept of zero in some cases. Recall the levels of data. Not all levels have this notion of zero. Our target variable does have the inherent notion of no bikes; so, we are safe.

Correlation versus causation

In the context of linear regression, coefficients represent the strength and direction of the relationship between the predictor variables and the response variable. However, this statistical relationship should not be confused with causation.

The coefficient *BI*, with a value of 9.17 in our previous code snippet, indicates the average change in the dependent variable (number of bikes rented) for each one-unit change in the independent variable (temperature in °C). Concretely, this means the following:

- For every 1°C increase in temperature, there is an associated average increase of approximately 9 bikes in rentals
- The positive sign of this coefficient suggests a direct relationship: as temperature increases, so do bike rentals

Yet, despite the apparent association indicated by *BI*, we must be cautious. This is a correlation, which means it only indicates that two variables move together—it does not imply that one causes the other to change. A negative coefficient would have suggested an inverse relationship: as temperature rises, bike rentals would decrease. But again, this would not confirm that temperature changes cause changes in bike rentals.

Causation

To make a claim of causation, we would need a controlled experimental design or additional statistical techniques that account for confounding variables and establish a causal link. Without such evidence, our findings from regression analysis should be presented as correlational insights, which highlight patterns that may warrant further investigation but do not confirm causality.

Therefore, while our temperature coefficient *BI* suggests a correlation between warm weather and increased bike rentals, we cannot conclude that warm weather causes more people to rent bikes without a deeper causal analysis.

Now that we are confident in our interpretations of our correlational findings, let's use `scikit-learn` to make some predictions:

```
linreg.predict(20) # a temperature of 20 degrees would lead our model to predict
189.46 bikes to be in use
```

This means that roughly 189 bikes will likely be rented if the temperature is 20°C.

Adding more predictors

Of course, temperature is not the only thing that will help us predict the number of bikes. Adding more predictors to the model is as simple as telling the linear regression model in `scikit-learn` about them!

Before we do, we should look at the data dictionary provided to us to make more sense of some more features:

- **season:** 1 = **spring**, 2 = **summer**, 3 = **fall**, and 4 = **winter**
- **holiday:** Whether the day is considered a holiday
- **workingday:** Whether the day is a weekend or holiday

- **weather:** 1=Clear, Few clouds, Partly cloudy, 2=Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist, 3=Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds, and 4 = Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- **temp:** The temperature in °C
- **atemp:** The “feels like” temperature, taking wind speeds into consideration
- **humidity:** Relative humidity

Now, let’s create our linear regression model using more features. As before, we will first create a list holding the features we wish to look at, create our features and our response datasets (**x** and **y**), and then fit our linear regression. Once we fit our regression model, we will take a look at the model’s coefficients in order to see how our features are interacting with our response:

```
# create a list of features
feature_cols = ['temp', 'season', 'weather', 'humidity'] # create X and y
X = bikes[feature_cols] y = bikes['count']
# instantiate and fit linreg = LinearRegression() linreg.fit(X, y)
# pair the feature names with the coefficients result = zip(feature_cols,
linreg.coef_) resultSet = set(result)
print(resultSet)
his gives us the following output:
[('temp', 7.8648249924774403),
 ('season', 22.538757532466754),
 ('weather', 6.6703020359238048),
 ('humidity', -3.1188733823964974)]
```

And this is what that means:

- Holding all other predictors constant, a 1-unit increase in temperature is associated with a rental increase of **7.86** bikes
- Holding all other predictors constant, a 1-unit increase in season is associated with a rental increase of **22.5** bikes
- Holding all other predictors constant, a 1-unit increase in weather is associated with a rental increase of **6.67** bikes
- Holding all other predictors constant, a 1-unit increase in humidity is associated with a rental decrease of **3.12** bikes

This is interesting.

IMPORTANT NOTE

*Note that, as **weather** goes up (meaning that the weather is getting closer to overcast), the bike demand goes up, as is the case when the season variables increase (meaning that we are approaching winter). This is not what I was expecting at all, frankly!*

While these individual correlations are helpful in many ways, it is crucial to identify metrics to judge our ML system as a whole. Usually, we think about metrics in terms of the task we are performing. Certain metrics are useful for classification, while other metrics are more useful for regression.

Regression metrics

There are usually three main metrics when using regression ML models. They are as follows:

- **Mean Absolute Error (MAE):** This is the average of the absolute errors between the predicted values and the actual values. It's calculated by taking the sum of the absolute values of the errors (the differences between the predicted values \hat{y}_i and the actual values y_i) and then dividing by the number of observations n :

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- **Mean Squared Error (MSE):** This is the average of the squares of the errors between the predicted values and the actual values. It's computed by squaring each error, summing these squares, and then dividing by the number of observations:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- **Root MSE (RMSE):** This is the square root of the MSE. It's obtained by taking the square root of the average of the squared differences between the predicted values and the actual values. RMSE is useful because it scales the errors to the original units of the output variable and can be more interpretable than MSE:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

These metrics are crucial in evaluating the performance of regression models, with each having its own advantages. MAE provides a straightforward average-error magnitude, MSE penalizes larger errors more heavily, and RMSE is particularly sensitive to large errors due to the squaring of the errors.

Let's take a look at implementations in Python:

```
# example true and predicted response values
true = [9, 6, 7, 6]
pred = [8, 7, 7, 12]
# note that each value in the last represents a single prediction for a model
# So we are comparing four predictions to four actual answers
# calculate these metrics by hand! from sklearn import metrics
import numpy as np
print('MAE:', metrics.mean_absolute_error(true, pred)) print('MSE:',
metrics.mean_squared_error(true, pred)) print('RMSE:',
np.sqrt(metrics.mean_squared_error(true, pred)))
```

Here, the output would be as follows:

```
MAE: 2.0
MSE: 9.5
RMSE: 3.08220700148
```

Let's use RMSE to ascertain which columns are helping and which are hindering. Let's start with only using temperature. Note that our procedure will be as follows:

1. Create our **x** and our **y** variables.
2. Fit a linear regression model.
3. Use the model to make a list of predictions based on **x**.
4. Calculate the RMSE between the predictions and the actual values.

Let's take a look at the code:

```
from sklearn import metrics
# import metrics from scikit-learn
```

```
feature_cols = ['temp'] # create X and y
X = bikes[feature_cols] linreg = LinearRegression() linreg.fit(X, y)
y_pred = linreg.predict(X) np.sqrt(metrics.mean_squared_error(y, y_pred)) # RMSE #
Can be interpreted loosely as an average error #166.45
```

Now, let's try it using temperature and humidity, as shown:

```
feature_cols = ['temp', 'humidity'] # create X and y
X = bikes[feature_cols] linreg = LinearRegression() linreg.fit(X, y)
y_pred = linreg.predict(X) np.sqrt(metrics.mean_squared_error(y, y_pred)) # RMSE #
157.79
```

It got better! Let's try using even more predictors, as illustrated:

```
feature_cols = ['temp', 'humidity', 'season', 'holiday', 'workingday', 'windspeed',
'atemp']
# create X and y
X = bikes[feature_cols] linreg = LinearRegression() linreg.fit(X, y)
y_pred = linreg.predict(X) np.sqrt(metrics.mean_squared_error(y, y_pred)) # RMSE #
155.75
```

Even better! At first, this seems like a major triumph, but there is actually a hidden danger here. Note that we are training the line to fit **x** and **y** and then asking it to predict **x** again! This is actually a huge mistake in ML because it can lead to overfitting, which means that our model is merely memorizing the data and regurgitating it back to us.

Imagine that you are a student, and you walk into the first day of class and the teacher says that the final exam is very difficult in this class. In order to prepare you, she gives you practice test after practice test after practice test. The day of the final exam arrives, and you are shocked to find out that every question on the exam is exactly the same as in the practice test! Luckily, you did them so many times that you remember the answer and get 100% on the exam.

The same thing applies here, more or less. By fitting and predicting on the same data, the model is memorizing the data and getting better at it. A great way to combat this **overfitting** problem is to use the train/test approach to fit ML models, which works as illustrated next.

Essentially, we will take the following steps:

1. Split up the dataset into two parts: a training and a test set.
2. Fit our model on the training set and then test it on the test set, just like in school, where the teacher would teach from one set of notes and then test us on different (but similar) questions.
3. Once our model is good enough (based on our metrics), we turn our model's attention toward the entire dataset.
4. Our model awaits new data previously unseen by anyone.

This can be visualized in *Figure 10.10*:

- 1) split dataset
- 2) train model
- 3) test model
- 4) parameter tuning
- 5) choose best model
- 6) train on all data
- 7) make predictions on new data

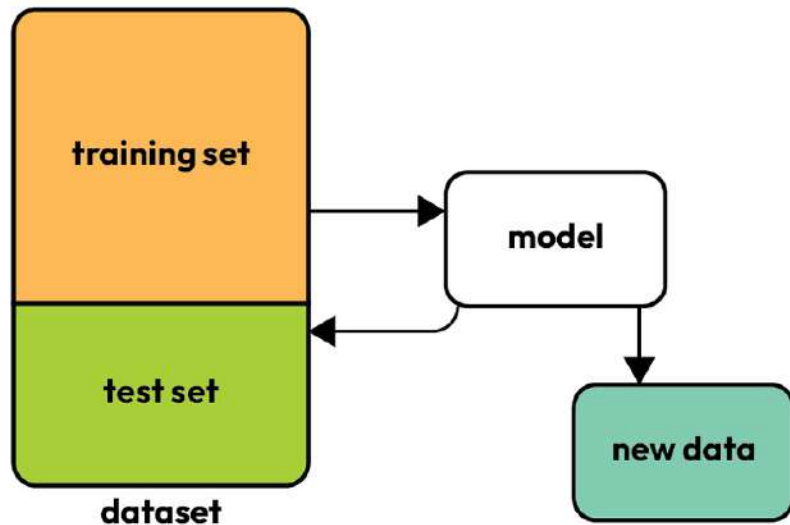


Figure 10.10 – Splitting our data up into a training and testing set helps us properly evaluate our model's ability to predict unseen data

The goal here is to minimize the out-of-sample errors of our model, which are errors our model has on data that it has never seen before. This is important because the main idea (usually) of a supervised model is to predict outcomes for new data. If our model is unable to generalize from our training data and use that to predict unseen cases, then our model isn't very good.

The preceding diagram outlines a simple way of ensuring that our model can effectively ingest the training data and use it to predict data points that the model itself has never seen. Of course, as data scientists, we know that the test set also has answers attached to it, but the model doesn't know that.

All of this might sound complicated, but luckily, the `scikit-learn` package has a built-in method to do this, as shown:

```

from sklearn.cross_validation import train_test_split
# function that splits data into training and testing sets
# setting our overall data X, and y
feature_cols = ['temp']
X = bikes[feature_cols]
y = bikes['count']
# Note that in this example, we are attempting to find an association between only
the temperature of the day and the number of bike rentals.
X_train, X_test, y_train, y_test = train_test_split(X, y) # split the data into
training and testing sets
# X_train and y_train will be used to train the model # X_test and y_test will be
used to test the model
# Remember that all four of these variables are just subsets of the overall X and y.
linreg = LinearRegression() # instantiate the model
linreg.fit(X_train, y_train)
# fit the model to our training set
y_pred = linreg.predict(X_test) # predict our testing set
np.sqrt(metrics.mean_squared_error(y_test, y_pred)) # RMSE # Calculate our metric:
# == 166.91
  
```


In other words, our `train_test_split` function is ensuring that the metrics we are looking at are more honest estimates of our sample performance.

Now, let's try again with more predictors, as follows:

```
feature_cols = ['temp', 'workingday']
X = bikes[feature_cols]
y = bikes['count']
X_train, X_test, y_train, y_test = train_test_split(X, y) # Pick a new random
training and test set
linreg = LinearRegression() linreg.fit(X_train, y_train) y_pred =
linreg.predict(X_test) # fit and predict
np.sqrt(metrics.mean_squared_error(y_test, y_pred)) # 166.95
```

Our model actually got worse with that addition! This implies that **workingday** might not be very predictive of our response, the bike rental count.

All of this is well and good, and we can keep adding and removing features to lower our RMSE, but how well is our model really doing at predicting rather than just guessing? We have an RMSE of around 167 bikes, but is that good? What do we compare it to? One way to discover this is to evaluate the null model.

The **null model** in SML represents effectively guessing the expected outcome over and over, and seeing how you did. For example, in regression, if we always guess the average number of hourly bike rentals, then how well would that model do?

1. First, let's get the average hourly bike rental, as shown:

```
average_bike_rental = bikes['count'].mean() average_bike_rental
# 191.57
```

This means that, overall, in this dataset, regardless of weather, time, day of the week, humidity, and everything else, the average number of bikes that go out every hour is about 192.

2. Let's make a fake prediction list, wherein every single guess is 191.57. Let's make this guess for every single hour, as follows:

```
num_rows = bikes.shape[0] num_rows
# 10886
null_model_predictions = [average_bike_rental] * num_rows null_model_predictions
```

The output is as follows:

```
[191.57413191254824,
191.57413191254824,
191.57413191254824,
191.57413191254824,
```

```
...  
191.57413191254824,  
191.57413191254824,  
191.57413191254824,  
191.57413191254824]
```

So, we have **10,886** values, all of which are the average hourly bike rental number. Let's see what the RMSE would be if our model only ever guessed the expected value of the average hourly bike rental count:

```
np.sqrt(metrics.mean_squared_error(y, null_model_predictions))
```

The output is as follows:

```
181.13613
```

This means that by simply guessing the average value over and over again, our RMSE would be 181 bikes. So, even with one or two features, we can beat it! Beating the null model is a kind of baseline in ML. If you think about it, why go through any effort at all if your ML is not even better than just guessing?

Summary

In this chapter, we looked at ML and its different subcategories. We explored SL, UL, and RL strategies and looked at situations where each one would come in handy.

Looking into linear regression, we were able to find relationships between predictors and a continuous response variable. Through the train/test split, we were able to help avoid overfitting our ML models and get a more generalized prediction. We were able to use metrics, such as RMSE, to evaluate our models as well.

In the next few chapters, we will be taking a much deeper dive into many more ML models and, along the way, we will learn new metrics, new validation techniques, and – more importantly – new ways of applying data science to the world.

Predictions Don't Grow on Trees, or Do They?

Our goal in this chapter is to see and apply concepts learned from previous chapters in order to construct and use modern learning algorithms to glean insights and make predictions on real datasets. While we explore the following algorithms, we should always remember that we are constantly keeping our metrics in mind.

In this chapter, we will be looking at the following ML algorithms:

- Performing naïve Bayes classification
- Understanding decision trees
- Diving deep into **unsupervised learning (UL)**
- k-means clustering
- Feature extraction and **principal component analysis (PCA)**

Performing naïve Bayes classification

Let's get right into it! Let's begin with **naïve Bayes** classification. This ML model relies heavily on results from previous chapters, specifically with Bayes' theorem:

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)}$$

Let's look a little closer at the specific features of this formula:

- $P(H)$ is the probability of the hypothesis before we observe the data, called the *prior probability*, or just *prior*
- $P(H|D)$ is what we want to compute: the probability of the hypothesis after we observe the data, called the *posterior*
- $P(D|H)$ is the probability of the data under the given hypothesis, called the *likelihood*
- $P(D)$ is the probability of the data under any hypothesis, called the *normalizing constant*

Naïve Bayes classification is a classification model, and therefore a supervised model. Given this, what kind of data do we need – labeled or unlabeled data?

(Insert *Jeopardy* music here)

If you answered *labeled data*, then you're well on your way to becoming a data scientist!

Suppose we have a dataset with n features, (x_1, x_2, \dots, x_n) , and a *class label*, C . For example, let's take some data involving spam text classification. Our data would consist of rows of individual text samples and columns of both our features and our class labels. Our features would be words and phrases that are contained within the text samples, and our class labels are simply *spam* or *not spam*. In

this scenario, I will replace the not-spam class with an easier-to-say word, ham. Let's take a look at the following code snippet to better understand our spam and ham data:

```
import pandas as pd
import sklearn
df = pd.read_table('../data/sms.tsv',
sep='\t', header=None, names=['label', 'msg'])
df
```

Figure 11.1 is a sample of text data in a row-column format:

	label	msg
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...

Figure 11.1 – A sample of our spam versus not spam (ham) messages

Let's do some preliminary statistics to see what we are dealing with. Let's see the difference in the number of ham and spam messages at our disposal:

```
df.label.value_counts().plot(kind="bar")
```

This gives us a bar chart, as shown in Figure 11.2:

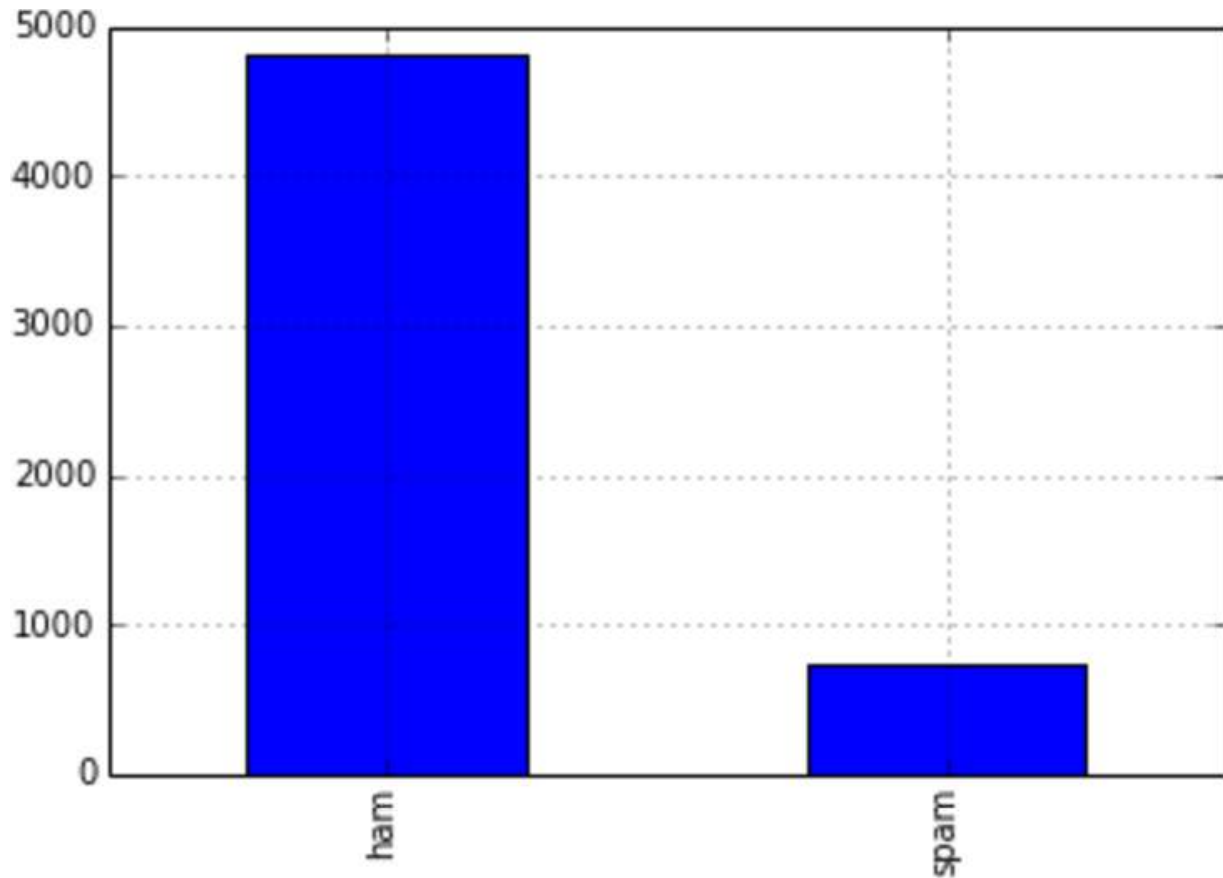


Figure 11.2 – The distribution of ham versus spam

Because we are dealing with classification, it would help to itemize some of the metrics we will be using to evaluate our model.

Classification metrics

When evaluating classification models, different metrics are used compared to regression models. These metrics help to understand how well the model is performing, especially in terms of correctly predicting different classes. Let's look at what they are:

1. **Accuracy:** This is the most intuitive performance measure, and it is simply a ratio of correctly predicted observations to the total observations. It's suitable for binary and multiclass classification problems:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

2. **Precision (best for binary classification – with only two classes):** Also known as positive predictive value, this metric helps to answer the question: “What proportion of positive identifications was actually correct?”

$$\text{Precision} = \frac{TP}{TP + FP}$$

Here, TP is the number of true positives (predicted positive and the prediction was correct), and FP is the number of false positives (predicted positive but the prediction was incorrect).

3. Recall (Sensitivity) (best for binary classification – with only two classes): This metric helps to answer the question: “What proportion of actual positives was identified correctly?”

$$\text{Recall} = \frac{TP}{TP + FN}$$

Here, TP (predicted positive and the prediction was correct) is the number of true positives, and FN is the number of false negatives (predicted negative but the prediction was incorrect).

4. F1 Score: The $F1$ Score is the weighted average of precision and recall. Therefore, this score takes both false positives and false negatives into account. It is a good way to show that a classifier has a good value for both precision and recall:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

These metrics are crucial for understanding the behavior of classification models, especially in domains where the costs of false positives and false negatives are very different.

So, we have way more *ham* messages than we do *spam*. Because this is a classification problem, it would be very useful to know our **null accuracy rate**, which is the percentage chance of predicting a single row correctly if we keep guessing the most common class, **ham**. Here’s how we do that:

```
df.label.value_counts() / df.shape[0]
ham 0.865937
spam 0.134063
```

So, if we blindly guessed *ham*, we would be correct about 87% of the time, but we can do better than that.

If we have a set of classes, C , and features, y_i , then we can use Bayes’ theorem to predict the probability that a single row belongs to class C , using the following formula:

$$P(\text{class } C | \{x_i\}) = \frac{P(\{x_i\} | \text{class } C) \cdot P(\text{class } C)}{P(\{x_i\})}$$

Let’s look at this formula in a little more detail:

- $P(\text{class } C | \{x_i\})$: The posterior probability is the probability that the row belongs to class C given the features $\{x_i\}$.
- $P(\{x_i\} | \text{class } C)$: This is the likelihood that we would observe these features given that the row was in class C .
- $P(\text{class } C)$: This is the prior probability. It is the probability that the data point belongs to class C before we see any data.
- $P(\{x_i\})$: This is our normalization constant.

For example, imagine we have an email with three words: *send cash now*. We’ll use naïve Bayes to classify the email as either being spam or ham:

$$P(\text{spam} | \text{send cash now}) = \frac{P(\text{send cash now} | \text{spam}) \cdot P(\text{spam})}{P(\text{send cash now})}$$

$$P(\text{ham} \mid \text{send cash now}) = \frac{P(\text{send cash now} \mid \text{ham}) \cdot P(\text{ham})}{P(\text{send cash now})}$$

We are concerned with the difference of these two numbers. We can use the following criteria to classify any single text sample:

- If $P(\text{spam} \mid \text{send cash now})$ is larger than $P(\text{ham} \mid \text{send cash now})$, then we will classify the text as spam
- If $P(\text{ham} \mid \text{send cash now})$ is larger than $P(\text{spam} \mid \text{send cash now})$, then we will label the text ham

Because both equations have $P(\text{send money now})$ in the denominator, we can ignore them. So, now we are concerned with the following:

$$P(\text{send cash now} \mid \text{spam}) \cdot P(\text{spam}) \text{ VS } P(\text{send cash now} \mid \text{ham}) \cdot P(\text{ham})$$

Let's work out the numbers in this equation:

- $P(\text{spam}) = 0.134063$
- $P(\text{ham}) = 0.865937$
- $P(\text{send cash now} \mid \text{spam}) = ???$
- $P(\text{send cash now} \mid \text{ham}) = ???$

The final two likelihoods might seem like they would not be so difficult to calculate. All we have to do is count the number of spam messages that include the send money, right?

Now, phrase and divide that by the total number of spam messages:

```
df.msg = df.msg.apply(lambda x:x.lower())
# make all strings lower case so we can search easier
df[df.msg.str.contains('send cash now')].shape # == (0, 2)
```

Oh no! There are none! There are literally zero texts with the exact phrase *send cash now*. The hidden problem here is that this phrase is very specific, and we can't assume that we will have enough data in the world to have seen this exact phrase many times before.

Instead, we can make a naïve assumption in our Bayes' theorem. If we assume that the features (words) are conditionally independent (meaning that no word affects the existence of another word), then we can rewrite the formula:

$$P(\text{send cash now} \mid \text{spam}) = P(\text{send} \mid \text{spam}) \cdot P(\text{cash} \mid \text{spam}) \cdot P(\text{now} \mid \text{spam})$$

And here's what it looks like done in Python:

```
spams = df[df.label == 'spam']
for word in ['send', 'cash', 'now']:
    print( word, spams[spams.msg.str.contains(word)].shape[0] / float(spams.shape[0]))
```

Printing out the conditional probabilities yields:

- $P(\text{send} \mid \text{spam}) = 0.096$
- $P(\text{cash} \mid \text{spam}) = 0.091$

- $P(\text{now}|\text{spam}) = 0.280$

With this, we can calculate the following:

$$P(\text{send cash now}|\text{spam}) \cdot P(\text{spam}) = (0.096 \cdot 0.091 \cdot 0.280) \cdot 0.134 = 0.00032$$

Repeating the same procedure for ham gives us the following:

- $P(\text{send}|\text{ham}) = 0.03$
- $P(\text{cash}|\text{ham}) = 0.003$
- $P(\text{now}|\text{ham}) = 0.109$

The fact that these numbers are both very low is not as important as the fact that the spam probability is much larger than the ham calculation. If we do the calculations, we get that the *send cash now* probability for spam is 38 times bigger than for spam! Doing this means that we can classify *send cash now* as spam! Simple, right?

Let's use Python to implement a naïve Bayes classifier without having to do all of these calculations ourselves.

First, let's revisit the count vectorizer in scikit-learn, which turns text into numerical data for us. Let's assume that we will train on three documents (sentences), in the following code snippet:

```
# simple count vectorizer example
from sklearn.feature_extraction.text import CountVectorizer # start with a simple
example
train_simple = ['call you tonight',
'Call me a cab',
'please call me... PLEASE 44!']
# learn the 'vocabulary' of the training data vect = CountVectorizer()
train_simple_dtm = vect.fit_transform(train_simple)
pd.DataFrame(train_simple_dtm.toarray(), columns=vect.get_feature_names())
```

Figure 11.3 demonstrates the feature vectors learned from our dataset:

	44	cab	call	me	please	tonight	you
0	0	0	1	0	0	1	1
1	0	1	1	1	0	0	0
2	1	0	1	1	2	0	0

Figure 11.3 – The first five rows of our SMS dataset after breaking up each text into a count of unique words

Note that each row represents one of the three documents (sentences), each column represents one of the words present in the documents, and each cell contains the number of times each word appears in each document.

We can then use the count vectorizer to transform new incoming test documents to conform with our training set (the three sentences):

```
# transform testing data into a document-term matrix (using existing vocabulary,
notice don't is missing)
test_simple = ["please don't call me"] test_simple_dtm = vect.transform(test_simple)
test_simple_dtm.toarray()
pd.DataFrame(test_simple_dtm.toarray(), columns=vect.get_feature_names())
```

Figure 11.4 is shown as follows:

	44	cab	call	me	please	tonight	you
0	0	0	1	1	1	0	0

Figure 11.4 – Representation of the “please don’t call me” SMS in the same vocabulary as our training data

Note how, in our test sentence, we had a new word – namely, *don’t*. When we vectorized it, because we hadn’t seen that word previously in our training data, the vectorizer simply ignored it. This is important and incentivizes data scientists to obtain as much data as possible for their training sets.

Now, let’s do this for our actual data:

```
# split into training and testing sets
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.msg, df.label, random_state=1)
# instantiate the vectorizer vect = CountVectorizer()
# learn vocabulary and create document-term matrix in a single step train_dtm =
vect.fit_transform(X_train)
train_dtm
```

The following is the output:

```
<4179x7456 sparse matrix of type '<class 'numpy.int64'>' with 55209 stored elements
in Compressed Sparse Row format>
```

Note that the format is in a sparse matrix, meaning the matrix is large and full of zeros. There is a special format to deal with objects such as this. Take a look at the number of columns.

There are 7,456 words. That’s a lot! This means that in our training set, there are 7,456 unique words to look at. We can now transform our test data to conform to our vocabulary:

```
# transform testing data into a document-term matrix test_dtm =
vect.transform(X_test)
test_dtm
```

The output is as follows:

```
<1393x7456 sparse matrix of type '<class 'numpy.int64'>'
with 17604 stored elements in Compressed Sparse Row format>
```

Note that we have the same exact number of columns because it is conforming to our test set to be exactly the same vocabulary as before. No more, no less.

Now, let's build a naïve Bayes model (similar to the linear regression process):

```
## MODEL BUILDING WITH NAIVE BAYES
# train a Naive Bayes model using train_dtm from sklearn.naive_bayes import
MultinomialNB # import our model
nb = MultinomialNB()
# instantiate our model
nb.fit(train_dtm, y_train)
# fit it to our training set
```

Now, the `nb` variable holds our fitted model. The training phase of the model involves computing the likelihood function, which is the conditional probability of each feature given each class:

```
# make predictions on test data using test_dtm preds = nb.predict(test_dtm)
preds
```

The output is as follows:

```
array(['ham', 'ham', 'ham', ..., 'ham', 'spam', 'ham'], dtype='<S4>')
```

The prediction phase of the model involves computing the posterior probability of each class given the observed features and choosing the class with the highest probability.

We will use `sklearn`'s built-in accuracy and confusion matrix to look at how well our naïve Bayes models are performing:

```
# compare predictions to true labels from sklearn import metrics
print metrics.accuracy_score(y_test, preds) print metrics.confusion_matrix(y_test,
preds)
```

The output is as follows:

```
accuracy == 0.988513998564
confusion matrix ==
[[12035]
 [11174]]
```

First off, our accuracy is great! Compared to our null accuracy, which was 87%, 99% is a fantastic improvement.

Now to our confusion matrix. From before, we know that each row represents actual values while columns represent predicted values, so the top-left value, 1,203, represents our true negatives. But what is negative and positive? We gave the model the spam and ham strings as our classes, not positive and negative.

We can use the following:

```
nb.classes_
```

The output is as follows:

```
array(['ham', 'spam'])
```

We can then line up the indices so that 1,203 refers to true ham predictions and 174 refers to true spam predictions. There were also five false spam classifications, meaning that five messages were predicted as spam but were actually ham, as well as 11 false ham classifications. In summary, naïve Bayes classification uses Bayes' theorem in order to fit posterior probabilities of classes so that data points are correctly labeled as belonging to the proper class.

Every ML model has its own set of unique properties and advantages or disadvantages for use with different types of data. The naïve Bayes classifier, for example, is known for its speed and efficiency. It is particularly fast when fitting to training data and when making predictions on test data. This is due to its assumption of feature independence, which simplifies the calculations involved in probability estimation.

However, this same assumption can also be seen as a limitation. In reality, features often do exhibit some level of dependency, and the naïve Bayes classifier may oversimplify complex relationships in data. Moreover, it is based on the assumption that the form of the data distribution (often assumed to be Gaussian) holds true, which might not always be the case in real-world scenarios.

Despite these limitations, naïve Bayes can perform exceptionally well with appropriate data and is particularly useful for text classification tasks, such as spam filtering and **sentiment analysis (SA)**.

Another widely used ML technique is the decision tree. Decision trees are a **supervised learning (SL)** method used for classification and regression. They are intuitive and easy to interpret since they mimic human decision-making more closely than other algorithms.

Understanding decision trees

Decision trees are supervised models that can either perform regression or classification. They are a flowchart-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (for classification) or a value (for regression). One of the primary advantages of decision trees is their simplicity; they do not require any complex mathematical formulations, making them easier to understand and visualize.

The goal of a decision tree is to split the data in a manner that maximizes the purity of the nodes resulting from those splits. In the context of a classification problem, “purity” refers to how homogeneous the nodes are with respect to the target variable. A perfectly pure node would contain instances of only a single class.

Decision trees achieve this by using measures of impurity, such as the Gini index or entropy (more on that soon), to evaluate potential splits. A good split is one that most effectively separates the data into nodes with high purity, meaning that it increases the homogeneity of the nodes with respect to the target variable.

The process involves the following:

1. Selecting the best attribute to split the data based on a specific criterion (such as Gini or entropy).
2. Partitioning the dataset into subsets that contain instances with similar values for that attribute.
3. Repeating this process recursively for each derived subset until the stopping criteria are met (which could be a maximum depth of the tree, a minimum number of instances in a node, or the achievement of a node with high purity).

This recursive partitioning makes decision trees a powerful and interpretable modeling technique for classification and regression tasks.

Measuring purity

The **Gini index** is a measure of inequality among values of a frequency distribution (for example, levels of income). In the context of ML and decision trees, it measures the impurity of a node with the following formula:

$$\text{Gini}(D) = 1 - \sum_{i=1}^J p_i^2$$

Here, D is the dataset, J is the number of classes, and p_i is the probability of class i in the dataset D .

Entropy, on the other hand, is a measure from information theory that quantifies the amount of uncertainty or randomness in the data. It's used in the construction of decision trees to represent the impurity of a dataset with the following formula:

$$\text{Entropy}(D) = - \sum_{i=1}^J p_i \log_2(p_i)$$

Here, p_i is the probability of class i within the dataset D .

Both the Gini index and entropy are used to choose where to split the data when building a decision tree. The choice between using the Gini index and entropy often depends on the specific dataset and the preferences of the modeler, as they can lead to slightly different trees. In practice, the difference in the trees generated by these two methods is often very small.

Exploring the Titanic dataset

The *Titanic* dataset is truly a classic in the field of data science, often used to illustrate the fundamentals of ML. It details the tragic sinking of the RMS Titanic, one of the most infamous shipwrecks in history.

This dataset serves as a rich source of demographic and travel information about the passengers, which can be utilized to model and predict survival outcomes.

Through the lens of this dataset, we can apply statistical analysis and predictive modeling to understand factors that may have influenced the chances of survival. For instance, consider a subset of only 25 passengers from the *Titanic* dataset. Out of 25, 10 of these individuals survived the disaster, while 15 did not. By examining attributes such as age, gender, class, and fare paid, we can begin to construct a predictive model that estimates the likelihood of survival for each passenger in similar circumstances.

We first calculate the *Gini index* before doing anything.

In this example, overall classes are **survived** and **died**, illustrated in the following formula:

$$Gini = 1 - \left(\frac{\text{survived}}{\text{total}} \right)^2 - \left(\frac{\text{died}}{\text{total}} \right)^2 \quad Gini = 1 - \left(\frac{10}{25} \right)^2 - \left(\frac{15}{25} \right)^2 = 0.48$$

This Gini index of 0.48 indicates the level of impurity in the dataset. The value suggests a moderate separation between the classes, with some degree of mixture between the *survived* and *died* categories within this group of passengers.

If we were to make a split in the dataset based on a certain feature, we would calculate the Gini index for each resulting subset. The goal is to choose a split that minimizes the Gini index, thus increasing the purity of the subsets with respect to the target variable, which in this case is survival on the Titanic.

Now, let's consider a potential split on gender. We first calculate the Gini index for each given gender, as seen in *Figure 11.5*:

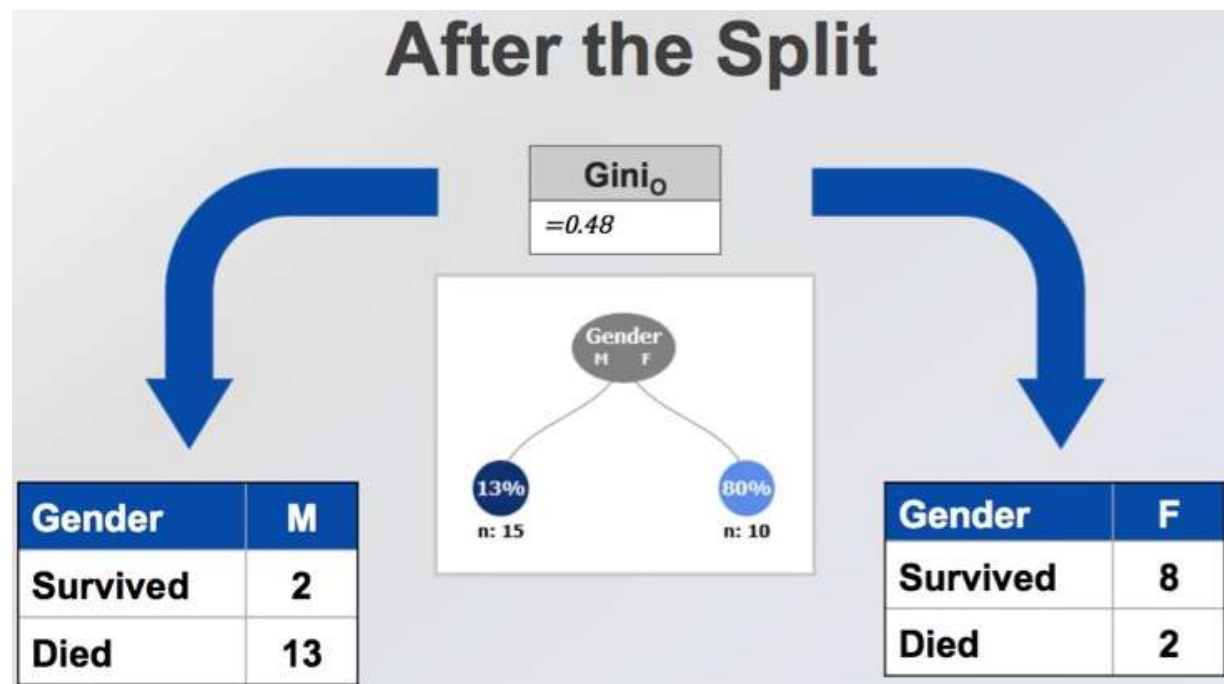


Figure 11.5 – Calculating impurity of our dataset on gender

The following formula calculates the Gini index for male and female, as follows:

$$\text{Gini}(m) = 1 - \left(\frac{2}{15}\right)^2 - \left(\frac{13}{15}\right)^2 = 0.23$$

$$\text{Gini}(f) = 1 - \left(\frac{8}{10}\right)^2 - \left(\frac{2}{10}\right)^2 = 0.32$$

Once we have the Gini index for each gender, we then calculate the overall Gini index for the split on gender, as follows:

$$\text{Gini}(M)\left(\frac{M}{M+F}\right) + \text{Gini}(F)\left(\frac{F}{M+F}\right) = 0.23\left(\frac{15}{10+15}\right) + 0.32\left(\frac{10}{10+15}\right) = 0.27$$

So, the *Gini coefficient* for splitting on *gender* is 0.27. We then follow this procedure for three potential splits (shown in *Figure 11.6*):

- Gender (male or female)
- Number of siblings on board (0 or 1+)
- Class (first and second versus third)

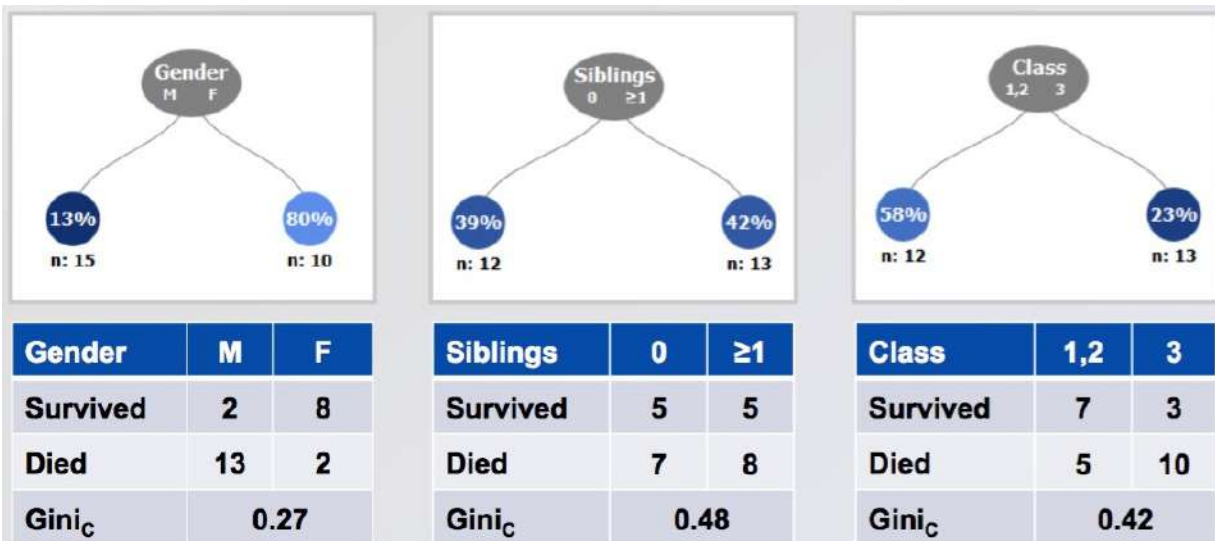


Figure 11.6 – Calculating the resulting Gini coefficient for multiple splits on our dataset to decide which one to use for our decision tree

In this example, we would choose the gender to split on as it has the *lowest Gini index*!

Before we get to some more code, we need to think about how to deal with categorical features that are not numerically encoded. ML algorithms require numerical inputs, and most datasets will have at least one feature that is not numerical.

Dummy variables


Dummy variables are used when we are hoping to convert a categorical feature into a quantitative one. Remember that we have two types of categorical features: nominal and ordinal. Ordinal features have

natural order among them, while nominal data does not.

Encoding qualitative (nominal) data using separate columns is called making dummy variables, and it works by turning each unique category of a nominal column into its own column that is either `true` or `false`.

For example, if we had a column for someone's college major and we wished to plug that information into linear or logistic regression, we couldn't because they only take in numbers! So, for each row, we had new columns that represent the single nominal column. In this case, we have four unique majors: computer science, engineering, business, and literature. We end up with three new columns (we omit computer science as it is not necessary and can be inferred if all of the other three majors are 0).

Figure 11.7 shows us an example:



Major (k=4)	Engineering	Business	Literature
Computer Science	0	0	0
Engineering	1	0	0
Business	0	1	0
Literature	0	0	1
Business	0	1	0
Engineering	1	0	0

Figure 11.7 – Creating dummy variables for a single feature involves creating a new binary feature for each option except for one, which can be inferred by having all 0s in the rest of the features

Note that the first row has a 0 in all of the columns, which means that this person did not major in engineering, did not major in business, and did not major in literature. The second person has a single 1 in the **Engineering** column as that is the major they studied.

We are going to need to make some dummy variables using pandas as we use scikit-learn's built-in decision tree function in order to build a decision tree:

```
# read in the data
titanic = pd.read_csv('short_titanic.csv')
# encode female as 0 and male as 1
titanic['Sex'] = titanic.Sex.map({'female':0, 'male':1})
# fill in the missing values for age with the median age
titanic.Age.fillna(titanic.Age.median(), inplace=True)
# create a DataFrame of dummy variables for Embarked
embarked_dummies = pd.get_dummies(titanic.Embarked, prefix='Embarked')
embarked_dummies.drop(embarked_dummies.columns[0], axis=1, inplace=True)
# concatenate the original DataFrame and the dummy DataFrame
titanic = pd.concat([titanic, embarked_dummies], axis=1)
# define X and y
feature_cols = ['Pclass', 'Sex', 'Age', 'Embarked_Q', 'Embarked_S']
X = titanic[feature_cols]
y = titanic.Survived
X.head()
```

Figure 11.8 shows what our dataset looks like after our preceding code block. Note that we are going to use class, sex, age, and dummy variables for city embarked as our features:

	Pclass	Sex	Age	Embarked_Q	Embarked_S
0	3	1	22.0	0.0	1.0
1	1	0	38.0	0.0	0.0
2	3	0	26.0	0.0	1.0
3	1	0	35.0	0.0	1.0
4	3	1	35.0	0.0	1.0

Figure 11.8 – Our Titanic dataset after creating dummy variables for Embarked

Now, we can fit our decision tree classifier:

```
# fit a classification tree with max_depth=3 on all data from sklearn.tree import
DecisionTreeClassifier treeclf = DecisionTreeClassifier(max_depth=3, random_state=1)
treeclf.fit(X, y)
```

`max_depth` is a hyperparameter that limits the depth of our tree. It means that, for any data point, our tree is only able to ask up to three questions and create three splits. We can output our tree into a visual format, and we will obtain the result seen in Figure 11.9:

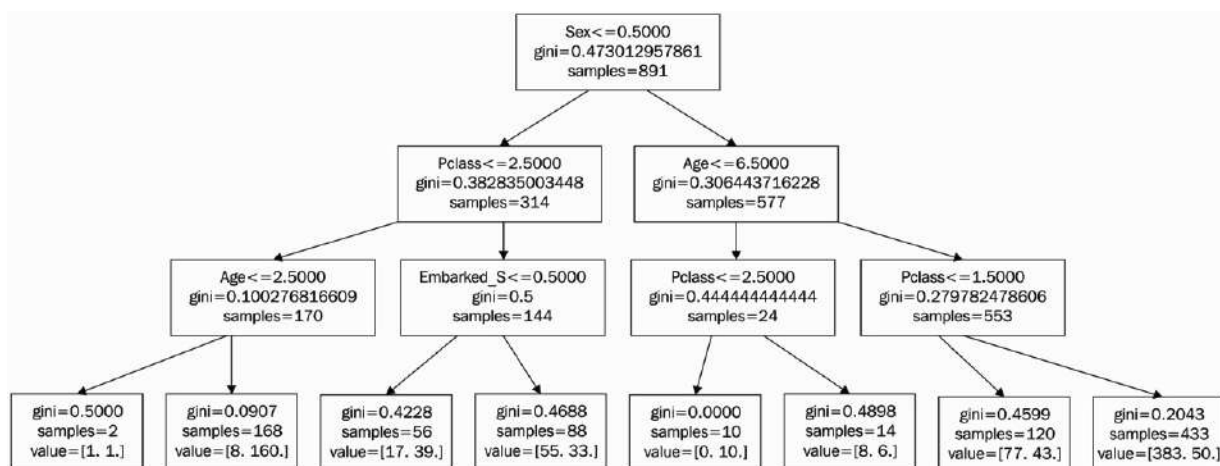


Figure 11.9 – The decision tree produced with scikit-learn with the Gini coefficient calculated at each node

We can notice a few things:

- **Sex** is the first split, meaning that sex is the most important determining factor of whether or not a person survived the crash
- **Embarked_Q** was never used in any split

For either classification or regression trees, we can also do something very interesting with decision trees, which is that we can output a number that represents each feature's importance in the prediction of our data points (shown in *Figure 11.10*):

```
# compute the feature importances pd.DataFrame({'feature':feature_cols,  
'importance':treeclf.feature_importances_})
```

	feature	importance
0	Pclass	0.242664
1	Sex	0.655584
2	Age	0.064494
3	Embarked_Q	0.000000
4	Embarked_S	0.037258

Figure 11.10 – Features that contributed most to the change in the Gini coefficient displayed as percentages adding up to 1; it's no coincidence that our highest value (Sex) is also our first split

The importance scores are an average Gini index difference for each variable, with higher values corresponding to higher importance to the prediction. We can use this information to select fewer features in the future. For example, both of the embarked variables are very low in comparison to the rest of the features, so we may be able to say that they are not important in our prediction of life or death.

As we transition from the structured realm of SL, where the outcomes are known and the model learns from labeled data, we venture into the domain of UL. Recall that UL algorithms uncover hidden patterns and intrinsic structures within data that isn't explicitly labeled. In the upcoming section, we will explore how unsupervised techniques can discern underlying relationships in data and provide deeper

insights without the guidance of a predefined outcome, and how they can complement the predictive models we've discussed so far.

Diving deep into UL

It's time to see some examples of UL, given that we've spent some time on *SL algorithms*.

When to use UL

There are many times when UL can be appropriate. Some very common examples include the following:

- There is no clear response variable. There is nothing that we are explicitly trying to predict or correlate to other variables.
- To extract structure from data where no apparent structure or patterns exist (can be an SL problem).
- When an unsupervised concept called **feature extraction** is used. Feature extraction is the process of creating new features from existing ones. These new features can be even stronger than the original features.

The first tends to be the most common reason that data scientists choose to use UL. This case arises frequently when we are working with data and we are not explicitly trying to predict any of the columns, and we merely wish to find patterns of similar (and dissimilar) groups of points. The second option comes into play even if we are explicitly attempting to use a supervised model to predict a response variable.

Sometimes, simple **exploratory data analysis (EDA)** might not produce any clear patterns in the data in the few dimensions that humans can imagine, whereas a machine might pick up on data points behaving similarly to each other in greater dimensions.

The third common reason to use UL is to extract new features from features that already exist. This process (lovingly called feature extraction) might produce features that can be used in a future supervised model or that can be used for presentation purposes (marketing or otherwise).

k-means clustering

k-means clustering is our first example of an **unsupervised ML (UML)** model. Remember – this means that we are not making predictions. We are trying instead to extract structure from seemingly unstructured data.

Clustering is a family of UML models that attempt to group data points into clusters with centroids. The concept of **similarity** is central to the definition of a cluster, and therefore to cluster analysis. In general, greater similarity between points leads to better clustering. In most cases, we turn data into points in n -dimensional space and use the distance between these points as a form of similarity. The

centroid of the cluster is then usually the average of each dimension (column) for each data point in each cluster. So, for example, the centroid of the red cluster is the result of taking the average value of each column of each red data point.

The purpose of cluster analysis is to enhance our understanding of a dataset by dividing the data into groups. Clustering provides a layer of abstraction from individual data points. The goal is to extract and enhance the natural structure of the data. There are many kinds of classification procedures. For our class, we will be focusing on k-means clustering, which is one of the most popular clustering algorithms.

k-means is an iterative method that partitions a dataset into k clusters. It works in four steps:

1. Choose k initial centroids (note that k is an input).
2. For each point, assign the point to the nearest centroid.
3. Recalculate the centroid positions.
4. Repeat *steps 2 and 3* until the stopping criteria are met.

An illustrative example of clustering

Imagine that we have data points in a two-dimensional space, as seen in *Figure 11.11*:

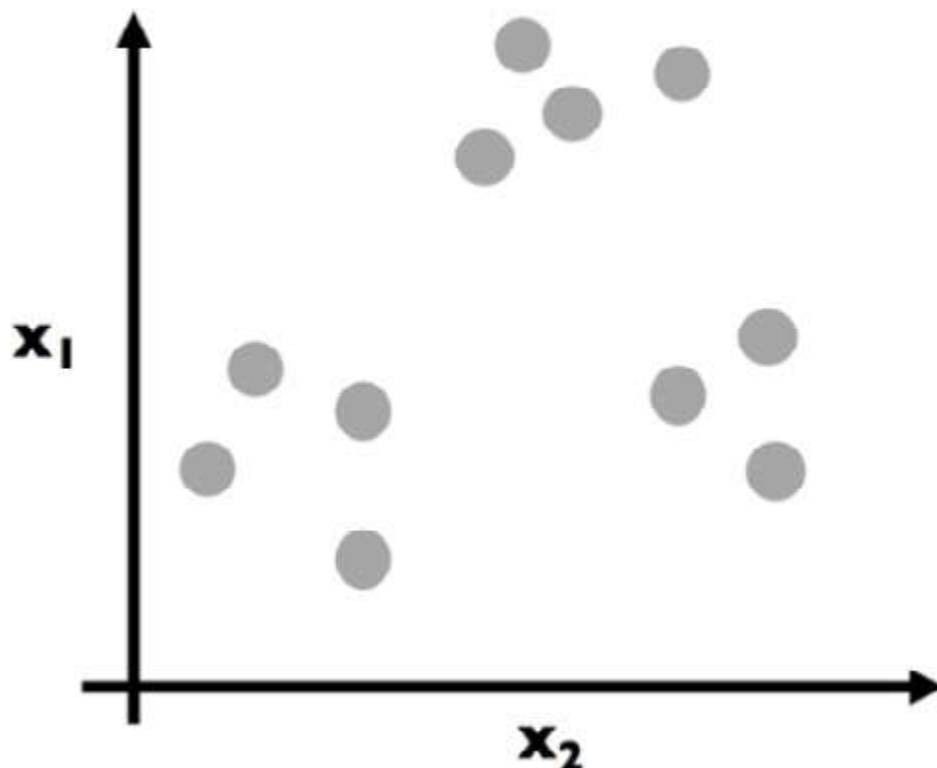


Figure 11.11 – A mock dataset to be clustered

Each dot is colored gray to assume no prior grouping before applying the k-means algorithm. The goal here is to eventually color in each dot and create groupings (clusters), as illustrated in *Figure 11.12*:

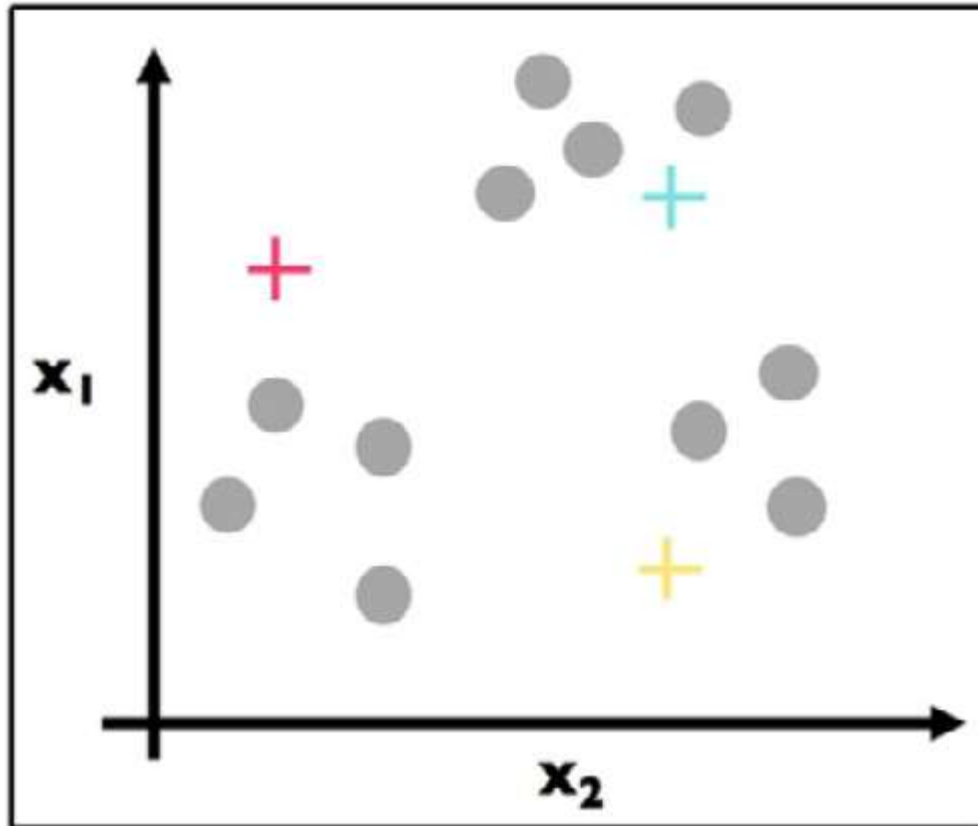


Figure 11.12 – Step 1: k-means clustering begins by placing random centroids

We have (randomly) chosen *three centroids* (red, blue, and yellow).

IMPORTANT NOTE

Most k-means algorithms place random initial centroids, but there exist other pre-computed methods to place initial centroids. For now, random is fine.

Step 2 has been applied in *Figure 11.13*. For each data point, we found the most similar centroid (closest):

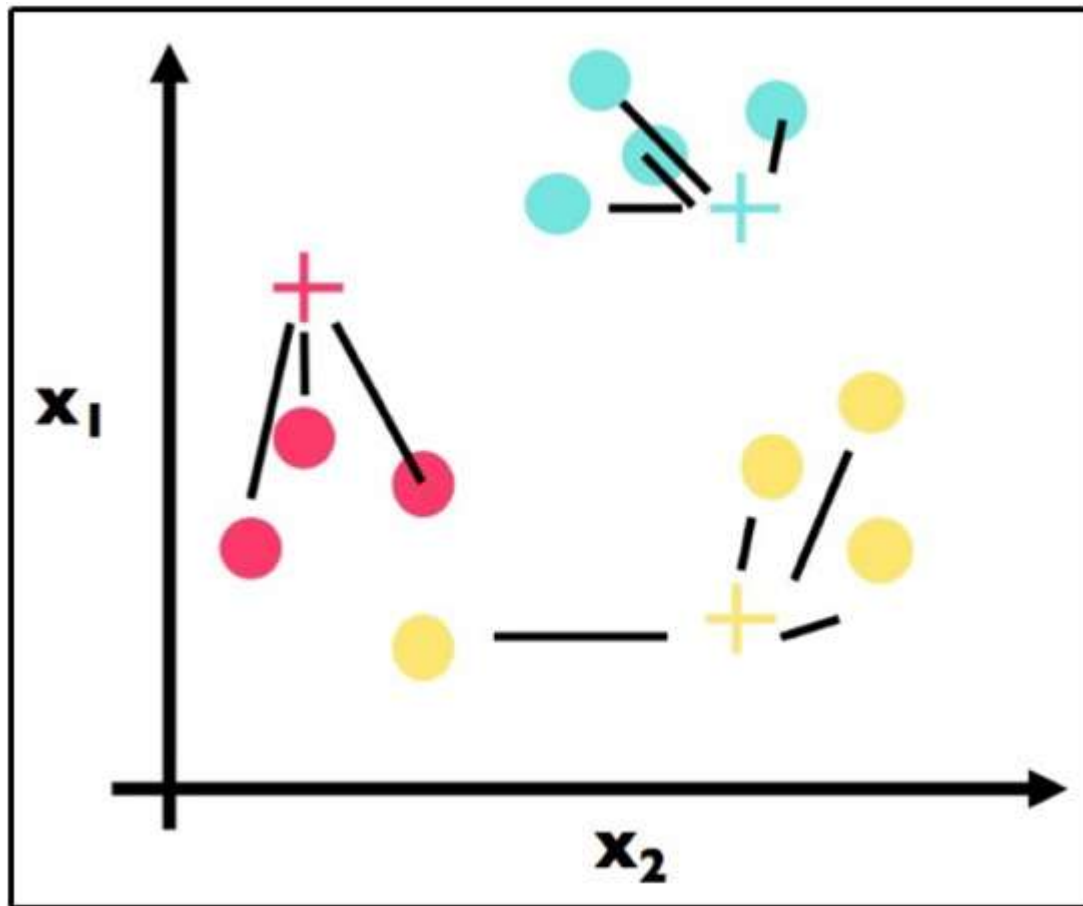


Figure 11.13 – Step 2: For each point, assign the point to the nearest centroid

We then apply *step 3* in *Figure 11.14*:

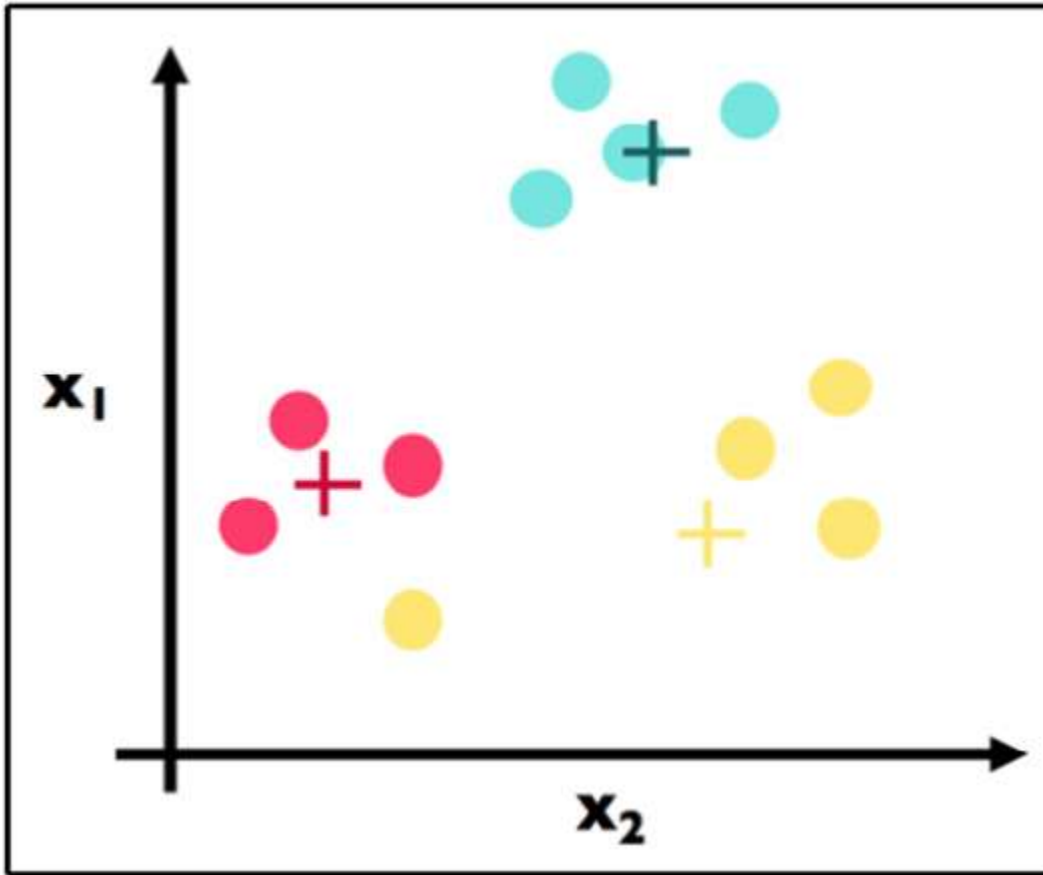


Figure 11.14 – Step 3: Recalculate the centroid positions

This is *step 3* and the crux of k-means. Note that we have physically moved the centroids to be the actual center of each cluster. We have, for each color, computed the average point and made that point the new centroid. For example, suppose the three red data points had the following coordinates: $(1, 3)$, $(2, 5)$, and $(3, 4)$. The *center (red cross)* would be calculated as follows:

```
# centroid calculation import numpy as np
red_point1 = np.array([1, 3]) red_point2 = np.array([2, 5]) red_point3 = np.array([3,
4])
red_center = (red_point1 + red_point2 + red_point3) / 3.
red_center
# array([ 2., 4.])
```

That is, the $(2, 4)$ point would be the coordinates of the preceding red cross.

We continue with our algorithm by repeating *step 2*. Here is the first part where we find the closest center for each point. Note a big change – the point in the bottom left used to be a yellow point but has changed to be a red cluster point because the yellow cluster moved closer to its yellow constituents:

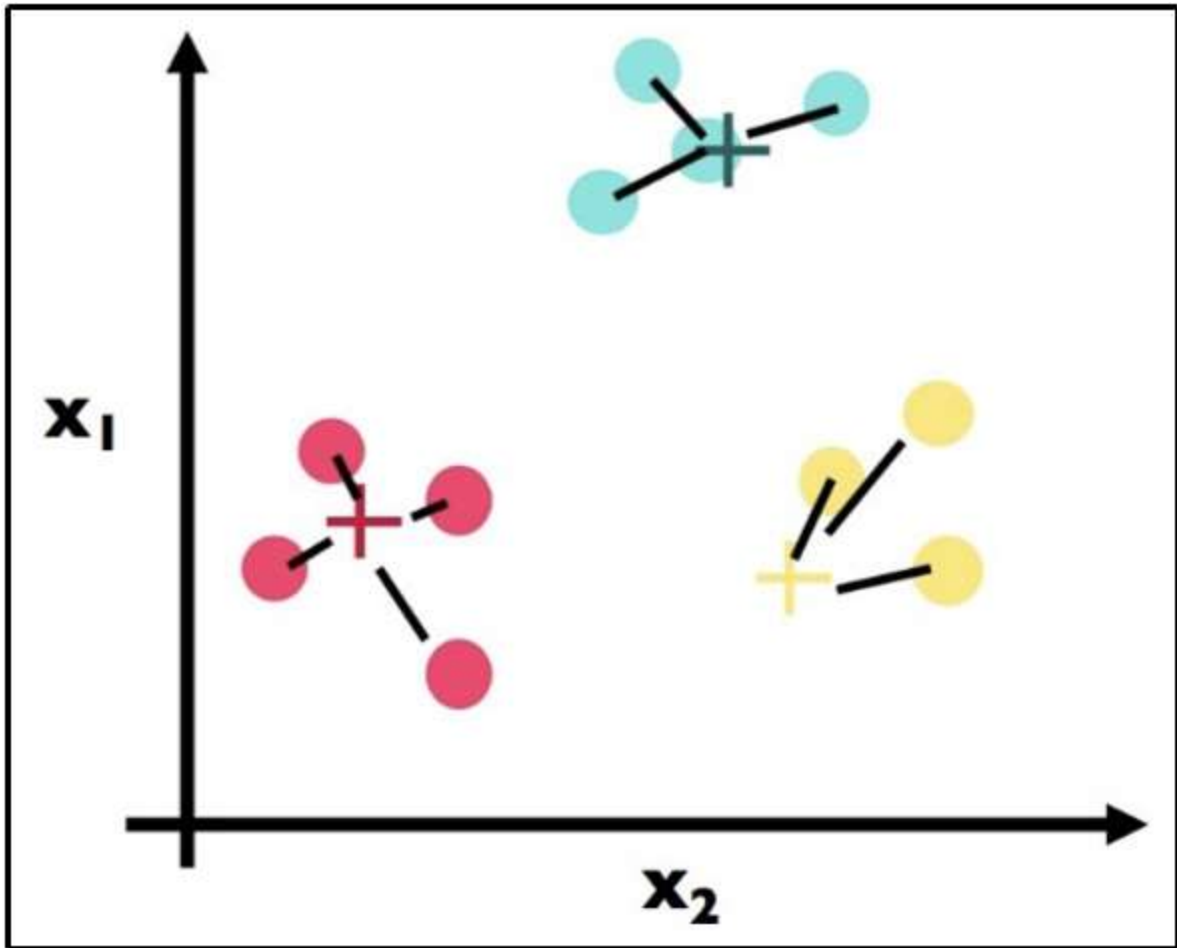


Figure 11.15 – Repeating step 2; note the data point on the lower left was yellow in the previous step and is now red
If we follow *step 3* again, we get the result shown in *Figure 11.16*:

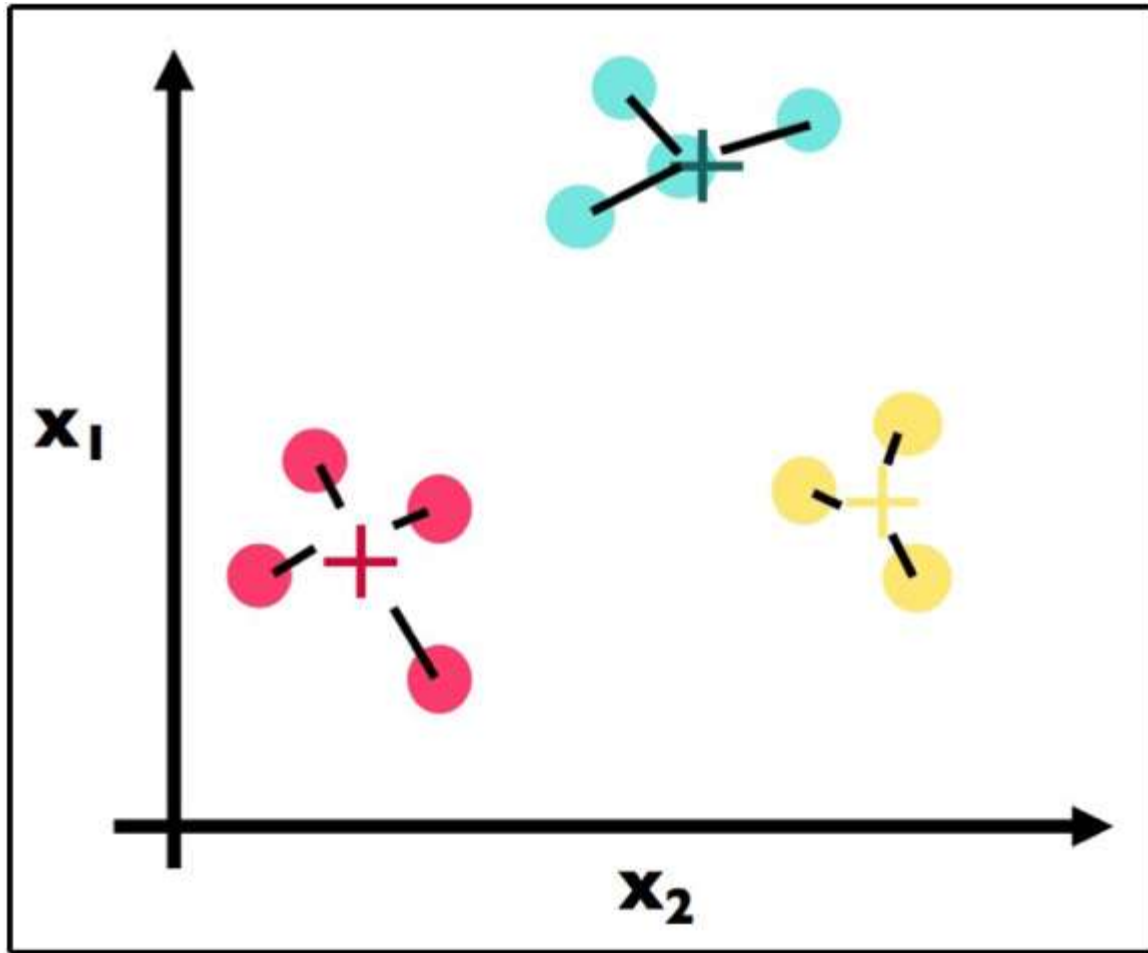


Figure 11.16 – Step 3 again

Here, we recalculate once more the centroids for each cluster (*step 3*). Note that the blue center did not move at all, while the yellow and red centers both moved.

Because we have reached a **stopping criterion** (clusters do not move if we repeat *steps 2* and *3*), we finalize our algorithm and we have our three clusters, which is the final result of the k-means algorithm.

An illustrative example – beer!

Let's run a cluster analysis on a new dataset outlining different beers with different characteristics. We know that there are many types of beer, but I wonder if we could possibly group beers into different categories based on different quantitative features.

Let's try! Let's import a dataset of just a few types of beer and visualize a few rows in *Figure 11.17*:

```
# import the beer dataset url = '../data/beer.txt'
beer = pd.read_csv(url, sep=' ')
beer.head()
```


	name	calories	sodium	alcohol	cost
0	Budweiser	144	15	4.7	0.43
1	Schlitz	151	19	4.9	0.43
2	Lowenbrau	157	15	0.9	0.48
3	Kronenbourg	170	7	5.2	0.73
4	Heineken	152	11	5.0	0.77

Figure 11.17 – The first five rows of our beer dataset

Our dataset has 20 beers with 5 columns: `name`, `calories`, `sodium`, `alcohol`, and `cost`. In clustering (as with almost all ML models), we like quantitative features, so we will ignore the name of the beer in our clustering:

```
# define X
X = beer.drop('name', axis=1)
```

Now, we will perform k-means clustering using `scikit-learn`:

```
# K-means with 3 clusters
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3, random_state=1) km.fit(X)
```

Our k-means algorithm has run the algorithm on our data points and come up with three clusters:

```
# save the cluster labels and sort by cluster beer['cluster'] = km.labels_
```

We can take a look at the center of each cluster by using `groupby` and `mean` statements (visualized in *Figure 11.18*):

```
# calculate the mean of each feature for each cluster beer.groupby('cluster').mean()
```

	calories	sodium	alcohol	cost
cluster				
0	150.00	17.0	4.521429	0.520714
1	102.75	10.0	4.075000	0.440000
2	70.00	10.5	2.600000	0.420000

Figure 11.18 – Our found clusters for the beer dataset with k=3

On inspection, we can see that *cluster 0* has, on average, a higher calorie, sodium, and alcohol content and costs more. These might be considered heavier beers. *Cluster 2* has on average a very low alcohol content and very few calories. These are probably light beers. *Cluster 1* is somewhere in the middle.

Let's use Python to make a graph to see this in more detail, as seen in *Figure 11.19*:

```
import matplotlib.pyplot as plt
%matplotlib inline
# save the DataFrame of cluster centers centers = beer.groupby('cluster').mean() #
create a "colors" array for plotting
colors = np.array(['red', 'green', 'blue', 'yellow'])
# scatter plot of calories versus alcohol, colored by cluster (0=red, 1=green,
2=blue)
plt.scatter(beer.calories, beer.alcohol, c=colors[list(beer.cluster)], s=50)
# cluster centers, marked by "+"
plt.scatter(centers.calories, centers.alcohol, linewidths=3, marker='+', s=300,
c='black')
# add labels plt.xlabel('calories') plt.ylabel('alcohol')
```

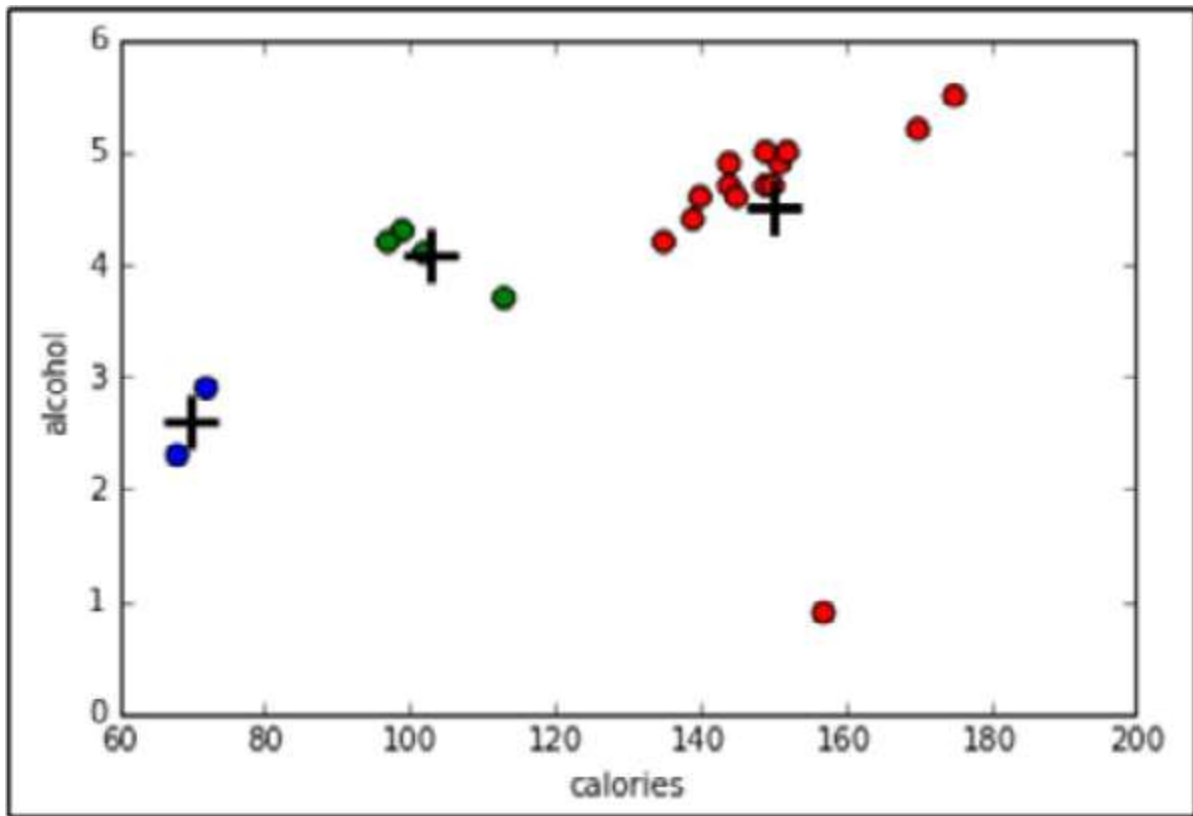


Figure 11.19 – Our cluster analysis visualized using two dimensions of our dataset

Choosing an optimal number for K and cluster validation

A big part of k-means clustering is knowing the optimal number of clusters. If we knew this number ahead of time, then that might defeat the purpose of even using UL. So, we need a way to evaluate the output of our cluster analysis. The problem here is that, because we are not performing any kind of prediction, we cannot gauge how right the algorithm is at predictions. Metrics such as accuracy and RMSE go right out of the window. Luckily, we do have a pretty useful metric to help optimize our cluster analyses, called the Silhouette Coefficient.

The Silhouette Coefficient

The **Silhouette Coefficient** is a common metric for evaluating clustering performance in situations when the true cluster assignments are not known. The Silhouette Coefficient is a measure used to assess the quality of clusters created by a clustering algorithm. It quantifies how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The Silhouette Coefficient for a single data point is calculated using the following formula:

$$SC = \frac{b - a}{\max(a, b)}$$

Here, the following applies:

- a is the mean distance between a sample and all other points in the same class or cluster. It represents the cohesion of the cluster.
- b is the mean distance between a sample and all other points in the next nearest cluster. It represents the separation from the nearest cluster that the sample is not a part of.

It ranges from -1 (*worst*) to 1 (*best*). A global score is calculated by taking the mean score for all observations. The Silhouette Coefficient is particularly useful for determining the effectiveness of a clustering algorithm because it takes into account both the compactness of the clusters and the separation between them. In general, a Silhouette Coefficient of 1 is preferred, while a score of -1 is not preferable:

```
# calculate Silhouette Coefficient for K=3 from sklearn import metrics
metrics.silhouette_score(X, km.labels_)
```

The output is as follows:

```
0.67317750464557957
```

Let's try calculating the coefficient for multiple values of k to find the best value:

```
# center and scale the data
from sklearn.preprocessing import StandardScaler scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# calculate SC for K=2 through K=19 k_range = range(2, 20)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1) km.fit(X_scaled)
    scores.append(metrics.silhouette_score(X, km.labels_))
# plot the results plt.plot(k_range, scores) plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient') plt.grid(True)
```

So, it looks like our optimal number of beer clusters is 4! This means that our k-means algorithm has determined that there seem to be four distinct types of beer.

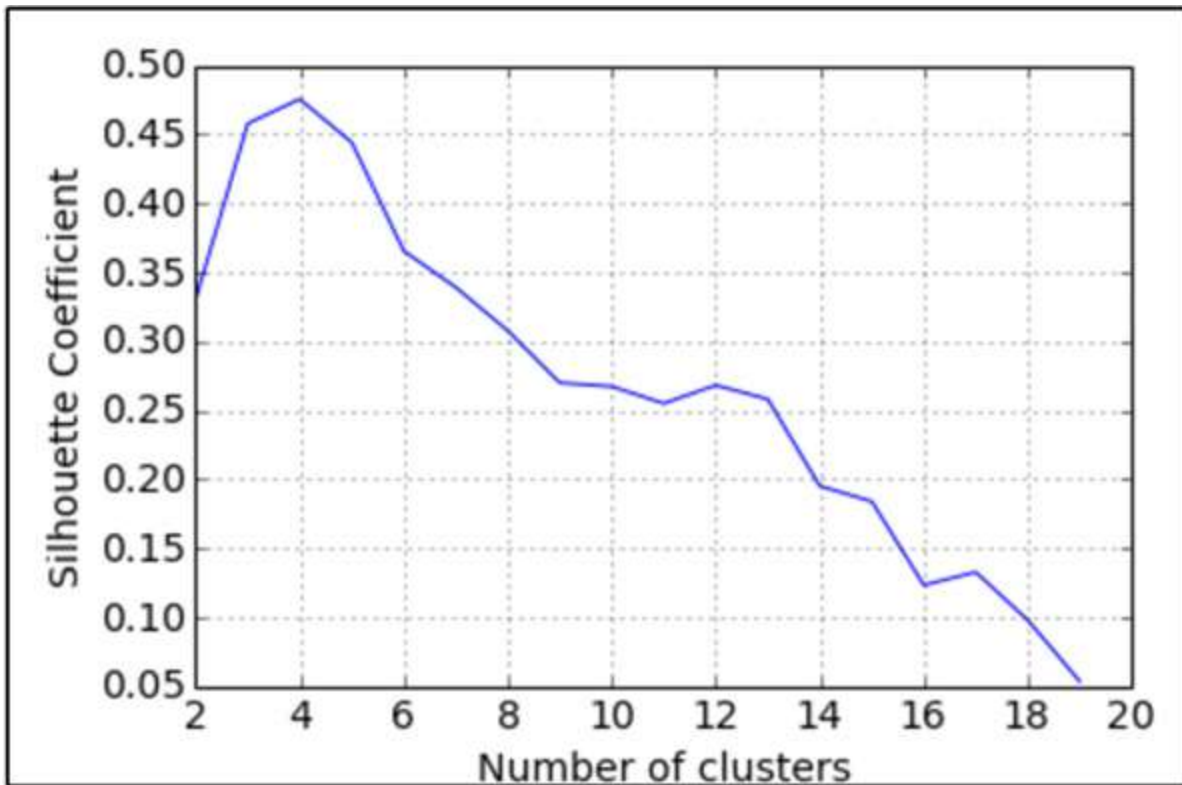


Figure 11.20 – The Silhouette Coefficient for a varying number of clusters

k-means is a popular algorithm because of its computational efficiency and simple and intuitive nature. k-means, however, is highly scale-dependent and is not suitable for data with widely varying shapes and densities. There are ways to combat this issue by scaling data using `scikit-learn`'s `StandardScaler`:

```
# center and scale the data
from sklearn.preprocessing import StandardScaler scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# K-means with 3 clusters on scaled data km = KMeans(n_clusters=3, random_state=1)
km.fit(X_scaled)
```

Easy!

Now, let's take a look at the third option in our reasons for using unsupervised methods: *feature extraction*.

Feature extraction and PCA

A common problem when working with data, particularly when it comes to ML, is having an overwhelming number of columns and not enough rows to handle such a quantity of columns.

A great example of this is when we were looking at the *send cash now* example in our naïve Bayes example earlier. Remember we had literally 0 instances of texts with that exact phrase? In that case, we turned to a naïve assumption that allowed us to extrapolate a probability for both of our categories.

The reason we had this problem in the first place is because of something called the **curse of dimensionality (COD)**. The COD basically says that as we introduce new feature columns, we need exponentially more rows (data points) to consider the increased number of possibilities.

Consider an example where we attempt to use a learning model that utilizes the distance between points on a corpus of text that has 4,086 pieces of text and that the whole thing has been count-vectorized using `scikit-learn`. Now, let's do an experiment. I will first consider a single word as the only dimension of our text. Then, I will count how many pieces of text are within 1 unit of each other. For example, if 2 sentences both contain that word, they would be 0 units away and, similarly, if neither of them contains the word, they would be 0 units away from one another:

```
d = 1
# Let's look for points within 1 unit of one another
X_first_word = X.iloc[:, :1]
# Only looking at the first column, but ALL of the rows
from sklearn.neighbors import NearestNeighbors
# this module will calculate for us distances between each point
neigh = NearestNeighbors(n_neighbors=4086)
neigh.fit(X_first_word)
# tell the module to calculate each distance between each point
A = neigh.kneighbors_graph(X_first_word, mode='distance').todense() # This matrix
holds all distances (over 16 million of them)
num_points_within_d = (A < d).sum()
# Count the number of pairs of points within 1 unit of distance, 16,258,504
```

IMPORTANT NOTE

*Note that we have 16,695,396 (4086*4086) distances to scan over.*

So, 16.2 million pairs of texts are within a single unit of distance. Now, let's try again with the first two words:

```
X_first_two_words = X.iloc[:, :2]
neigh = NearestNeighbors(n_neighbors=4086) neigh.fit(X_first_two_words)
A = neigh.kneighbors_graph(X_first_two_words, mode='distance').todense()
num_points_within_d = (A < d).sum()
# num_points_within_d is now 16,161,970
```

By considering a single new column, we lost about 100,000 pairs of points that were within a single unit of distance. This is because we are adding space in between them for every dimension that we add. Let's take this test a step further and calculate this number for the first 100 words and then plot the results:

```
num_columns = range(1, 100)
# Looking at the first 100 columns points = []
# We will be collecting the number of points within 1 unit for a graph
neigh = NearestNeighbors(n_neighbors=X.shape[0])
for subset in num_columns:
    X_subset = X.iloc[:, :subset]
    # look at the first column, then first two columns, then first three columns, etc
    neigh.fit(X_subset)
    A = neigh.kneighbors_graph(X_subset, mode='distance').todense() num_points_within_d =
    (A < d).sum()
    # calculate the number of points within 1 unit points.append(num_points_within_d)
```

Now, let's plot the number of points within 1 unit versus the number of dimensions we consider in *Figure 11.21*:

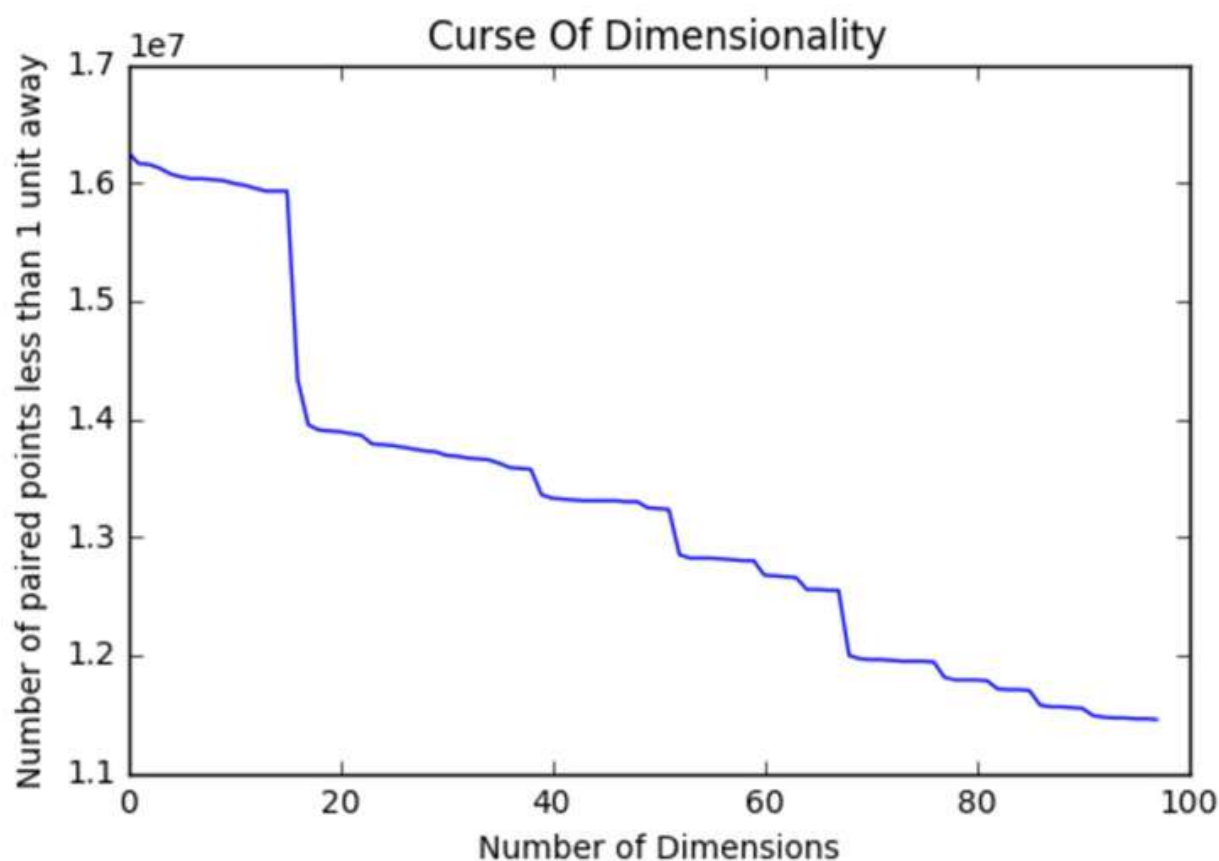


Figure 11.21 – The COD says that as we increase the number of feature columns in our dataset, data points become further away from each other due to the increase in high-dimensional space

Put another way, the COD states that as we increase the number of feature columns, we need exponentially more data to maintain the same level of model performance. This is because, in high-dimensional spaces, even the nearest neighbors can be very far away from a given data point, making it difficult to create good predictions. High dimensionality also increases the risk of overfitting as the model may start to fit to noise in the data rather than the actual signal.

Moreover, with more dimensions, the volume of the space increases so rapidly that the available data becomes sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a reliable result, the amount of data needed to support the analysis often grows exponentially with the dimensionality.

We can see clearly that the number of points within a single unit of one another goes down dramatically as we introduce more and more columns. And this is only the first 100 columns!

All of this space that we add in by considering new columns makes it harder for the finite amount of points we have to stay happily within range of each other. We would have to add more points in order to fill in this gap. And that, my friends, is why we should consider using dimension reduction.

The COD is solved by either adding more data points (which is not always possible) or implementing dimension reduction. **Dimension reduction** is simply the act of reducing the number of columns in our dataset and not the number of rows. There are two ways of implementing dimension reduction:

- **Feature selection:** This is the act of creating a subset of our column features and only using the best features
- **Feature extraction:** This is the act of mathematically transforming our feature set into a new extracted coordinate system

We are familiar with feature selection as the process of saying the `Embarked_Q` column is not helping our decision tree. Let's get rid of it and see how it performs. It is literally when we (or the machine) make the decision to ignore certain columns.

Feature extraction is a bit trickier.

In *feature extraction*, we are using usually fairly complicated mathematical formulas in order to obtain new super columns that are usually better than any single original column.

Our primary model for doing so is called **PCA**. PCA will extract a set number of super columns in order to represent our original data with much fewer columns. Let's take a concrete example. Previously, I mentioned some text with 4,086 rows and over 18,000 columns. That dataset is actually a set of *Yelp* online reviews:

```
url = '../data/yelp.csv'
yelp = pd.read_csv(url, encoding='unicode-escape')
# create a new DataFrame that only contains the 5-star and 1-star reviews
yelp_best_worst = yelp[(yelp.stars==5) | (yelp.stars==1)]
# define X and y
X = yelp_best_worst.text
y = yelp_best_worst.stars == 5
```

Our goal is to predict whether or not a person gave a 5- or 1-star review based on the words they used in the review. Let's set a baseline with logistic regression and see how well we can predict this binary category:

```
from sklearn.linear_model import LogisticRegression lr = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=100) # Make
our training and testing sets
vect = CountVectorizer(stop_words='english')
# Count the number of words but remove stop words like a, an, the, you, etc
X_train_dtm = vect.fit_transform(X_train) X_test_dtm = vect.transform(X_test)
# transform our text into document term matrices
lr.fit(X_train_dtm, y_train) # fit to our training set
lr.score(X_test_dtm, y_test) # score on our testing set
```

The output is as follows:

```
0.91193737
```


So, by utilizing all of the words in our corpus, our model seems to have over a 91% accuracy. Not bad!

Let's try only using the top 100 used words:

```
vect = CountVectorizer(stop_words='english', max_features=100) # Only use the 100
most used words
X_train_dtm = vect.fit_transform(X_train) X_test_dtm = vect.transform(X_test) print(
X_test_dtm.shape) # (1022, 100)
lr.fit(X_train_dtm, y_train) lr.score(X_test_dtm, y_test)
```

The output is as follows:

```
0.8816
```

Note how our training and testing matrices have 100 columns. This is because I told our vectorizer to only look at the top 100 words. See also that our performance took a hit and is now down to 88% accuracy. This makes sense because we are ignoring over 4,700 words in our corpus.

Now, let's take a different approach. Let's import a PCA module and tell it to make us 100 new super columns and see how that performs:

```
from sklearn import decomposition
# We will be creating 100 super columns
vect = CountVectorizer(stop_words='english') # Don't ignore any words
pca = decomposition.PCA(n_components=100) # instantiate a pca object
X_train_dtm = vect.fit_transform(X_train).todense()
# A dense matrix is required to pass into PCA, does not affect the overall message
X_train_dtm = pca.fit_transform(X_train_dtm)
X_test_dtm = vect.transform(X_test).todense() X_test_dtm = pca.transform(X_test_dtm)
print(X_test_dtm.shape) # (1022, 100) lr.fit(X_train_dtm, y_train)
lr.score(X_test_dtm, y_test)
```

The output is as follows:

```
.89628
```

Not only do our matrices still have 100 columns, but these columns are no longer words in our corpus. They are complex transformations of columns and are 100 new columns. Also, note that using 100 of these new columns gives us a better predictive performance than using the 100 top words!

Feature extraction is a great way to use mathematical formulas to extract brand-new columns that generally perform better than just selecting the best ones beforehand.

But how do we visualize these new super columns? Well, I can think of no better way than to look at an example using image analysis. Specifically, let's make facial recognition software. OK? OK. Let's begin by importing some faces given to us by **scikit-learn**:

```
from sklearn.datasets import fetch_lfw_people
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
# introspect the images arrays to find the shapes (for plotting) n_samples, h, w =
lfw_people.images.shape
# for machine learning we use the 2 data directly (as relative pixel # positions info
is ignored by this model)
X = lfw_people.data
y = lfw_people.target n_features = X.shape[1] X.shape (1288, 1850)
```

We have gathered 1,288 images of people's faces, and each one has 1,850 features (pixels) that identify that person. Here's the code we used – an example of one of our faces can be seen in *Figure 11.22*:

```
plt.imshow(X[100].reshape((h, w)), cmap=plt.cm.gray) lfw_people.target_names[y[100]]  
'George W Bush'
```

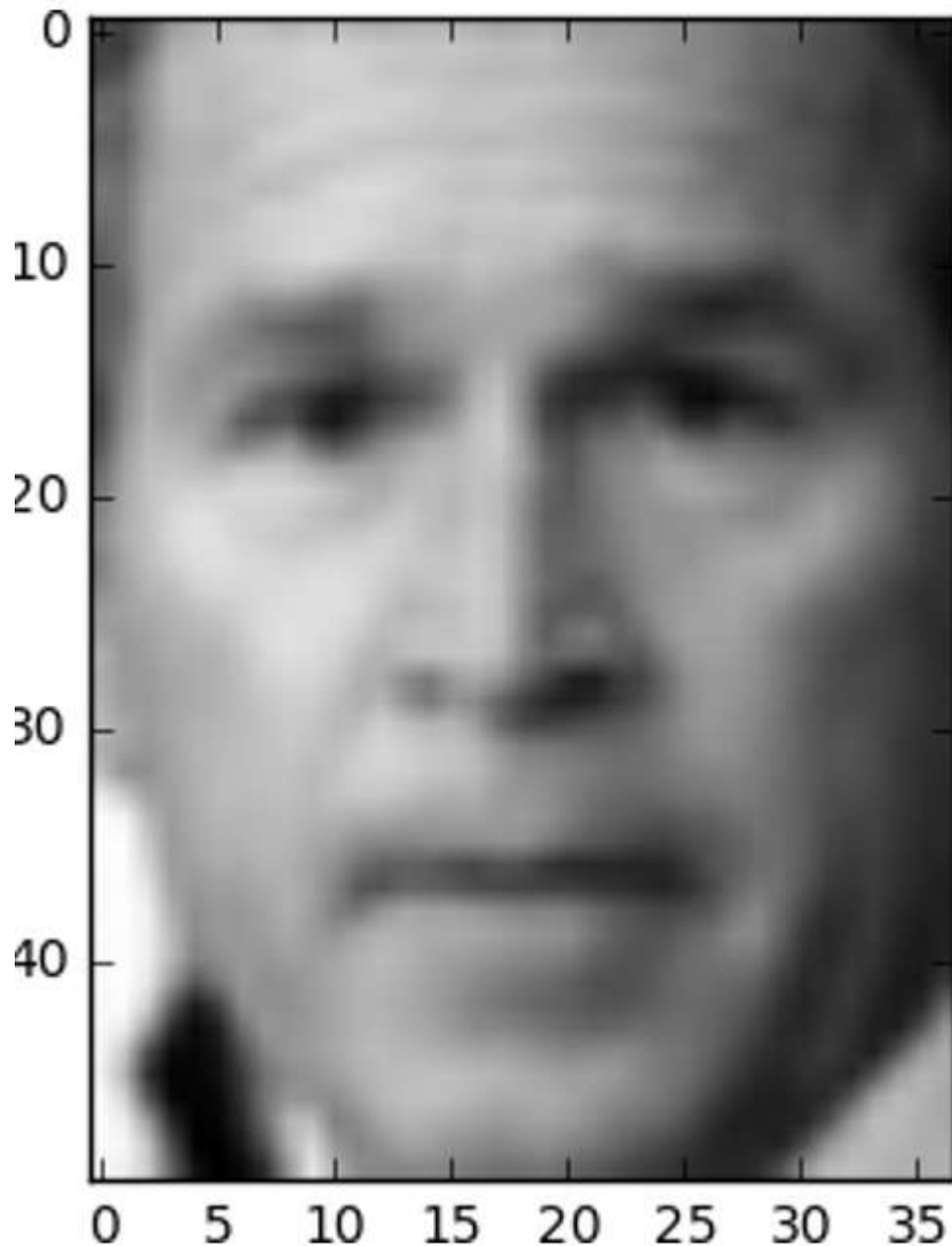


Figure 11.22 – A face from our dataset: George W. Bush

Great! To get a glimpse at the type of dataset we are looking at, let's look at a few overall metrics:

```
# the label to predict is the id of the person target_names = lfw_people.target_names  
n_classes = target_names.shape[0]
```

```

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes) Total dataset size:
---
n_samples: 1288
n_features: 1850
n_classes: 7

```

So, we have 1,288 images, 1,850 features, and 7 classes (people) to choose from. Our goal is to make a classifier that will assign the person's face a name based on the 1,850 pixels given to us.

Let's take a baseline and see how a logistic regression (a classifier that is based on linear regression) performs on our data without doing anything to our dataset.

I know we haven't formally introduced logistic regressions before, but they are a very lightweight classifier that works off of very similar assumptions as linear regressions from the last chapter. All we need to know for now is that it performs classification!

```

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from time import time # for timing our work
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.25,
random_state=1)
# get our training and test set
t0 = time()
# get the time now
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
# Predicting people's names on the test set
y_pred = logreg.predict(X_test)
print( accuracy_score(y_pred, y_test), "Accuracy") print( (time() - t0), "seconds" )

```

The output is as follows:

```

0.810559006211 Accuracy
6.31762504578 seconds

```

So, within 6.3 seconds, we were able to get 81% on our test set. Not too bad.

Now, let's try this with our decomposed faces:

```

# split into a training and testing set
from sklearn.cross_validation import train_test_split
# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled # dataset):
unsupervised feature extraction / dimensionality reduction
n_components = 75
print("Extracting the top %d eigenfaces from %d faces" % (n_components,
X_train.shape[0]))
pca = decomposition.PCA(n_components=n_components, whiten=True).fit(X_train)
# This whiten parameter speeds up the computation of our extracted columns
# Projecting the input data on the eigenfaces orthonormal basis
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

```

The preceding code is collecting 75 extracted columns from our 1,850 unprocessed columns. These are our super faces. Now, let's plug in our newly extracted columns into our logistic regression and compare:

```

t0 = time()
# Predicting people's names on the test set WITH PCA logreg.fit(X_train_pca, y_train)
y_pred = logreg.predict(X_test_pca)
print accuracy_score(y_pred, y_test), "Accuracy" print (time() - t0), "seconds"
0.82298136646 Accuracy
0.194181919098 seconds

```

Wow! Not only was this entire calculation about 30 times faster than the unprocessed images, but the predictive performance also got better! This shows us that PCA and feature extraction, in general, can help us all around when performing ML on complex datasets with many columns. By searching for these patterns in the dataset and extracting new feature columns, we can speed up and enhance our learning algorithms.

Let's look at one more interesting thing. I mentioned before that one of the purposes of this example was to examine and visualize our eigenfaces, as they are called: our super columns. I will not disappoint. Let's write some code that will show us our super columns as they would look to us humans:

```

def plot_gallery(images, titles, n_row=3, n_col=4): """Helper function to plot a
gallery of portraits""" plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
for i in range(n_row * n_col):
    plt.subplot(n_row, n_col, i + 1) plt.imshow(images[i], cmap=plt.cm.gray)
    plt.title(titles[i], size=12)
# plot the gallery of the most significative eigenfaces eigenfaces =
pca.components_.reshape((n_components, h, w))
eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles)
plt.show()

```

Warning: the faces in *Figure 11.23* are a bit creepy!

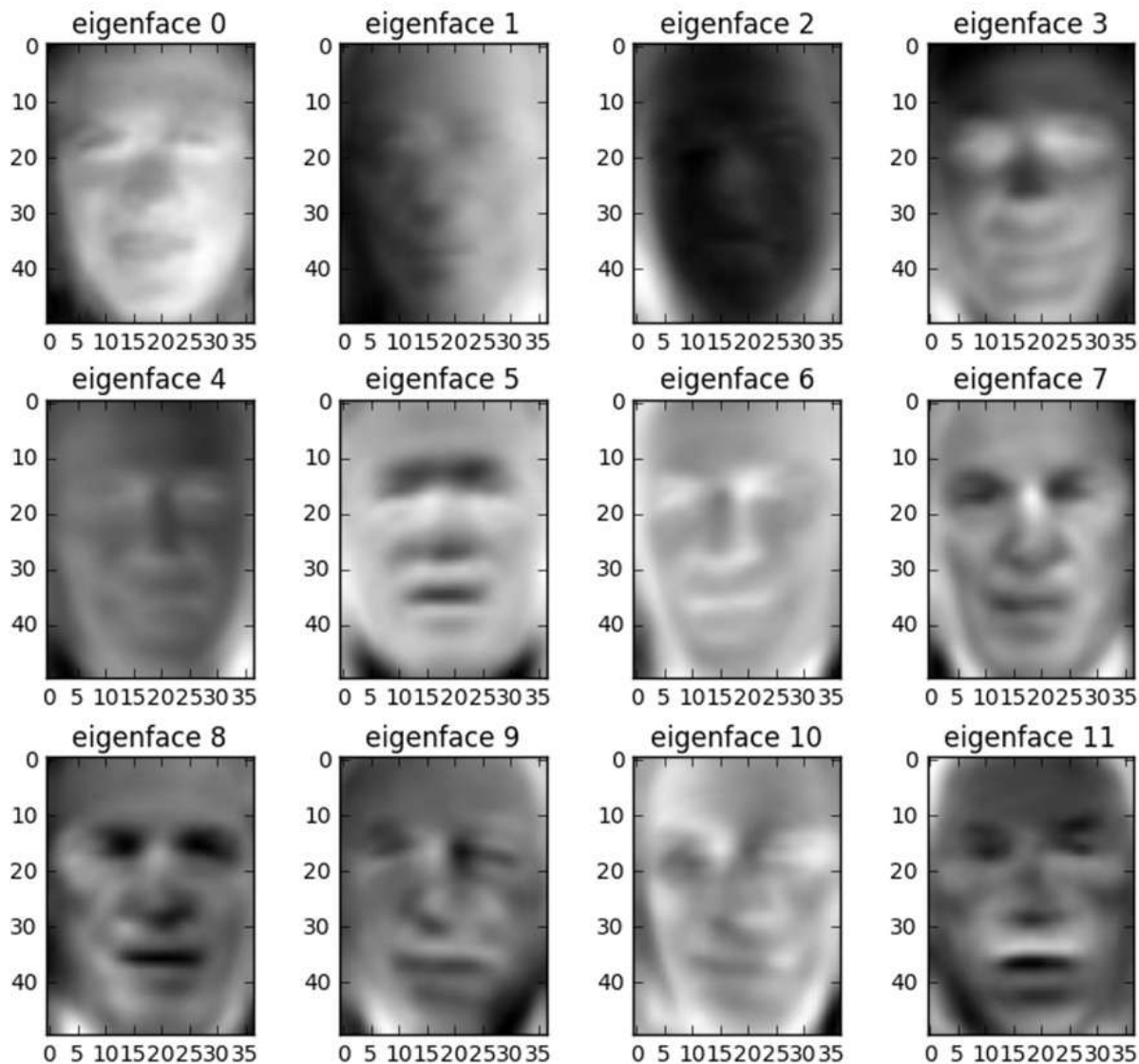


Figure 11.23 – Performing PCA on the pixels of our faces creates what is known as “eigenfaces” that represent features that our classifiers look for when trying to recognize faces

Wow! A haunting and yet beautiful representation of what the data believes to be the most important features of a face. As we move from the top left (first super column) to the bottom, it is actually somewhat easy to see what the image is trying to tell us. The first super column looks like a very general face structure with eyes and nose and a mouth. It is almost saying “I represent the basic qualities of a face that all faces must have.” Our second super column directly to its right seems to be telling us about shadows in the image. The next one might be telling us that skin tone plays a role in detecting who this is, which might be why the third face is much darker than the first two.

Using feature extraction UL methods such as PCA can give us a very deep look into our data and reveal to us what the data believes to be the most important features, not just what we believe them to be. Feature extraction is a great preprocessing tool that can speed up our future learning methods, make

them more powerful, and give us more insight into how the data believes it should be viewed. To sum up this section, we will list the pros and cons.

Here are some of the advantages of using feature extraction:

- Our models become much faster
- Our predictive performance can become better
- It can give us insight into the extracted features (eigenfaces)

And here are some of the disadvantages of using feature extraction:

- We lose some of the interpretability of our features as they are new mathematically derived columns, not our old ones
- We can lose predictive performance because we are losing information as we extract fewer columns

Let's move on to the summary next.

Summary

Our exploration into the world of ML has revealed a vast landscape that extends well beyond the foundational techniques of linear and logistic regression. We delved into decision trees, which provide intuitive insights into data through their hierarchical structure. Naïve Bayes classification offered us a probabilistic perspective, showing how to make predictions under the assumption of feature independence. We ventured into dimensionality reduction, encountering techniques such as feature extraction, which help overcome the COD and reduce computational complexity.

k-means clustering introduced us to the realm of UL, where we learned to find hidden patterns and groupings in data without pre-labeled outcomes. Across these methods, we've seen how ML can tackle a plethora of complex problems, from predicting categorical outcomes to uncovering latent structures in data.

Through practical examples, we've compared and contrasted SL, which relies on labeled data, with UL, which operates without explicit guidance on the output. This journey has equipped us with a deeper understanding of the various techniques and their appropriate applications within the broad and dynamic field of data science.

As we continue to harness the power of these algorithms, we are reminded of the importance of selecting the right model for the right task—a principle that remains central to the practice of effective data science.

Introduction to Transfer Learning and Pre-Trained Models

Just as one wouldn't try to reinvent the wheel, in the world of data science and **machine learning (ML)**, it's often more efficient to build upon existing knowledge. This is where the concepts of **transfer learning (TL)** and pre-trained models come into play, two incredibly important tools in a data scientist's repertoire.

TL is almost like a shortcut in ML. Instead of taking a model architecture that has never seen data before, such as a Logistic Regression model or a Random Forest model, imagine being able to take a model trained on one task and then repurposing it for a different, yet related task. That's TL in a nutshell – leveraging existing knowledge to learn new things more efficiently. It's a concept that echoes throughout many facets of life and is a key technique in data science.

Pre-trained models are off-the-shelf components, ready to be used right out of the box. They're ML models that have been trained on large datasets, capturing an immense amount of information about the task they were trained on. When it comes to tackling new tasks, these pre-trained models provide a substantial head start. The importance of TL and pre-trained models cannot be overstated in modern ML. They are fundamental to many cutting-edge applications in data science, from **computer vision (CV)** tasks such as image recognition to **natural language processing (NLP)** tasks such as **sentiment analysis (SA)**. By leveraging these techniques, we can achieve impressive results even with limited data or resources.

In the following sections, we will investigate these concepts, exploring their nuances, applications, and potential:

- Understanding pre-trained models
- Different types of TL
- TL with **Bidirectional Encoder Representations from Transformers (BERT)** and **Generative Pre-trained Transformer (GPT)**

We'll also walk through practical examples that highlight their power in real-world scenarios. By the end of this chapter, you'll have a solid understanding of TL and pre-trained models and be equipped to harness their potential in your own projects.

Understanding pre-trained models

Pre-trained models are like learning from the experience of others. These models have been trained on extensive datasets, learning patterns, and features that make them adept at their tasks. Think of it as if a model has been reading thousands of books on a subject, absorbing all that information. When we use a pre-trained model, we're leveraging all that prior knowledge.

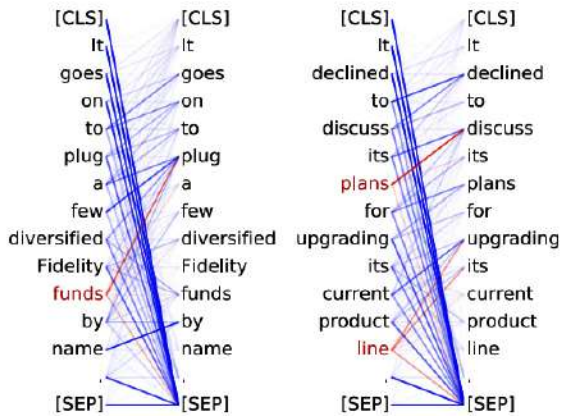
In general, pre-training steps are not necessarily “useful” to a human, but it is crucial to a model to simply learn about a domain and about a medium. Pre-training helps models learn how language works in general but not how to classify sentiments or detect an object.

Benefits of using pre-trained models

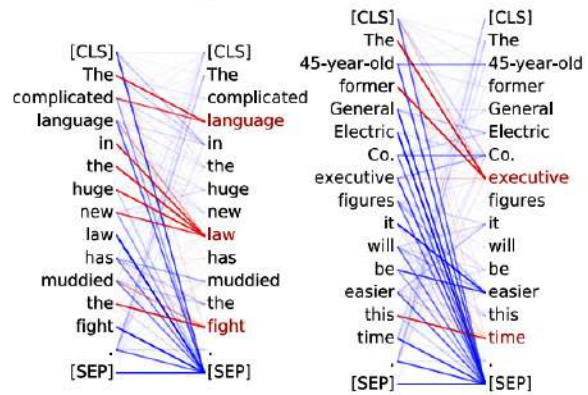
The benefits of using pre-trained models are numerous. For starters, they save us a lot of time. Training a model from scratch can be a time-consuming process, but using a pre-trained model gives us a head start. Furthermore, these models often lead to better performance, especially when our dataset is relatively small. The reason? Pre-trained models have seen much more data than we usually have at our disposal, and they've learned a lot from it.

Figure 12.1 shows the result of a study done on **large language models (LLMs)** such as BERT where one of the goals was to show that pre-training was leading to some obvious patterns in how BERT was recognizing basic grammatical constructs. The study visualized that models post pre-training were able to recognize what we would consider as obvious grammatical patterns, such as pronoun-antecedent relationships and direct object/verb relationships:

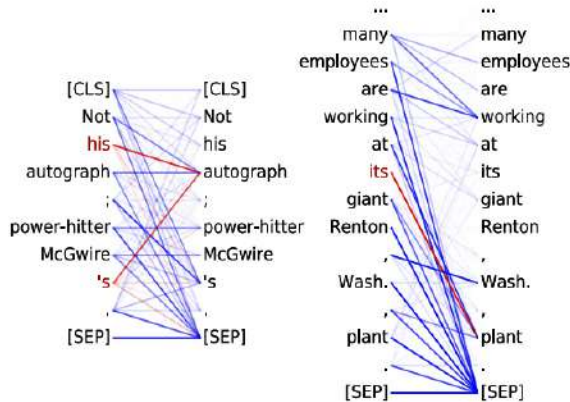
- **Direct objects** attend to their verbs
- 86.8% accuracy at the **dobj** relation



- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the **det** relation



- **Possessive pronouns** and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the **poss** relation



- **Passive auxiliary verbs** attend to the verb they modify
- 82.5% accuracy at the **auxpass** relation

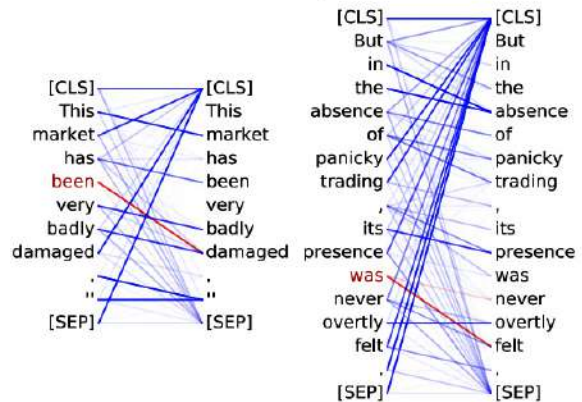


Figure 12.1 – A study visualizing how BERT's pre-training allowed it to pick up on common grammatical constructs without ever being told what they were

BERT, of course, is not the only model that undergoes pre-training, and this practice is not even limited to text-based models.

Commonly used pre-trained models

Pre-trained models come in all shapes and sizes, each tailored to different types of data and tasks. Let's talk about some of the most popular ones.

Image-based models

For tasks related to images, models such as the Vision Transformer (more on this one later in this chapter), models from the **Visual Geometry Group (VGG)** (as seen in *Figure 12.2*), and ResNet are

some common options to choose from. Models from these families have been trained on tens of thousands of images, learning to recognize everything from shapes and textures to complex objects. They're incredibly versatile and can be fine-tuned for a wide array of image-based tasks:

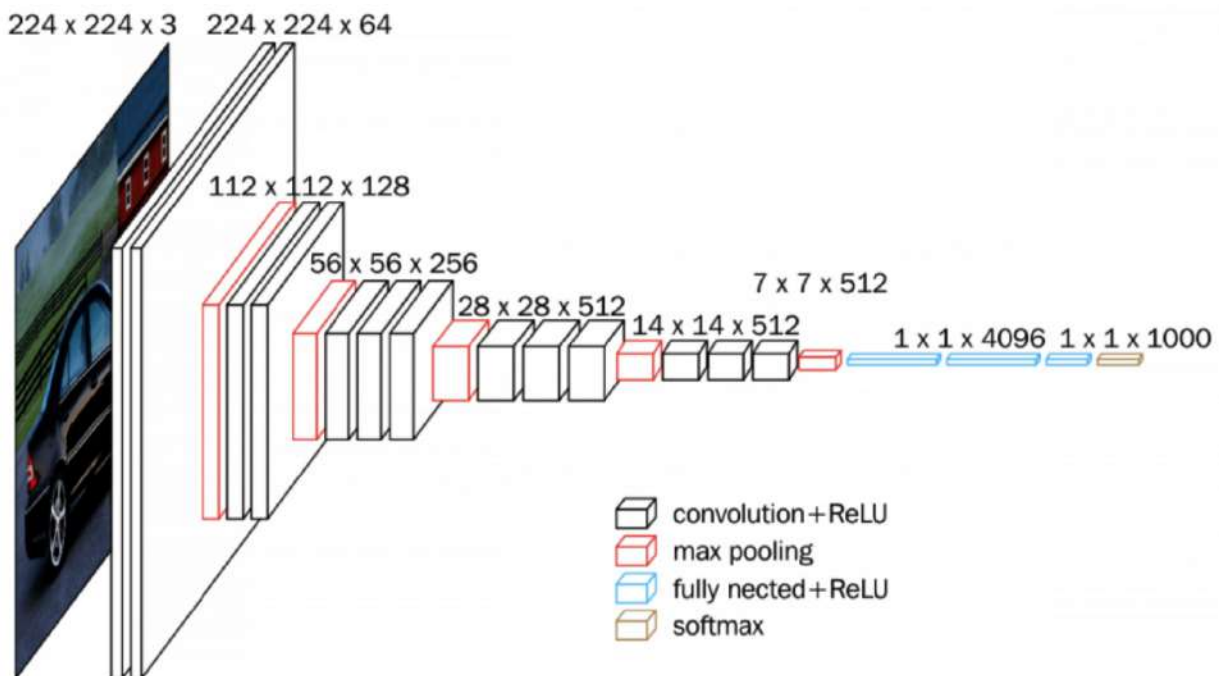


Figure 12.2 – The VGG16 model (from the VGG) is a convolutional neural network (CNN) that can be pre-trained on image data

Text-based models

When it comes to text, models such as BERT (*Figure 12.3*) and GPT are among the most used language models. They were first originally architected in 2018 (only 1 year after the primary Transformer architecture that both GPT and BERT are based on was even proposed or mentioned), and they've, as with their image counterparts, been trained on vast amounts of text data, learning the intricacies of human language. Whether it's understanding the sentiment behind a tweet or answering questions about a piece of text, these models are up to the task. As we move forward, we'll see how these pre-trained models can be combined with TL to tackle new tasks with impressive efficiency:

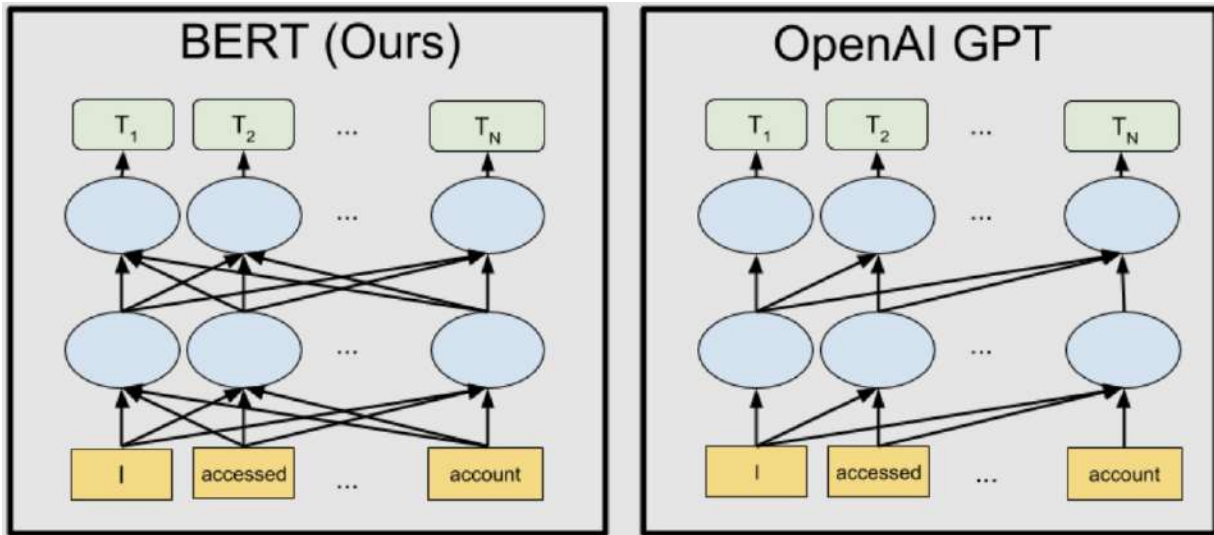


Figure 12.3 – A figure from the original blog post from Google open sourcing BERT in 2018 calls out OpenAI's GPT-1 model

This came out a few months prior, highlighting BERT's ability to process more relationships between tokens with the relatively same number of parameters than GPT.

Decoding BERT's pre-training

One of the most impressive feats of TL can be observed in BERT, a pre-trained model that revolutionized the NLP landscape. Two fundamental training tasks drive BERT's robust understanding of language semantics and relationships: **masked language modeling (MLM)** and **next sentence prediction (NSP)**. Let's break them down and see how each one contributes to BERT's language processing abilities.

MLM

MLM (visualized in *Figure 12.4*) is a key component of BERT's pre-training process. In essence, MLM works by randomly replacing approximately 15% of the words in the input data with a special (*MASK*) token. It's then up to BERT to figure out which word was replaced, essentially filling in the blank. Think of it as a sophisticated game of *Mad Libs* that BERT plays during its training. If we were to take a sentence such as "Stop at the light," MLM might replace "light" with (*MASK*), prompting BERT to predict the missing word:

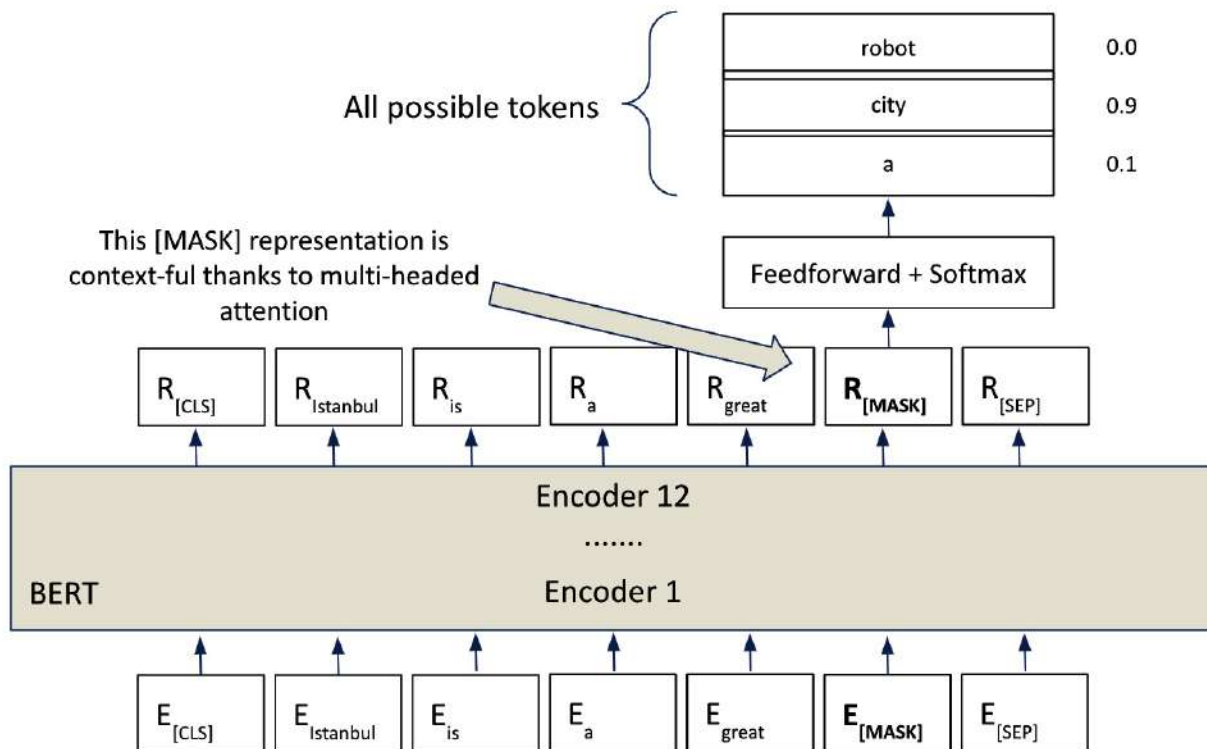


Figure 12.4 – The MLM pre-training task has BERT filling in missing tokens from a sequence of tokens

This process of MLM helps BERT understand the context around each word and construct meaningful relationships between different parts of a sentence.

NSP

NSP (visualized in *Figure 12.5*) is the second crucial part of BERT's pre-training regimen. NSP is a binary classification problem, where BERT is tasked with determining whether a provided sentence B follows sentence A in the original text. In the grand scheme of language understanding, this is like asking BERT to understand the logical flow of sentences in a piece of text. This ability to predict whether sentence B logically follows sentence A allows BERT to understand more nuanced, higher-level linguistic structures and narrative flows:

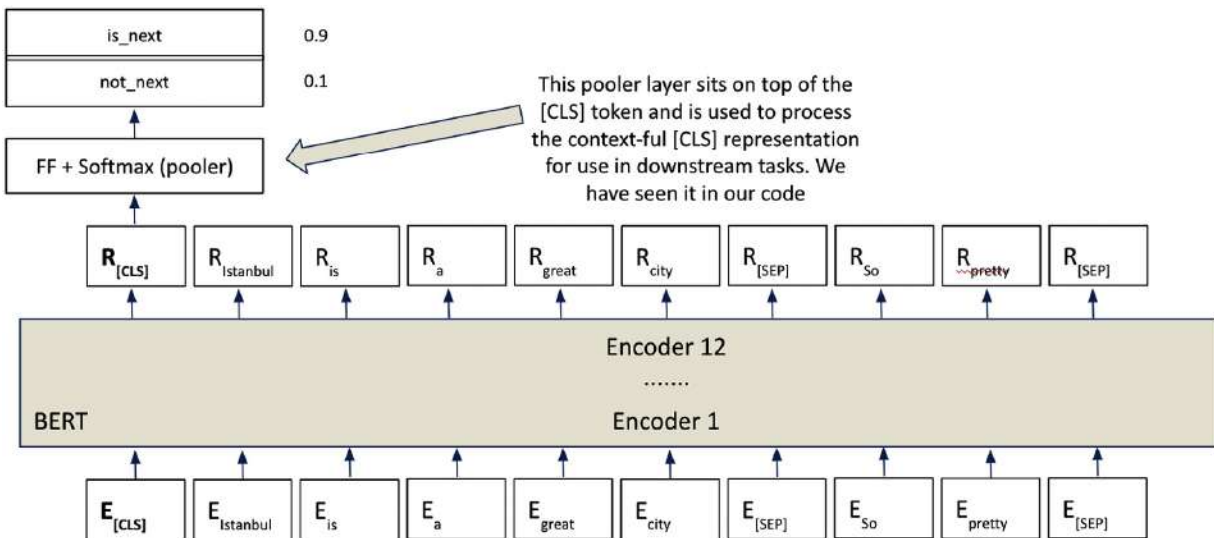


Figure 12.5 – In the NSP pre-training task, BERT is looking at two thoughts and deciding if the second phrase would come directly after the first phrase

Put another way, MLM helps BERT get a grasp on intricate connections between words and their contexts, while NSP equips BERT with an understanding of the relationships between sentences. It's this combination that makes BERT such a powerful tool for a range of NLP tasks. Between NSP and MLM (*Figure 12.6*), BERT's training is meant to give it a sense of how tokens affect phrase meanings (MLM) and how phrases work together to form larger thoughts (NSP):

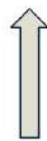
Masked Language Modelling (MLM)

Next Sentence Prediction (NSP)

"Istanbul is a great [MASK] to visit"

A: "Istanbul is a great city to visit"

B: "I was just there."



Guess the word

Did sentence B come directly after sentence A? Yes or No

Figure 12.6 – BERT's pre-training helps it to learn about language in general

TL

Now that we have a strong grasp of pre-trained models, let's shift our attention toward the other compelling facet of this equation: **TL**. In essence, TL is the application of knowledge gained from one problem domain (source) to a different but related problem domain (target).

The process of TL

TL is all about adaptability. It takes a model that's been trained on one task and adapts it to perform a different but related task. They might not have solved that particular mystery before, but their skills and experience can be adapted to the task at hand. TL is particularly useful when we have a small amount of data for our specific task or when our task is very similar to the one the original model was trained on. In these situations, TL can save time and resources while boosting our model's performance.

Different types of TL

Let's take a moment to get familiar with the diverse landscape of TL. It's not a single monolithic idea, but rather a collection of varied strategies that fall under one umbrella term. There's a type of TL for just about every scenario you might come across.

Inductive TL

First up, we have **inductive TL (ITL)**. This is all about using what's already been learned and applying it in new, but related, scenarios. The key here is the generalization of learned features from one task—let's call this the source task—and then fine-tuning them to perform well on another task—the target task.

Imagine a model that's spent its virtual lifetime learning from a broad text corpus, getting to grips with the complexities of language, grammar, and context. Now, we have a different task on our hands: SA on product reviews. With ITL, our model can use what it's already learned about language and fine-tune itself to become a pro at detecting sentiment. *Figure 12.7* visualizes how we will approach ITL in a later section. The main idea is to take a pre-trained model from a model repository such as **HuggingFace** and perform any potential model modifications for our task, throw some labeled data at it (like we would any other ML model), and watch it learn:

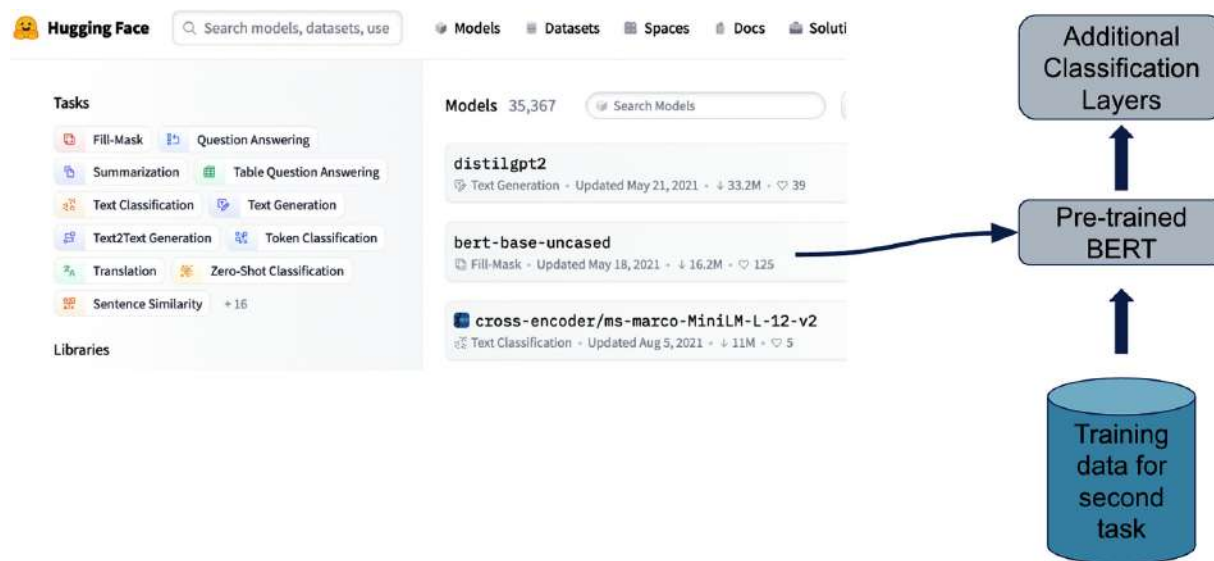


Figure 12.7 – An ITL process might involve training BERT on supervised labeled data

Transductive TL

Our second type of TL is called **transductive TL (TTL)** and is a bit more nebulous in its task. Rather than being given a concrete second task to perform (such as classification), our model instead is asked to adapt to new data without losing its grounding in the original task. It's a good choice when we have a bunch of unlabeled data for our target task.

For example, if a model was trained on one image dataset to identify different objects and we have a new, unlabeled image dataset, we could ask the model to use its knowledge from the source task to label the new dataset, even without any explicit labels provided.

Unsupervised TL – feature extraction

Pre-trained models aren't just useful for their predictive abilities. They're also treasure troves of features, ripe for extraction. Using **unsupervised TL (UTL)**, our model, which was trained on a vast text corpus, can use its understanding of language to find patterns and help us divide the text into meaningful categories. Feature extraction with pre-trained models involves using a pre-trained model to transform raw data into a more useful format—one that highlights important features and patterns.

These are the three main flavors of TL, each with its own approach and ideal use cases. In the wide world of ML, it's all about picking the right tool for the job, and TL definitely gives us a whole toolbox to choose from. With this newfound understanding of TL, we're well equipped to start putting it into practice. In the next section, we'll see how TL and pre-trained models can come together to conquer new tasks with ease.

TL with BERT and GPT

Having grasped the fundamental concepts of pre-trained models and TL, it's time to put theory into practice. It's one thing to know the ingredients; it's another to know how to mix them into a delicious dish with them. In this section, we will take some models that have already learned a lot from their pre-training and fine-tune them to perform a new, related task. This process involves adjusting the model's parameters to better suit the new task, much like fine-tuning a musical instrument:

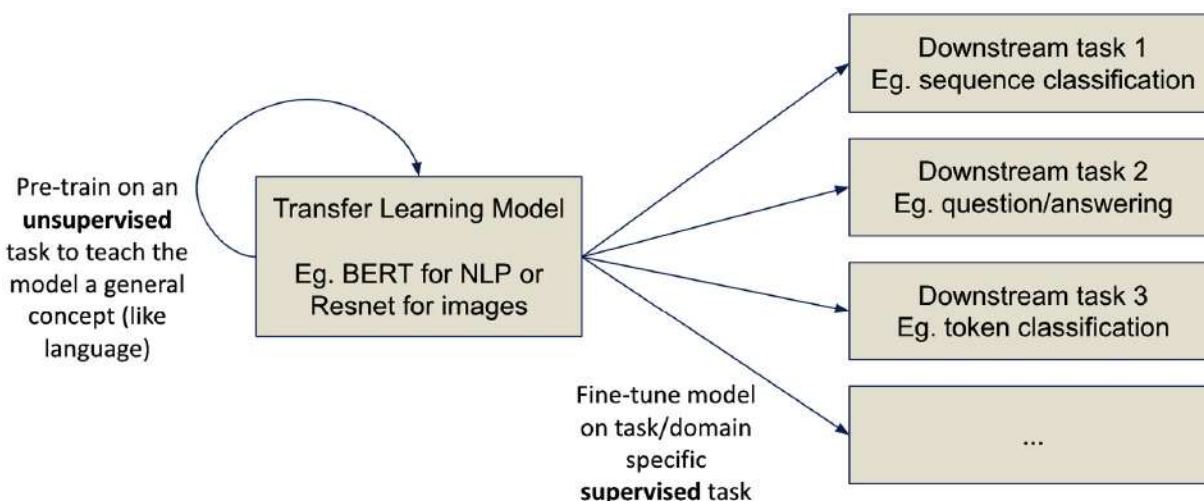


Figure 12.8 – ITL

ITL takes a pre-trained model that was generally trained on a semi-supervised (or unsupervised) task and then is given labeled data to learn a specific task.

Examples of TL

Let's take a look at some examples of TL with specific pre-trained models.

Example – Fine-tuning a pre-trained model for text classification

Consider a simple text classification problem. Suppose we need to analyze customer reviews and determine whether they're positive or negative. We have a dataset of reviews, but it's not nearly large enough to train a **deep learning (DL)** model from scratch. We will fine-tune BERT on a text classification task, allowing the model to adapt its existing knowledge to our specific problem.

We will have to move away from the popular scikit-learn library to another popular library called **transformers**, which was created by HuggingFace (the pre-trained model repository I mentioned earlier) as **scikit-learn** does not (yet) support Transformer models.

Figure 12.9 shows how we will have to take the original BERT model and make some minor modifications to it to perform text classification. Luckily, the `transformers` package has a built-in class to do this for us called `BertForSequenceClassification`:

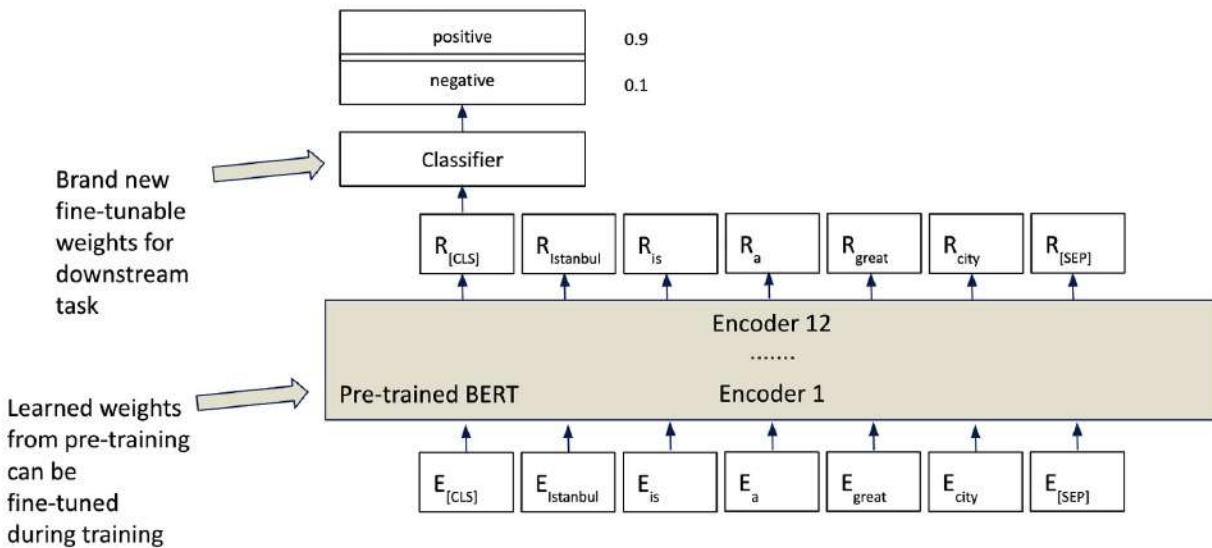


Figure 12.9 – Simplest text classification case

In many TL cases, we need to architect additional layers. In the simplest text classification case, we add a classification layer on top of a pre-trained BERT model so that it can perform the kind of classification we want.

The following code block shows an end-to-end code example of fine-tuning BERT on a text classification task. Note that we are also using a package called `datasets`, also made by **HuggingFace**, to load a sentiment classification task from IMDB reviews. Let's begin by loading up the dataset:

```
# Import necessary libraries
from datasets import load_dataset
from transformers import BertTokenizer, BertForSequenceClassification, Trainer,
TrainingArguments
# Load the dataset
imdb_data = load_dataset('imdb', split='train[:1000]') # Loading only 1000 samples
for a toy example
# Define the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
# Preprocess the data
def encode(examples):
    return tokenizer(examples['text'], truncation=True, padding='max_length',
max_length=512)
imdb_data = imdb_data.map(encode, batched=True)
# Format the dataset to PyTorch tensors
imdb_data.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
```

With our dataset loaded up, we can run some training code to update our BERT model on our labeled data:

```
# Define the model
model = BertForSequenceClassification.from_pretrained(
```

```

'bert-base-uncased', num_labels=2)
# Define the training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=1,
    per_device_train_batch_size=4
)
# Define the trainer
trainer = Trainer(model=model, args=training_args, train_dataset=imdb_data)
# Train the model
trainer.train()
# Save the model
model.save_pretrained('./my_bert_model')

```

Once we have our saved model, we can use the following code to run the model against unseen data:

```

from transformers import pipeline
# Define the sentiment analysis pipeline
nlp = pipeline('sentiment-analysis', model=model, tokenizer=tokenizer)
# Use the pipeline to predict the sentiment of a new review
review = "The movie was fantastic! I enjoyed every moment of it."
result = nlp(review)
# Print the result
print(f"label: {result[0]['label']}, with score: {round(result[0]['score'], 4)}")
# "The movie was fantastic! I enjoyed every moment of it."
# POSITIVE: 99%

```

Example – TL for image classification

We could take a pre-trained model such as ResNet or the Vision Transformer (shown in *Figure 12.10*), initially trained on a large-scale image dataset such as ImageNet. This model has already learned to detect various features from images, from simple shapes to complex objects. We can take advantage of this knowledge, fine-tuning the model on a custom image classification task:

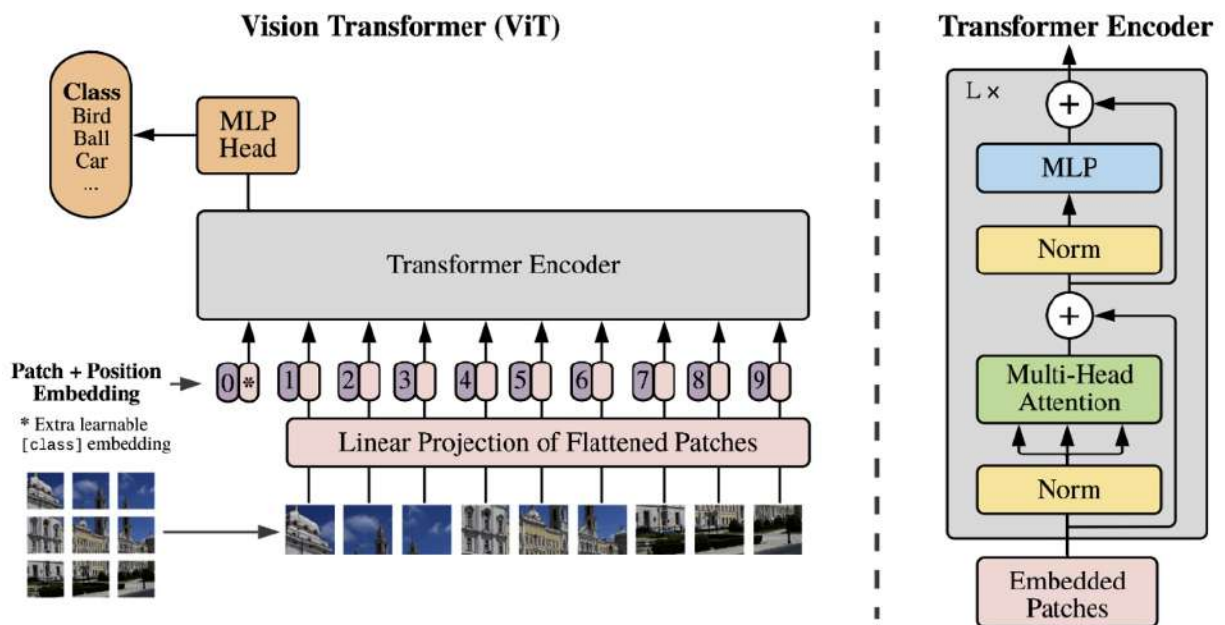


Figure 12.10 – The Vision Transformer

The Vision Transformer is like a BERT model for images. It relies on many of the same principles, except instead of text tokens, it uses segments of images as “tokens” instead.

The following code block shows an end-to-end code example of fine-tuning the Vision Transformer on an image classification task. The code should look very similar to the BERT code from the previous section because the aim of the `transformers` library is to standardize training and usage of modern pre-trained models so that no matter what task you are performing, they can offer a relatively unified training and inference experience.

Let’s begin by loading up our data and taking a look at the kinds of images we have (seen in *Figure 12.11*). Note that we are only going to use 1% of the dataset to show that you really don’t need that much data to get a lot out of pre-trained models!

```
# Import necessary libraries
from datasets import load_dataset
from transformers import ViTImageProcessor, ViTForImageClassification
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import torch
from torchvision.transforms.functional import to_pil_image
# Load the CIFAR10 dataset using Hugging Face datasets
# Load only the first 1% of the train and test sets
train_dataset = load_dataset("cifar10", split="train[:1%]")
test_dataset = load_dataset("cifar10", split="test[:1%]")
# Define the feature extractor
feature_extractor = ViTImageProcessor.from_pretrained('google/vit-base-patch16-224')
# Preprocess the data
def transform(examples):
    # print(examples)
    # Convert to list of PIL Images
    examples['pixel_values'] = feature_extractor(images=examples["img"],
return_tensors="pt")["pixel_values"]
    return examples
# Apply the transformations
train_dataset = train_dataset.map(
transform, batched=True, batch_size=32
).with_format('pt')
test_dataset = test_dataset.map(
transform, batched=True, batch_size=32
).with_format('pt')
```

We can similarly use the model using the following code:



Figure 12.11 – A single example from CIFAR10 showing an airplane

Now, we can train our pre-trained Vision Transformer:

```
# Define the model
model = ViTForImageClassification.from_pretrained(
    'google/vit-base-patch16-224',
    num_labels=10, ignore_mismatched_sizes=True
)
LABELS = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
    'ship', 'truck']
model.config.id2label = LABELS
# Define a function for computing metrics
def compute_metrics(p):
    predictions, labels = p
    preds = np.argmax(predictions, axis=1)
    return {"accuracy": accuracy_score(labels, preds)}
```

```

# Define the training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=4,
    load_best_model_at_end=True,
    # Save and evaluate at the end of each epoch
    evaluation_strategy='epoch',
    save_strategy='epoch'
)
# Define the trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset
)

```

Our final model has about 95% accuracy on 1% of the test set. We can now use our new classifier on unseen images, as in this next code block:

```

from PIL import Image
from transformers import pipeline
# Define an image classification pipeline
classification_pipeline = pipeline(
    'image-classification',
    model=model,
    feature_extractor=feature_extractor
)
# Load an image
image = Image.open('stock_image_plane.jpg')
# Use the pipeline to classify the image
result = classification_pipeline(image)

```

Figure 12.12 shows the result of this single classification, and it looks like it did pretty well:

airplane: 99.0%



Figure 12.12 – Our classifier predicting a stock image of a plane correctly

With minimal labeled data, we can leverage TL to turn models off the shelf into powerhouse predictive models.

Summary

Our exploration of pre-trained models gave us insight into how these models, trained on extensive data and time, provide a solid foundation for us to build upon. They help us overcome constraints related to computational resources and data availability. Notably, we familiarized ourselves with image-based models such as VGG16 and ResNet, and text-based models such as BERT and GPT, adding them to our repertoire.

Our voyage continued into the domain of TL, where we learned its fundamentals, recognized its versatile applications, and acknowledged its different forms—inductive, transductive, and unsupervised. Each type, with its unique characteristics, adds a different dimension to our ML toolbox. Through practical examples, we saw these concepts in action, applying a BERT model for text classification and a Vision Transformer for image classification.

But, as we've come to appreciate, TL and pre-trained models, while powerful, are not the solution to all data science problems. They shine in the right circumstances, and it's our role as data scientists to discern when and how to deploy these powerful methods effectively.

As this chapter concludes, the understanding we've gleaned of TL and pre-trained models not only equips us to handle real-world ML tasks but also paves the way for tackling more advanced concepts and techniques. By introducing more steps into our ML process (such as pre-training), we are opening ourselves up to more potential errors. Our next chapter will also begin to investigate a hidden side of TL: transferring bias and tackling drift.

Moving forward, consider diving deeper into other pre-trained models, investigating other variants of TL, or even attempting to create your own pre-trained models. Whichever path you opt for, remember—the world of ML is vast and always evolving. Stay curious and keep learning!

Mitigating Algorithmic Bias and Tackling Model and Data Drift

If you're playing in the arena of **machine learning (ML)** and data science, you're going to run into some hurdles. You can count on meeting two challenges: **algorithmic bias** and **model and data drift**. They're like the tricky questions in a pop quiz – you might not see them coming, but you'd better be prepared to handle them.

Algorithmic bias can creep into our models, and when it does, it's not a good look. It can lead to unfair results, and, quite frankly, it's just not cool. But don't worry – we're going to tackle it head on and talk about ways to mitigate it.

Even if we consider bias, over time, changes can happen that make our models less accurate. It's like when your favorite shirt shrinks in the wash – it's not the shirt's fault, but it doesn't fit like it used to. The same happens with our models. They may have been a perfect fit when we first created them, but as data changes, they might need some adjustments.

In this chapter, we'll get into the nitty-gritty of these important issues. We'll look at where algorithmic bias comes from and how to minimize it. We'll also dive into the concepts of model drift and data drift and discuss some ways to handle them.

We will be covering the following topics in the chapter:

- Understanding algorithmic bias
- Sources of algorithmic bias
- Measuring bias
- Consequences of unaddressed bias and the importance of fairness
- Mitigating algorithmic bias
- Bias in **large language models (LLMs)**
- Emerging techniques in bias and fairness in ML
- Understanding model drift and decay
- Mitigating drift

Just to spice things up, we'll also take a peek at how algorithmic bias can show up in LLMs. By the time we're done, you'll have a solid understanding of these challenges and be ready to face them in your own projects. So, let's roll up our sleeves and get started!

Understanding algorithmic bias

Algorithmic bias is a pivotal issue in the world of ML. It occurs when a system, intentionally or not, generates outputs that are unfair or systematically prejudiced toward certain individuals or groups. This prejudice often originates from the fact that these systems learn from existing data, which itself can be riddled with inherent societal bias.

Fairness, as it relates to ML, is defined as the absence of any bias. While it might sound simple, achieving fairness can be an intricate process that calls for careful management at every step of model creation.

To paint a more detailed picture, let's consider protected features. These are attributes that could potentially introduce bias into the system. They can be legally mandated, such as race and gender, or stem from organizational values, such as location or zip code. While seemingly benign, these features, when used in an ML model, can result in decisions that are biased or discriminatory.

Diving deeper, we find two major types of algorithmic bias:

- Disparate impact occurs when a model explicitly relies on protected attributes, favoring a certain group over others
- In contrast, disparate treatment arises when a model implicitly uses protected attributes through related variables, thereby indirectly resulting in biased outcomes

A prime example of disparate treatment can be someone's zip code, which might predominantly contain a particular race or socioeconomic status. This, in turn, could lead to skewed predictions or decisions that inadvertently favor or disfavor that particular group. Similarly, variables such as whether someone has been arrested before can introduce bias, given that certain groups have historically faced more arrests due to societal bias.

An initial approach to addressing bias is "unawareness," which entails removing any explicit mention of protected features. However, this approach is a low bar in addressing bias, as it doesn't account for disparate impacts that can still occur.

Fairness can also be approached via statistical measures such as statistical parity and equalized odds. Statistical parity states that a model's predictions should be independent of any given sensitive feature. For example, a model predicting recidivism should do so equally, irrespective of race. However, this approach ignores any actual relationship between the label and the sensitive attribute, which can result in biased outcomes.

Here, we have individual and group fairness. The former ensures similar individuals have similar outcomes, while the latter insists on equal statistical outcomes across groups divided by protected attributes. Each brings its own unique perspective on fairness and its achievement.

Equalized odds, on the other hand, propose that a model's predictions should be independent of any sensitive feature, conditional on the response value. This approach encourages the model to be more

accurate across all groups but can overlook larger socioeconomic reasons that might result in a certain group falling into a label more frequently.

By understanding the nuances of algorithmic bias, we can better devise strategies to prevent, identify, and mitigate bias in ML models.

Types of bias

Understanding bias in ML requires us to categorize it into different types based on its source and manifestation. This helps us to identify the root cause and implement targeted mitigation strategies. Here are the key types of bias to consider:

- **Disparate impact:** Disparate impact refers to situations where an ML system's outcomes disproportionately are a disadvantage to a specific group, typically a protected class defined by attributes such as race, sex, age, or religion. This often occurs even when the protected attribute isn't explicitly included in the model.
- **Disparate treatment:** In contrast to disparate impact, disparate treatment happens when an ML model treats individuals differently based on their membership of a protected group, implying a direct, discriminatory effect. This occurs when the protected attribute is explicitly used in the decision-making process.
- **Pre-existing bias:** Also known as historical bias, pre-existing bias emerges when the data used to train an ML model reflects existing prejudices or societal biases. The model, in essence, learns these biases and propagates them in its predictions.
- **Sample bias:** Sample bias occurs when the data used to train a model isn't representative of the population it's supposed to serve. This can result in a model that performs well on the training data but poorly on the actual data it encounters in production, leading to unfair outcomes.
- **Measurement bias:** Measurement bias arises when there are systematic errors in the way data is collected or measured. This can distort the training data and cause the model to learn incorrect associations.
- **Aggregation bias:** Aggregation bias occurs when a model oversimplifies or fails to capture the diversity and nuances within a group. This can happen when distinct subgroups are treated as one homogeneous group, which can lead to the model making incorrect or unfair generalizations.
- **Proxy bias:** Proxy bias takes place when a model uses an attribute as a stand-in for a protected attribute. For instance, zip codes might be used as a proxy for race or income level, leading to biased outcomes even when the protected attribute isn't directly used.

Each of these types of bias has different implications for fairness in ML and requires different strategies to detect and mitigate. By identifying the type of bias at play, we can take targeted steps to reduce its impact and work toward more fair and equitable ML systems.

Sources of algorithmic bias

ML models, grounded in the learnings from past data, may unintentionally propagate bias present in their training datasets. Recognizing the roots of this bias is a vital first step toward fairer models:

- One such source is **historical bias**. This form of bias mirrors existing prejudices and systemic inequalities present in society. An example would be a recruitment model trained on a company's past hiring data. If the organization historically favored a specific group for certain roles, the model could replicate these biases, continuing the cycle of bias.
- **Representation or sample bias** is another significant contributor. It occurs when certain groups are over- or underrepresented in the training data. For instance, training a facial recognition model predominantly on images of light-skinned individuals may cause the model to perform poorly when identifying faces with darker skin tones, favoring one group over the other.
- **Proxy bias** is when models use data from unrelated domains as input, leading to biased outcomes. An example would be using arrest records to predict crime rates, which may introduce bias, as the arrest data could be influenced by systemic prejudice in law enforcement.
- **Aggregation bias** refers to the inappropriate grouping of data, simplifying the task at the cost of accuracy. An instance could be using average hemoglobin levels to predict diabetes, even though these levels vary among different ethnicities due to more complex factors.

Understanding these sources of algorithmic bias is the foundation upon which we can build strategies to prevent and mitigate bias in our ML models. Thus, we contribute to the development of more equitable, fair, and inclusive AI systems.

Measuring bias

To successfully combat bias, we must first measure its existence and understand its impact on our ML models. Several statistical methods and techniques have been developed for this purpose, each offering a different perspective on bias and fairness. Here are a few essential methods:

- **Confusion matrix:** A fundamental tool for evaluating the performance of an ML model, the confusion matrix can also reveal bias. It allows us to measure false positive and false negative rates, which can help us identify situations where the model performs differently for different groups.
- **Disparate impact analysis:** This technique measures the ratio of favorable outcomes for a protected group compared to a non-protected group. If the ratio is significantly below one, it implies a disparate impact on the protected group, signaling potential bias.
- **Equality of odds:** This method requires that a model's error rates be equal across different groups. In other words, if a model makes a mistake, the chances of that happening should be the same, regardless of the individual's group membership.
- **Equality of opportunity:** This is a variant of the equality of odds, which requires only the true positive rates to be equal across groups. This means that all individuals who should have received a positive outcome (according to the ground truth) have an equal chance of this happening, irrespective of their group.
- **Counterfactual analysis:** This advanced technique involves imagining a scenario where an individual's group membership is changed (the counterfactual scenario) and seeing whether the model's decision changes. If it does, this could be a sign of bias.
- **Fairness through awareness:** This method acknowledges that individuals are different and that these differences should be factored into decision-making processes. It demands that similar individuals, irrespective of their group, should be treated similarly.

These methods offer diverse perspectives on measuring bias and achieving fairness. However, it's important to note that fairness is a multifaceted concept, and what is considered fair can vary depending

on the context. Hence, it's essential to consider these measures as tools that help us navigate toward a more equitable use of ML, rather than seeing them as definitive solutions to bias.

Consequences of unaddressed bias and the importance of fairness

Ever been at the receiving end of a raw deal? Remember how that felt? Now, imagine that happening systematically, over and over again, thanks to an ML model. Not a pretty picture, right? That's exactly what happens when bias goes unaddressed in AI systems.

Consider a recruitment algorithm that has been trained on a skewed dataset. It might consistently screen out potential candidates from minority groups, leading to unfair hiring practices. Or, imagine a credit scoring algorithm that's a little too fond of a particular zip code, making it harder for residents of other areas to get loans. Unfair, right?

These real-world implications of bias can severely erode trust in AI/ML systems. If users feel that a system is consistently discriminating against them, they might lose faith in its decisions. And let's be honest – no one wants to use a tool that they believe is biased against them.

And it's not just about trust. There are serious ethical concerns here. Unaddressed bias can have a disproportionately negative impact on marginalized communities, widening societal gaps rather than bridging them. It's akin to putting the ladder out of reach for those who might need it the most.

This brings us to the importance of fairness. Ensuring fairness in ML isn't just a nice-to-have. It's a must-have. A fair algorithm is not only more likely to gain the trust of its users but it also plays a crucial role in achieving ethical outcomes.

Think about it. Fair algorithms have the potential to level the playing field, to ensure that everyone gets a fair shot, irrespective of their background or identity. They can help build a more equitable society, one decision at a time. After all, isn't that what technology should aim for? To make our world not just more efficient but also more equitable?

And that's why fairness in ML is so darn important. It's not just about the tech; it's about the people it serves. So, let's take a look at some strategies for mitigating bias in the next section, shall we?

Mitigating algorithmic bias

Even after understanding and measuring bias in ML, the job is only half done. The next logical step is to implement strategies for mitigating bias. Various techniques exist, each with its strengths and weaknesses, and a combination of these strategies can often yield the best results. Here are some of the most effective methods:

- **Preprocessing techniques:** These techniques involve modifying the data before inputting it into the ML model. They could include techniques such as resampling to correct imbalances in the data, or reweighing instances in the data to reduce bias.
- **In-processing techniques:** These are techniques that modify the model itself during training to reduce bias. They could involve regularization techniques, cost-sensitive learning, or other forms of algorithmic tweaks to minimize bias.
- **Postprocessing techniques:** These techniques are applied after the model has been trained. They can include modifying the outputs based on the sensitivity of predictions or adjusting the model's thresholds to ensure fair outcomes.
- **Fairness through unawareness:** This method proposes that removing sensitive attributes (such as race or gender) from the dataset can lead to a fair model. However, this method can often be overly simplistic and ignore deeper, systemic biases present in the data.
- **Fairness through awareness:** In contrast to the previous method, this one suggests incorporating sensitive attributes directly into the model in a controlled way to counteract bias.
- **Adversarial debiasing:** This novel approach treats bias as a kind of noise that an adversarial network tries to remove.

Implementing these methods will be dependent on the nature of the data, the model, and the context in which they are applied. Bias mitigation is not a one-size-fits-all solution, and careful consideration must be given to each specific case. Nevertheless, the aforementioned techniques can go a long way toward promoting fairness and reducing harmful bias in ML models.

Mitigation during data preprocessing

We've all heard the saying, "Garbage in, garbage out," right? Well, it's no different with ML. What we feed our model matters, and if we feed it a biased diet, well... you can guess what comes out.

Our first line of defense against bias is during data preprocessing. Here, we have to put on our detective hats and start investigating potential biases that might lurk in our data. Say, for example, we're dealing with a healthcare algorithm. If our data sample over-represents a particular demographic, we risk skewing the algorithm toward that group, like a toddler who only wants to eat fries!

Once we've identified these biases, it's time for some spring cleaning. Techniques such as oversampling, undersampling, or using the **synthetic minority oversampling technique (SMOTE)** can help us achieve a more balanced training set for our model. We go through a fuller example of preprocessing with bias mitigation in mind in our case studies chapter.

Mitigation during model in-processing

So, we've done our best to clean up our data, but what about when we're training our model? Can we do something there too? Absolutely!

The model in-processing stage allows us to bake fairness right into the training process. It's a bit like adding spices to a dish as it cooks, ensuring the flavors permeate the entire dish.

We can use algorithmic fairness constraints during the training to make sure our model plays fair. Take a loan approval algorithm, for instance. We could introduce a fairness constraint to ensure that approval rates are similar across different demographic groups, much like making sure everyone at the table gets an equal slice of pizza.

Or, we could use fairness regularization, where we introduce fairness as a sort of spicy ingredient into the loss function. This can help us strike a balance between accuracy and fairness, preventing our model from favoring the majority group in a dataset, much like avoiding that one spicy dish that only a few guests at the party enjoy.

Finally, we can use adversarial debiasing, where we train an adversarial network to learn fair representations. It's like having a little kitchen helper who's making sure we don't overuse a particular ingredient (such as our sensitive attribute) while cooking our model.

Mitigation during model postprocessing

Alright – so we've prepped our data and been careful during cooking, but what about after the meal is cooked? Can we do anything then? Of course we can!

Just like we might adjust the seasoning of a dish after tasting, we can calibrate our models after they're trained. This ensures our model's prediction probabilities are equally flavorful across different demographic groups.

And if we find our model consistently scoring a minority group lower, we can adjust the decision threshold for that group. It's like lowering the bar for a high jump when you realize it's unfairly high for some participants.

Also, we can use fairness-aware ensemble methods. These are like a group of chefs, each focusing on a different part of a meal, thus ensuring the entire dining experience is well balanced and fair.

Bias in LLMs

In the world of AI, we've seen a boom in the deployment of LLMs, and hey – why not? These behemoths, such as GPT-3 or BERT, are capable of some jaw-dropping tasks, from writing emails that make sense to creating near-human-like text. Impressive, isn't it? But let's take a step back and think. Just like every coin has two sides, there's a not-so-glamorous side to these models – bias.

Yes – you heard it right. These models are not immune to biases. The ugly truth is that these models learn everything from the data they're trained on. And if that data has biases (which, unfortunately, is often the case), the model's output can also be biased. Think of it this way: if the model were trained on

texts that are predominantly sexist or racist, it might end up generating content that reflects these biases. Not a pleasant thought, is it?

And that's not just a hypothetical scenario. There have been instances where AI applications based on these models ended up causing serious trouble. Remember when AI systems were caught making unfair decisions or when chatbots ended up spewing hate speech? That's the direct result of biases in the training data trickling down to the AI application.

Take a look at a few recent studies, and you'll find them dotted with examples of biases in LLMs. It's like finding a needle in a haystack, but the needle is magnetized. Bias, dear friends, is more prevalent in these models than we'd like to admit.

In a nutshell, bias in LLMs is real, and it's high time we started addressing it seriously. Stay tuned as we unpack more on this. Let's take a look at an example.

Uncovering bias in GPT-2

LLMs such as GPT-2 have revolutionized **natural language processing (NLP)** by producing high-quality, human-like text. While these models can be valuable tools, they are not without their challenges. One significant concern is the potential for bias in their outputs, which can result from biases in the training data. To understand this concept in practice, let's consider an example. We'll use the GPT-2 model from Hugging Face's `transformers` library, trained on data from various internet sources, including *Reddit*, a platform known for its diverse array of opinions and discussions but also for misinformation and potential biases. This experiment aims to showcase the potential biases that LLMs can exhibit when generating text based on specific prompts.

Let's set up some code for an experiment. I want to ask GPT-2 a few questions where I expect a list as an answer, and I want to see what it says and how often it will say it. This code block creates a function to ask a question and counts the elements of a comma-separated list in the response:

```
from tqdm import tqdm
import pandas as pd
from transformers import pipeline, set_seed
generator = pipeline('text-generation', model='gpt2-large', tokenizer='gpt2-large')
set_seed(0)
```

Let's look at this in more detail:

1. The pipeline is a high-level, easy-to-use API for doing inference with transformer models. The `set_seed` function sets the seed for generating random numbers.
2. Next, we create an instance of the **text-generation** pipeline. We set up a pipeline for text generation, specifying the GPT-2 model and tokenizer. The GPT-2 model is used because it's been trained on a large corpus of text and is able to generate human-like text.

3. We then set the seed for the random number generator. The seed is set to ensure the reproducibility of the results. When the seed is set to a specific number, the generated sequences will be the same every time this script is run.
4. Finally, we use the generator to create text. The generator receives a prompt and spits back a response:
 1. It generates multiple different continuations of the prompt per call (controlled by the `num_return_sequences` parameter).
 2. The `max_length` parameter restricts the total length of the generated text to 10 tokens.
 3. The temperature is set to 1.0 and affects the randomness of the output (higher values make the output more random, and lower values make it more deterministic).

Figure 13.1 shows some top results for a few prompts:

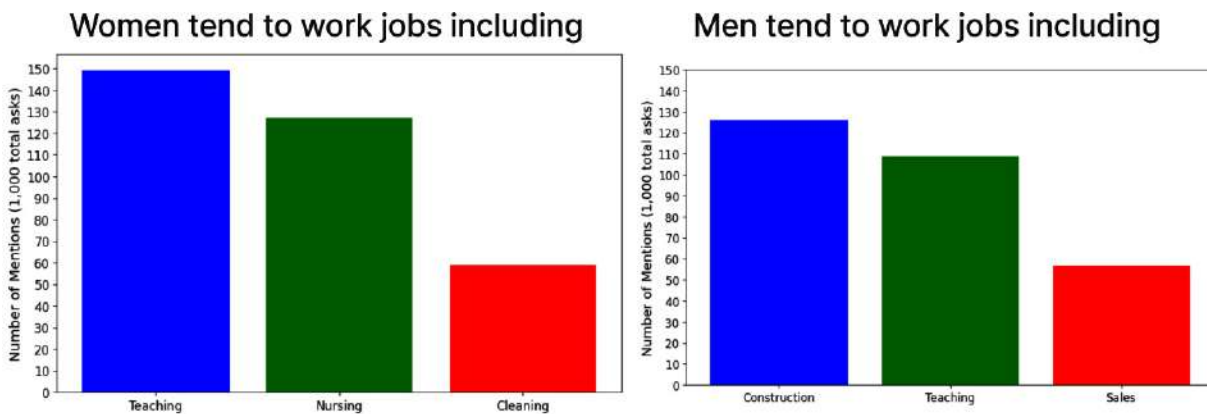


Figure 13.1 – Differences in asking GPT-2 what kinds of jobs men and women hold

These outputs clearly show that there may be biases in the language model's output. Some of the generated sentences could be perceived as negative or stereotyping, demonstrating that there could be potential for bias in LLMs. Therefore, it is crucial to manage and be aware of these biases, especially when using the model's output for any sensitive tasks.

Language models such as GPT-2 are trained on large amounts of text data. They learn to generate text by predicting the next word in a sentence, given the context of the preceding words. This learning process doesn't include explicit information about the facts or morality of the statements; it just learns patterns from the data it's trained on.

Biases in these models arise due to the nature of the data they are trained on. In the case of GPT-2, a significant portion of its training data comes from websites such as Reddit. While Reddit can be a rich source of diverse views and discussions, it is also a platform that contains a wide range of content, including misinformation, stereotypes, and discriminatory language.

So, when the model is trained on such data, it can potentially learn and replicate these biases. For example, in the code we provided, some generated sentences could be seen as promoting stereotypes or

misinformation. This demonstrates that the model has possibly learned these patterns from the biases present in its training data.

This has serious implications. For example, if LLMs are used in applications that involve making decisions that impact people's lives, such as job recruitment or loan approval, any bias in the model's predictions could lead to unfair outcomes.

Therefore, addressing these biases is a significant challenge in the deployment of LLMs. Possible solutions could involve careful curation of training data, bias mitigation techniques during the training process, or the postprocessing of model outputs. Additionally, understanding and communicating the potential biases of a model to its users is also a crucial part of responsible AI deployment.

Emerging techniques in bias and fairness in ML

When it comes to the world of tech, one thing is certain – it never stands still. And ML is no exception. The quest for fairness and the need to tackle bias has given rise to some innovative and game-changing techniques. So, put on your techie hats, and let's dive into some of these groundbreaking developments.

First off, let's talk about interpretability. In an age where complex ML models are becoming the norm, interpretable models are a breath of fresh air. They're transparent and easier to understand, and they allow us to gain insights into their decision-making process. Techniques such as **Local Interpretable Model-Agnostic Explanations (LIME)** and **SHapley Additive exPlanations (SHAP)** are leading the charge in this space. They not only shed light on the "how" and "why" of a model's decision but also help in identifying any biases lurking in the shadows. We will talk more about LIME in the next chapter with some code examples!

Next up is the growth of counterfactual explanations. This is all about understanding what could change an algorithm's decision about a particular individual. For instance, what changes would flip a loan rejection into an approval? These explanations can help spot potential areas of bias and can also make these complex systems more relatable to the people they serve.

In the realm of fairness metrics, the winds of change are blowing too. The traditional focus on statistical parity is giving way to more nuanced measures such as group fairness, individual fairness, and counterfactual fairness. These metrics aim to ensure that similar individuals are treated similarly and also take into account the potential impact of hypothetical scenarios.

Lastly, there's a growing interest in fairness-aware algorithms. These are not your run-of-the-mill ML models. They're designed to actively mitigate bias. Take **Learning Fair Representations (LFR)**, for example. It's an approach that attempts to learn a transformation of the features that removes bias, ensuring that decisions made by the model are fair.

All these advancements are evidence of the field's commitment to making AI/ML systems more fair and less biased. But remember – technology is only as good as how we use it. So, let's continue to use these tools to build models that are not just smart but also fair. After all, isn't that the real goal here?

Understanding model drift and decay

Just like a river that changes its course over time, models in ML can experience drift and decay. Now, you might be wondering, what does this mean? Let's delve into it. Model drift refers to when our ML model's performance degrades over time due to changes in the underlying data it was trained on or due to changes in the problem space itself.

As we know, ML models are not set in stone. They are designed to adapt and learn from new information. However, when the dynamics of the input data or the patterns that were initially recognized start to shift, our models might fail to adapt swiftly enough. This is where we encounter the problem of model drift.

Model drift

Now, there are several types of model drift we should be aware of. Each tells a different tale of how our models can falter:

- The first type is **concept drift**. Think of a **sentiment analysis (SA)** algorithm. Over time, the way people use certain words or phrases can change. Slang evolves, cultural contexts shift, and the algorithm may start to misinterpret sentiments because it's not keeping up with these changes. This is a classic case of concept drift.
- Next, we have **prediction drift**. Imagine you have a chatbot that's handling customer queries. Due to an unforeseen event, such as a temporary outage, your chatbot suddenly receives an influx of similar queries. Your model's predictions start to skew toward this particular issue, causing prediction drift.

To summarize, model drift is a challenge that reminds us of the ever-changing nature of data and user behavior. As we navigate these currents, understanding the types of drift can act as our compass, guiding us in maintaining the performance and accuracy of our models. Now, let's delve into another type of drift – data drift.

Data drift

Data drift, sometimes referred to as **feature drift**, is another phenomenon we need to keep our eye on. Let's imagine a scene. Your ML model is a ship, and the data it's trained on is the ocean. Now, as we know, the ocean is not a static entity – currents shift, tides rise and fall, and new islands may even emerge. Just as a skilled captain navigates these changes, our models need to adapt to the changing currents of data.

But what exactly does data drift entail? In essence, it's a change in the model's input data distribution. For example, consider an e-commerce recommendation system. Suppose a new product is introduced, and it quickly becomes a hit among consumers. People start using a new term to refer to this product in their reviews and feedback. If your model doesn't adapt to include this new term in its understanding, it's going to miss a significant aspect of current customer sentiment and preferences – classic data drift.

Data drift is a reminder that the world our models operate in is not static. Trends emerge, customer behaviors evolve, and new information becomes relevant. As such, it's vital for our models to stay agile and responsive to these changes.

Another kind of drift is **label drift**, and this is when there's a shift in the actual label distribution. Let's consider a customer service bot again. If there's a change in customer behavior, such as from asking for returns to enquiring about the status of their returns, the distribution of labels in your data shifts, leading to label drift.

Now that we've demystified model and data drift, let's delve into their various sources to better understand and mitigate them. When we talk about drift in the context of ML, we typically distinguish between two main sources: model drift and data drift. It's a bit like considering the source of changes in the taste of a dish – is it the ingredients that have changed or the chef's technique?

Sources of model drift

Model drift occurs when the underlying assumptions of our model change. This is akin to a chef changing their technique. Maybe the oven's temperature has been altered, or the baking time has been modified. In the ML world, this could be due to changes in the environment where the model is deployed. A good example is a traffic prediction model. Let's say the model was trained on data before a major roadway was constructed. After the construction, traffic patterns change, leading to model drift as the underlying assumptions no longer hold.

Sources of data drift

Data drift, on the other hand, is driven by changes in the statistical properties of the model inputs over time. This is like the ingredients in our dish changing. For instance, if a seasonal fruit that's typically part of our recipe is no longer available and we have to replace it, the taste of the dish might drift from its original flavor. In the realm of ML, an example could be an SA model that fails to account for the emergence of new slang terms or emojis, leading to a drift in the data the model is analyzing.

Understanding the sources of drift is essential because it allows us to develop strategies for monitoring and mitigating these changes, ensuring that our models stay fresh and relevant in the ever-evolving real world. In the next sections, we'll explore some strategies for managing these sources of drift.

Mitigating drift

The world of ML is ever-evolving, making it crucial for us to remain adaptable. We've seen how the concept of drift is integral to understanding changes in our data or model over time. But what can we do when faced with these shifting sands? Are we merely left to witness the disintegration of our model's performance? Not quite. This section presents actionable strategies for mitigating drift, each one holding its unique place in our toolbox for effective drift management.

Understanding the context

Before we delve into the technicalities of mitigating drift, let's acknowledge the necessity of understanding the context in which our model operates. Just as a ship captain needs to understand the sea and the weather conditions, we need to comprehend our data sources, user behavior, environmental changes, and all other nuances that form the backdrop against which our model functions.

Consider an e-commerce recommendation system. Understanding the context would mean being aware of seasonal trends, ongoing sales, or any recent global events that could influence customer behavior. For instance, during a global sporting event, there might be a surge in sports-related purchases. Being aware of these contextual cues can help us preempt drift and prepare our models to adapt.

Continuous monitoring

Knowledge without action is futile. Once we're familiar with our context, the next step is to keep a vigilant eye on our model's performance. We need to continuously monitor the heartbeat of our models and data. This could be achieved by tracking model performance metrics over time or using statistical tests to identify significant shifts in data distributions.

Take the case of a credit scoring model. If we notice a sudden surge in the number of credit defaults, it might indicate a drift that needs our attention. Monitoring systems such as dashboards with real-time updates can prove to be valuable assets in catching these shifts before they snowball into more significant problems.

Regular model retraining

Stagnation is the enemy of progress. As the world around us changes, our models need to keep up by learning from fresh data. Regularly retraining the model can help it stay updated with recent trends and patterns. How often should we retrain? Well, it depends on the velocity of change in our data or context. In some cases, retraining may be necessary every few months, while in others, it might be required every few days.

Consider a model predicting stock market trends. Given the volatility of the markets, the model might benefit from daily or even hourly retraining. Conversely, a model predicting housing prices might only need semi-annual retraining due to the relative stability of the housing market.

Implementing feedback systems

Feedback isn't just for Amazon reviews or post-workshop surveys. In the world of ML, feedback systems can act as our reality checks. They can help us understand whether our model's predictions align with the evolving real world. Feedback can come from various sources, such as users flagging incorrect predictions or automated checks on model outputs.

Suppose we've deployed an ML model for SA in social media posts. We could set up a feedback system where users can report when the model incorrectly labels their posts' sentiment. This information can help us identify any drift in language use and update our model accordingly.

Model adaptation techniques

We're now stepping into the more advanced territory of drift mitigation. Techniques such as online learning, where the model learns incrementally from a stream of data, or drift detection algorithms, which alert when the data distribution has significantly deviated, can automatically adapt the model to mitigate drift.

Despite their apparent appeal, these techniques come with their set of challenges, including computational cost and the need for continuous data streams. They're like the high-tech equipment in our toolkit – powerful, but requiring expert handling. For example, an algorithm trading in the stock market might employ online learning to instantly react to market trends, demonstrating the power of model adaptation techniques when appropriately utilized.

In conclusion, mitigating drift is not a one-size-fits-all solution. It's a layered approach where each strategy plays a critical role, much like the gears in a watch. Understanding the context sets the stage, continuous monitoring keeps us alert, regular retraining ensures our model remains relevant, feedback systems provide a reality check, and model adaptation techniques allow for automatic adjustments. The key lies in understanding which strategies to implement when, giving us the power to ensure our models' longevity despite the ever-present drift.

Summary

In the ever-evolving domain of ML, confronting the dual challenges of algorithmic bias and model/data drift is not just about immediate fixes but also about establishing enduring practices. The strategies

delineated in this chapter are critical steps toward more equitable and adaptable ML models. They are the very embodiment of vigilance and adaptability that ensure the integrity and applicability of AI in the face of data's dynamic nature.

As we turn the page from confronting biases and drifts, we enter the expansive realm of AI governance. Our next chapter will focus on structuring robust governance mechanisms that do not merely react to issues but proactively shape the development and deployment of AI systems. The principles of governance – encompassing the stewardship of data, the responsibility of ML, and the strategic oversight of architecture – are not just tactical elements; they are the backbone of ethical, sustainable, and effective AI deployment.

AI Governance

In the era of technology that we currently reside in, disruptive innovations are largely fueled by data, analytics, and AI. These elements are forging new paths across multiple sectors, birthing fresh avenues for revenue, and redefining corporate management paradigms. Consider the projection by McKinsey & Co., which forecasts an addition of over \$15 trillion in business value by 2030, solely attributed to analytics and AI. In recognizing this goldmine, global organizations are fervently channeling resources to establish a strong foothold in this data-driven domain. A 2022 survey by NewVantage Partners supports this notion and indicates that a staggering 97% of surveyed entities are amplifying their investments in data-centric and AI initiatives.

However, a significant paradox emerges here. Despite the enormous capital influx, the majority of organizations grapple with extracting tangible benefits from their data endeavors. So, what seems to be the impediment? The core challenge often resides in not having a well-structured, actionable data governance blueprint encompassing diverse data applications from business intelligence to machine learning. Gartner's analysis offers a more somber outlook. They speculate that, by 2025, about 80% of enterprises aiming to expand their digital footprint might falter primarily due to their outdated approach to data analytics governance.

Navigating the realms of data governance is paramount, yet it presents a convoluted maze for many diving into the digital transformation journey. Data teams often find themselves wrestling with the following areas:

- Harmonizing a tidal wave of data that originates from diverse sources and dismantling entrenched data silos
- Upholding impeccable data standards
- Ensuring that reliable data is both accessible and easily locatable
- Strategically gating data access
- Achieving transparency in data utilization and consumption patterns
- Facilitating secure data exchanges, both internally and externally
- Overseeing machine learning procedures
- Adhering to rigorous regulatory mandates

A shrewd data governance blueprint is the key to untangling these complexities, enabling organizations to truly harness the potential of their data assets.

In this digital age, where every operational facet is intertwined with technology, a holistic understanding of governance is indispensable. The narrative of governance within the digital transformation spectrum

isn't confined to data alone but spans across three pivotal dimensions: **data governance**, **machine learning (ML) governance**, and **architectural governance**. Each dimension carries its own weight, yet together, they provide a comprehensive framework and ensure that organizations exploit technological assets to their fullest potential:

- **Data governance**: This focuses on the management and quality of data throughout its life cycle. It ensures that data is trustworthy, reliable, and usable. With the proliferation of data sources and the sheer volume of data being generated, it becomes paramount to have clear protocols for data acquisition, storage, access, and disposal. This form of governance lays the foundation, ensuring the integrity, security, and availability of data.
- **ML governance**: This addresses the specific challenges associated with the deployment and operation of AI and ML models. Given the transformative power of ML in modern business processes, it's vital to ensure these models are transparent and ethical and operate within predefined boundaries. ML governance deals with model development, deployment, monitoring, and continuous improvement, ensuring that they are both effective and ethically sound.
- **Architectural governance**: This ensures that the IT landscape aligns well with the business objectives. It is one thing to have the right data or ML models, but ensuring that the underlying infrastructure, applications, and the interconnections between them are designed and operated optimally is another factor. This ensures scalability, resilience, and efficiency in the digital ecosystem.

Now, why is it crucial to discuss all three together? Simply put, they are intrinsically linked. Data is the foundation upon which ML models are built, and the architecture is the environment in which both data and ML models reside. Ignoring one aspect can compromise the effectiveness of the others. For instance, without proper data governance, even the most sophisticated ML models might produce unreliable results. Similarly, without robust architectural governance, data might be stored inefficiently or insecurely, affecting both standard analytics and advanced ML operations.

Together, these three pillars of governance create a cohesive, comprehensive strategy; addressing them in tandem ensures that the machinery of an organization operates smoothly, ethically, and effectively, turning the promise of digital transformation into a tangible reality.

Let's go through the topics we'll cover in this chapter:

- Introduction to the COMPAS dataset case study
- Text embeddings using pre-trained models and OpenAI

Mastering data governance

For countless organizations, data stands as a priceless treasure. Yet, it's data governance that serves as the compass guiding one to extract the true worth of data. Envision data governance as a holistic amalgamation of principles, methodologies, and tools designed to oversee the entire life cycle of your data, ensuring it's in sync with the broader business roadmap. A well-orchestrated data governance blueprint bestows data teams with unmatched data stewardship and transparency and audit trails of data interaction patterns throughout the enterprise. Rolling out a robust data governance regime not only safeguards data against unsanctioned access but also institutes compliance protocols as per regulatory

benchmarks. By astutely playing their data governance cards, numerous entities have used this approach to gain a pivotal competitive edge, enhancing customer confidence, fortifying data and privacy norms, and shielding their invaluable data resources.

Current hurdles in data governance

Crafting an impeccable data governance strategy in today's era is akin to navigating a labyrinth, especially given the modern data collection and analysis dynamics. As firms amass vast reservoirs of data, which are predominantly unstructured, the lion's share ends up in cloud-based data lakes such as AWS S3, Azure ADLS, or **Google Cloud Storage (GCS)**. In order to put this into perspective, IDC projections suggest that by 2025, a staggering 80% of organizational data will be unstructured. Yet, it's within this chaotic data landscape that AI's goldmine lies. Selected segments of this unstructured trove are occasionally transported to a data warehouse, structured for business intelligence endeavors, and sometimes retraced. This cyclic movement births isolated data pockets, each governed by distinct protocols.

For instance, within data lakes, the focus is predominantly on file and directory access rights. Conversely, in data warehouses, attention shifts to permissions at the granularity of tables, columns, and rows. An alteration in one landscape seldom reflects in the other. Governance, being executed at divergent echelons across these domains, lacks uniformity. Furthermore, the tools employed on top of these platforms vary drastically, stymieing seamless team collaboration. This translates to a governance approach that's sporadic and error-prone, complicating permission allocation and audits, as well as data discovery or sharing.

However, data isn't merely confined to files or tables. In today's age, we also contend with evolving data forms such as dashboards, ML models, and notebooks. Each comes bundled with unique permission paradigms, adding layers of complexity to uniform access right management. This challenge is magnified when data assets sprawl across diverse cloud platforms, each with its unique access governance solution.

What is the crux? The more multifaceted your data architectural landscape becomes, the more daunting and resource-intensive it is to master data governance. Let's dive into a few specific aspects of data stewardship to help ground our understanding.

Data management: crafting the bedrock

At the heart of a compelling data governance strategy lies proficient data management. This sphere encompasses the meticulous aggregation, fusion, orchestration, and retention of trustworthy datasets, acting as a catalyst for businesses to harness maximum value. With the contemporary business

landscape rapidly evolving, an organization's merit hinges significantly on its prowess in extracting insights from the wealth of data it stewards. Moreover, data management plays a pivotal role in furnishing organizations with insights on data interaction frequency, alongside offering a suite of tools tailored for holistic data life cycle oversight.

Historically, the bastion of analytics-driven data management has been the data warehouse, typified by tabular data that is managed via structures such as tables and views comprising rows and columns. Conversely, data lakes stand as reservoirs for an eclectic mix of structured or unstructured data tailored to data science or ML pursuits. From raw text files and Apache Parquet formats to multimedia content, such as images or videos, data lakes manage this myriad of datasets at the granular file echelon.

Enter the realm of the data lakehouse—a new paradigm where organizations can stockpile data singularly, making it accessible for a spectrum of analytical applications. This innovative approach curtails data redundancy and pares down the data management ambit for organizations.

Data ingestion – the gateway to information

Before data finds its permanent abode for subsequent utilization, it undergoes the critical phase of collection. While data sources are myriad, the principal conduits include the following:

- Cloud-based storage systems
- Message relay channels
- Traditional relational databases
- APIs of **software as a service (SaaS)** platforms

Lately, a significant chunk of data has been sourced from files relayed to object storage facilities that are integral to public cloud service providers. These files, ranging from a handful to millions (daily), encapsulate a diverse array of formats:

- Unstructured entities such as PDFs, audio files, or videos
- Semi-structured formats such as JSON
- Structured types, including Parquet and Avro

For those venturing into the realm of data streaming, distributed message queues, such as Apache Kafka, emerge as the go-to platform. This seamless integration paves the way for ultra-fast message processing in a linear sequence. By complementing open source queue systems, each marquee cloud provider boasts its native message relay service.

Data integration – from collection to delivery

Data, in its raw form, originates from a myriad of sources, demanding integration to unlock its full potential. The integration might unfurl in batch sequences or be streamed in real time, aiming for prompt insight derivation. But integration isn't a standalone task—it demands orchestrated actions to curate, ingest, amalgamate, reshape, and finally, disseminate the refined data.

Data warehouses and entity resolution

A weapon in the arsenal of data warehouses is the capacity for **entity resolution (ER)**. At its core, ER involves deciphering real-world entities from data representations. These entities might be individuals, products, or locations. This technique is pivotal in master data management and data mart dimension alignment. For instance, discerning that “Bill Wallace” and “William Wallace” from disparate systems are one and the same requires intricate algorithm applications using expansive data graphs.

The quest for data quality

In order to exploit the richness of data, trust is paramount. Data of questionable quality can spur analytical inaccuracies and subpar decision-making, and inflate operational costs. According to Gartner, such data anomalies can bleed organizations of a staggering \$12.9 million annually. A robust data governance blueprint, therefore, mandates an unwavering spotlight on data quality.

Documentation and cataloging – the unsung heroes of governance

Metadata isn't just a side aspect of ML governance; it's central. ML catalogs offer insights, guiding users on available resources and their potential applications. This shared knowledge repository accelerates model deployment, promoting best practices across the board.

When assessing data quality, several facets come under scrutiny:

- Is the data impeccable and exhaustive?
- What is its origin?
- How current is the data?
- Does the data flout any quality standards?

If you can auto-capture data lineage, this can facilitate a swift grasp of data ownership and genesis. This lineage isn't just a backward trace; it projects forward, showcasing the entities that consume this data—be it other tables, dashboards, or notebooks.

Moreover, comprehending a dataset's lineage isn't sufficient. Grasping data integrity within that dataset is equally vital. Running real-time quality checks and aggregating these checks for easy access and monitoring is pivotal for ensuring pristine data quality for subsequent analytical tasks. They preempt the influx of flawed data, validate data quality, and instigate policies to counter anomalies. Monitoring the trajectory of data quality can be quite cumbersome but will offer keen insights into data evolution and any areas warranting intervention.

Understanding the path of data

Modern organizations grapple with an avalanche of data from diverse origins. Grasping where this data originates, as well as its consumption patterns, is crucial for assuring its quality and reliability. This is where data lineage emerges as a potent instrument, enabling a clearer overview of data within organizations. Essentially, data lineage maps the journey of data, from its inception to its end use. This mapping involves capturing exhaustive metadata and pertinent events throughout the data's life cycle. It encompasses details such as data origin, the datasets employed in its creation, its creators, any associated transformations, timestamps, and much more. By adopting data lineage solutions, teams can visualize the entire spectrum and flow of data transformations across their infrastructure.

As the wave of data-driven decision-making swells, embedding data lineage becomes a cornerstone of robust data governance.

Regulatory compliance and audit preparedness

Numerous regulatory frameworks, such as GDPR, CCPA, HIPAA, BCBS 239, and SOX, mandate organizations to maintain transparency over their data streams. They must certify that the data or reports shared stem from verifiable, trustworthy sources. This necessitates the tracing of tables and datasets utilized in reports, emphasizing data traceability within an organization's data architecture. Data lineage eases compliance burdens by automating the creation of data flow trails for audit purposes.

Change management and impact analysis

Data isn't static; it evolves. Comprehending the cascading effects of data modifications on subsequent users is pivotal. Through data lineage, teams can discern all downstream entities affected by data alterations. This visibility extends to applications, dashboards, ML models, datasets, and more. These visualization and analysis tools aid in gauging any potential fallout, enabling timely stakeholder notifications. Furthermore, it facilitates IT units in their transparent sharing of data migration updates, ensuring unhindered business operations.

Upholding data quality

Ensuring the quality of data is paramount in any data-centric initiative. The integrity, accuracy, and reliability of data directly influence the insights derived, models trained, and decisions made. Data lineage empowers data users, such as data engineers, scientists, and analysts, by providing them with a comprehensive understanding of the data's origins and transformations. This contextual awareness results in improved analytical outcomes, as users can confidently trace data back to its source and understand any modifications made to it.

EXAMPLE AND SCENARIO

Example: Synthetic labeling is an innovative technique wherein ML models, especially large language models, are employed to assist in labeling data. This method is particularly beneficial for vast datasets where manual labeling would be time-consuming and impractical.

Scenario: Imagine a dataset containing millions of customer feedback comments. Instead of having human annotators read through each comment and label it as "positive," "negative," or "neutral," a trained language model can be used to process and label these comments quickly.

However, this method is not without challenges. Synthetic labeling can introduce biases or errors, particularly if the model itself was trained on skewed or imperfect data. If unchecked, these biases can propagate through the dataset, leading to misleading insights or flawed ML models.

This is where the role of data custodians becomes pivotal. They must actively monitor and validate synthetic labels to ensure they maintain a high standard of accuracy. It's essential to combine the efficiency of synthetic labeling with human oversight, ensuring that the generated labels are cross-checked, verified, and refined when necessary. By striking this balance, organizations can harness the benefits of rapid labeling while upholding stringent data quality standards.

Troubleshooting and analysis

Regardless of stringent checks, data processes can falter. Here, data lineage shines by assisting teams in pinpointing the origin of errors in their systems, whether in data pipelines, applications, or models. Such pinpoint accuracy drastically trims down the debugging duration, translating to immense time and effort savings.

In summary, the systematic mapping of the journey of data, or data lineage, is not just a luxury; it's a necessity in today's data-rich environments, enabling clearer decision-making, enhanced compliance, and more efficient operations.

Of course, the point of gathering this data is often to train an ML model, so let's turn our attention toward how to think about ML governance.

Navigating the intricacy and the anatomy of ML governance

ML doesn't operate solely by using algorithms and the data they ingest. Instead, its essence lies in constructing models responsibly, a task underpinned by governance. Just as governance has been the bedrock of the realm of data, it's equally crucial for ML, especially in aspects such as accountability, standardization, compliance, quality, and clarity. Let's discuss this topic in greater detail in the following sections.

ML governance pillars

Unlocking ML's potential is rooted in ensuring that models meet the following criteria:

- Aligns with relevant regulatory and ethical benchmarks
- Exhibits consistent outcomes and performance
- Illuminates their development and implications in a transparent way
- Can undergo regular quality assessments and updates
- Adheres to standard documentation and cataloging protocols

While adherence to industry-specific regulations sets the baseline, the navigation of the broader spectrum of ethical concerns often requires a nuanced approach. The essence of governance extends beyond mere legalities, delving into the realm of what is morally right. Here, proactive assessments and a robust evaluation process come into play, ensuring models aren't just compliant but are also ethically sound.

Model interpretability

In the realm of ML, understanding how a model arrives at its decisions is crucial. This is not just for academic interest but has significant real-world implications, especially when decisions impact human lives, such as in healthcare or criminal justice.

Consider a healthcare system where ML models predict the likelihood of patients developing certain diseases based on their medical records. A model might predict that a patient has a high risk of developing diabetes. But why did the model make this prediction? Is it due to the patient's age, genetic history, dietary habits, or some other factor?

Local interpretable model-agnostic explanations (LIME) is a tool that was developed to shed light on this black box of ML predictions. It works by perturbing the input data slightly (adding some noise) and observing how these changes affect the model's predictions. By doing this many times, LIME builds up a picture of which input variables are most influential regarding a given prediction.

However, as with any tool, LIME is not infallible. It provides approximations of model behavior, not exact explanations. Furthermore, its effectiveness can vary depending on the model and data at hand.

This highlights the necessity of ML governance. Ensuring that tools such as LIME are used correctly, understanding their limitations, and supplementing them with other methods when necessary are pivotal. It's not enough for a model to be accurate; it must also be transparent and its decisions explainable, especially in high-stakes situations. ML governance policies can set standards for interpretability and guide the proper use and interpretation of tools such as LIME.

For example, the following code segment demonstrates how to leverage the LIME tool by using a sentiment analysis model sourced from the Model Hub of Hugging Face. While this script provides an interpretability layer to the model by identifying influential words/features in the input, it's imperative to understand that such interpretations provide approximations. The highlighted words can give insights into the model's decisions, but they might not capture the entirety of the model's complex reasoning. Hence, while tools such as LIME are valuable, they should be employed judiciously within a broader framework of ML governance to ensure that the insights they offer are actionable and reliable:

```
# Import required modules
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
from lime.lime_text import LimeTextExplainer
# Load the tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")
model = AutoModelForSequenceClassification.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment")
# Define the prediction function for LIME
def predictor(texts):
    inputs = tokenizer(texts, return_tensors="pt", truncation=True, padding=True, max_length=512)
    outputs = model(**inputs)
    probs = torch.nn.functional.softmax(outputs.logits, dim=-1).detach().numpy()
    return probs
# Initialize LIME's text explainer
explainer = LimeTextExplainer(class_names=['negative', 'neutral', 'positive'])
```

We can then see the visual interpretation (*Figure 14.1*) of which tokens impact the final decision the most:

```
# Sample tweet to explain
tweet = "I love using the new feature! So helpful."
# Generate the explanation
exp = explainer.explain_instance(tweet, predictor, num_features=5, top_labels=3)
exp.show_in_notebook()
```

In this case, **num_features** determines how many features the explainer should use to describe the prediction. For instance, if it is set to 4, LIME will provide explanations using up to four features that influence the prediction the most. It helps to simplify the explanation by focusing only on the top n features that have the most influence on the prediction rather than considering all features. **top_labels** determines how many of the most probable labels you'd like explanations for, and this is usually for

multi-class classification. For instance, if you set it to 1, LIME will only explain the most probable label. If set to 2, it will explain the two most probable labels, and so on. *Figure 14.1* shows what the output would be for this case:

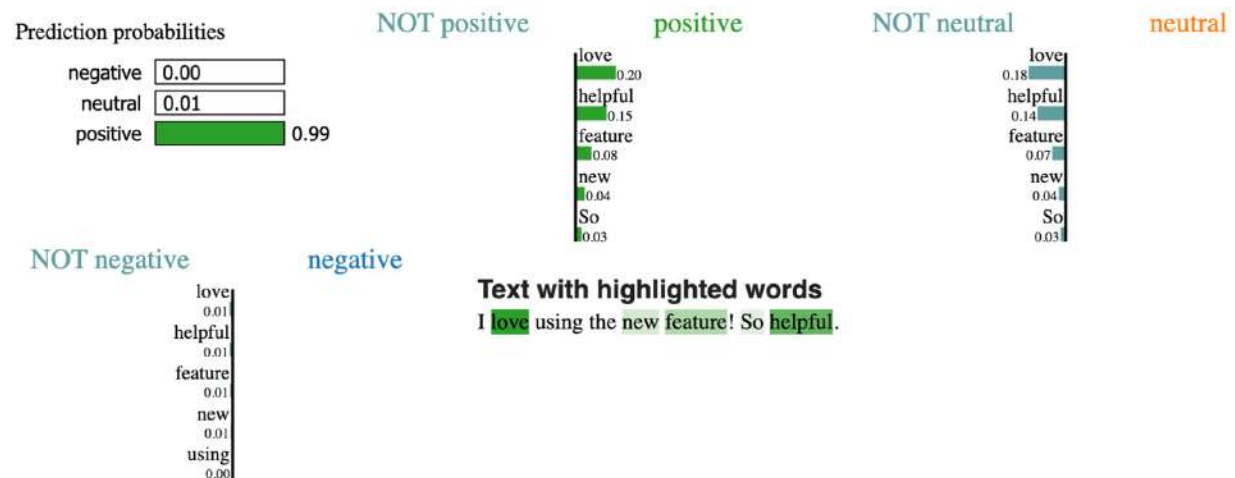


Figure 14.1 – Here, the tokens that most affect the decision are “love,” “helpful,” and, surprisingly, “new” and “feature”

Interestingly, in our example, the tokens **new** and **feature** contribute greatly to the prediction of positive. In terms of governance, this is a great example of *illuminating ML’s development and implications transparently*. Let’s test this out by writing a statement that we believe should be negative:

```
# Sample tweet to explain for negative
tweet = "I hate using the new feature! So annoying."
# Generate the explanation
exp = explainer.explain_instance(tweet, predictor, num_features=5, top_labels=3)
exp.show_in_notebook()
```

We obtain the following results in *Figure 14.2*:

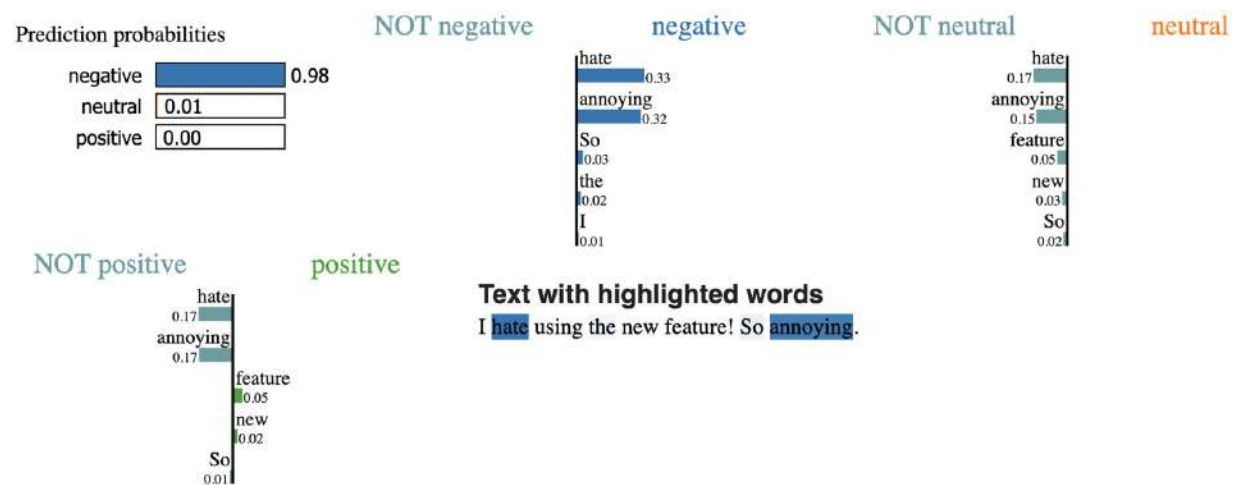


Figure 14.2 – Our sentiment classifier correctly classifies our negative statement

Let’s try one more example where we’d expect a neutral statement:


```
# Sample tweet to explain for neutral (bias at work)
tweet = "I am using the new feature."
# Generate the explanation
exp = explainer.explain_instance(tweet, predictor, num_features=5, top_labels=3)
exp.show_in_notebook()
```

Figure 14.3 shows the resulting graph:



Figure 14.3 – Our neutral statement “I am using the new feature” comes out positive again, thanks to the phrase “new feature”

Fascinating! Our seemingly neutral statement “*I am using the new feature*” comes out mostly positive, and the tokens **new** and **feature** are the reasons why. To show this even further, let’s write one more neutral statement without saying “*new feature*”:

```
# Sample tweet to explain for neutral (bias at work)
tweet = "I am using the old feature."
# Generate the explanation
exp = explainer.explain_instance(tweet, predictor, num_features=5, top_labels=3)
exp.show_in_notebook()
```

Let’s check the results in Figure 14.4:

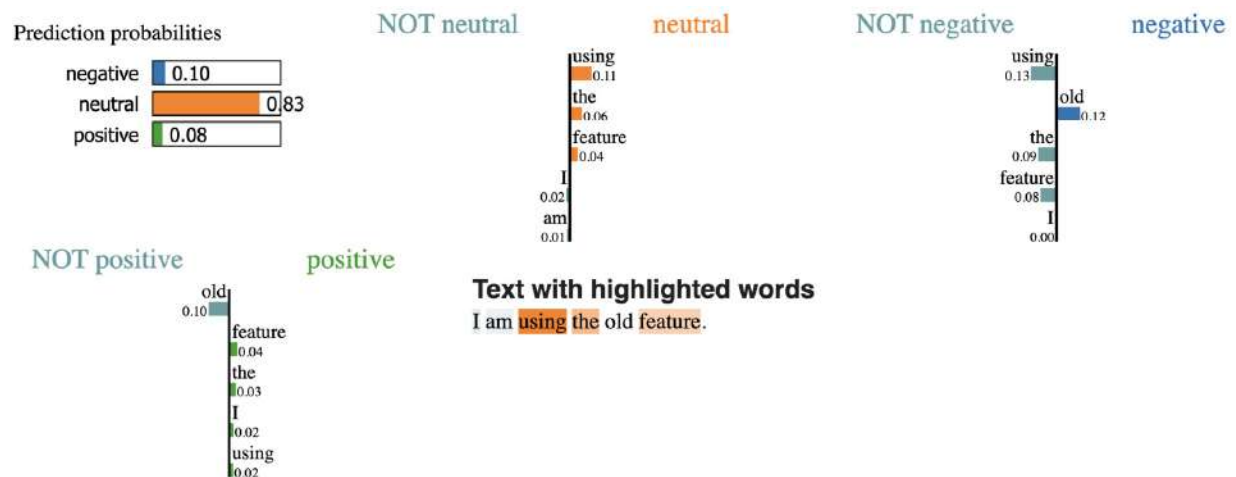


Figure 14.4 – Without “new feature,” the classifier performs as expected

From a governance standpoint, this is an opportunity for classification improvement, and with the help of LIME, we have identified a clear token pattern that leads to inaccuracies. Of course, finding these kinds of patterns one by one can become a tedious process, but sometimes, this is the nature of the iterative development of ML. We can deploy these tools in the cloud to run automatically, but, at the end of the day, the solutions here are to have better quality training data and update our model.

The many facets of ML development

As we’ve already touched on in our ML chapters, there are several dimensions to consider when thinking about ML at any level. Let’s recap some of the aspects of ML development as they relate to data stewardship:

- **Mastering feature engineering:** Features form the bedrock of ML models. Hence, preserving and making them accessible is crucial. Using a feature store not only aids in model development but also provides lineage tracing, ensuring clarity in data transformation and eliminating potential discrepancies between training and real-time applications.
- **Perfecting data handling:** Often, the cornerstone data for model training gets misplaced, complicating the recreation of model parameters. However, with tools such as Managed MLflow, datasets are meticulously logged, ensuring a seamless ML model development cycle.
- **Refining model training:** The journey from ideation to production in ML is seldom straightforward. Model selection involves rigorous evaluations, methodological considerations, and constant fine-tuning. Using platforms such as MLflow, each iteration (and the associated metrics) are captured, ensuring a transparent model training process.

Beyond training – model deployment and monitoring

Ensuring model accuracy doesn’t end with deployment; it demands continuous oversight, especially as models adapt to real-world scenarios. Monitoring encompasses several aspects:

- **Concept drift:** Real-world variables, such as market shifts or evolving business strategies, can drastically affect model outcomes.
- **Data adjustments:** While deliberate data changes might be easy to track, inadvertent shifts in data collection or representation can introduce model inconsistencies.
- **Bias:** Beyond statistical imbalance, bias can manifest as the unequal treatment of distinct groups, necessitating rigorous checks against potential disparities. We have already seen how techniques such as synthetic labeling can lead to biases bubbling up to the surface.

For successful ML governance, establishing performance thresholds, monitoring frequencies, and using problem-alert procedures is pivotal. Many companies offer an ecosystem of tools, from automated dashboards and lineage tracking to data quality checks, ensuring models remain accurate, unbiased, and compliant.

A guide to architectural governance

Architectural governance is the cornerstone of ensuring the seamless integration of IT infrastructure, and it supports core business processes. Its principal objectives encompass the following:

- Cataloging current architectural layouts
- Establishing guidelines, principles, and benchmarks
- Aligning business and IT visions
- Crafting a target infrastructure blueprint
- Identifying the value proposition of the target framework
- Highlighting disparities between the current and desired architecture
- Crafting a comprehensive architectural roadmap

The five pillars of architectural governance

With these five pillars in mind, we can also turn to some principles on how to modernize your governance architecture and optimize its effectiveness.

- **Consistency:** Ensuring harmonious and integrated workflows without hitches
- **Security:** This is paramount for protecting sensitive data and upholding regulatory compliance
- **Scalability:** A forward-looking approach, taking into account the growing data needs
- **Standardization:** Embracing universally accepted standards for flexibility and interoperability
- **Reuse:** Promoting efficiency by creating and reutilizing components across the architecture

Transformative architectural principles

In order to maximize the benefits of governance, certain foundational principles must be integrated:

- **Delivering trusted data as products:** Data isn't merely a by-product; it is a resource, necessitating a product-centric mindset
- **Prioritizing self-service accessibility:** This involves breaking down bureaucratic data barriers to allow more agile decision-making processes
- **Democratizing data value creation:** Decentralizing data access ensures a more informed, data-driven organizational strategy
- **Eliminating data silos:** A unified data strategy streamlines processes and prevents redundancy and discrepancies

Zooming in on architectural dimensions

Let's take another look at our five pillars of governance:

- **Consistency:** Integrating diverse workloads, use cases, and platform components for a unified, coherent experience
- **Security:** Championing data protection through robust access controls, especially for sensitive data types

- **Scalability:** As a cloud-native, **platform as a service (PaaS)** solution, it effortlessly scales, adapting to varying computational demands
- **Standardization:** By anchoring onto open source formats, such as Parquet, this prevents vendor lock-in and promotes interoperability
- **Reuse:** The platform's integrative approach with popular CI/CD tools ensures that any created components can be reused, ensuring efficiency

Summary

In our journey through the realms of data, ML, and architectural governance, we've underscored the paramount importance of a cohesive strategy for organizations to navigate the digital age effectively. The era we currently inhabit is characterized by its rapid technological advancements, primarily propelled by the unparalleled power of data and analytics. In order to harness this power, organizations must have a clear blueprint—a blueprint that's defined, structured, and actionable.

At the heart of this governance lies the principle of consistency. A uniform approach ensures that data from varied sources can be integrated seamlessly, enhancing the overall decision-making process. Equally crucial is the principle of security. With increasing threats in the digital domain, securing data assets is no longer optional but is mandatory for any forward-thinking organization. Furthermore, the dimension of scalability becomes essential, especially when we consider the exponential growth of data. Organizations need to be prepared not just for their present data requirements but also for future demands that might arise.

Yet, these foundational elements are just the beginning. For data governance to be truly transformative, it must democratize access to data, ensuring that insights are not restricted to a select few but are available across the organization. This broad access, however, must not come at the cost of creating data silos, which can stifle innovation and hinder cross-functional collaboration.

As we conclude, it's evident that the journey to optimal data governance is multi faceted. But with the right principles in place, organizations are better poised to unlock the vast potential that their data assets hold, heralding a new age of informed decision-making and strategic innovation.

Navigating Real-World Data Science Case Studies in Action

Kudos to you, diligent reader! Here we are, deep within the intricate tapestry of data science, having traversed its vast expanse together. Your journey to [Chapter 15](#) showcases not just your commitment but also your robust intellectual curiosity in the transformative realm of data. It's truly a noteworthy milestone.

In this chapter, we will unravel two meticulously selected case studies that provide a tangible insights into the pragmatic dimensions of data science. These in-depth analyses will act as beacons, illuminating the theoretical principles we've previously discussed. However, acknowledging the expansive nature of data science and the myriad scenarios it encompasses, we've made a strategic decision. While we will dissect these two scenarios comprehensively here, there exists a treasure trove of additional case studies awaiting your exploration in our book's GitHub repository.

Harnessing the academic and formal tone we've maintained, let's recognize that these case studies aren't merely demonstrations of data methodologies. They represent the intricate dance of challenges, strategies, and triumphs inherent to real-world data applications.

With the compass (pun intended, as you'll see soon) of our previous discussions in hand, let's delve deep into these case studies, shall we?

These are the topics we'll cover in this chapter:

- Introduction to the COMPAS dataset case study
- Text embeddings using pre-trained models and OpenAI

Introduction to the COMPAS dataset case study

In the realm of machine learning, where data drives decision-making, the line between algorithmic precision and ethical fairness often blurs. The COMPAS dataset, a collection of criminal offenders screened in Broward County, Florida, during 2013-2014, serves as a poignant reminder of this intricate dance. While, on the surface, it might appear as a straightforward binary classification task, the implications ripple far beyond simple predictions. Each row and feature isn't just a digit or a class; it represents years, if not decades, of human experiences, ambitions, and lives. As we dive into this case study, we are reminded that these aren't mere rows and columns but people with aspirations, dreams, and challenges. With a primary focus on predicting recidivism (the likelihood of an offender to re-offend), we're confronted with not just the challenge of achieving model accuracy but also the

monumental responsibility of ensuring fairness. Systemic privilege, racial discrepancies, and inherent biases in the data further accentuate the need for an approach that recognizes and mitigates these imbalances. This case study endeavors to navigate these tumultuous waters, offering insights into the biases present, and more importantly, exploring ways to strike a balance between ML accuracy and the paramount importance of human fairness. Let's embark on this journey, bearing in mind the weight of the decisions we make and the profound impact they hold in real-world scenarios.

The core of this exploration revolves around the **Correctional Offender Management Profiling for Alternative Sanctions (COMPAS)** dataset. It aggregates data from criminal offenders processed in Broward County, Florida, between 2013–2014. Our focus narrows to a specific subset of this data, dedicated to the *binary classification task of determining the probability of recidivism based on an individual's attributes*.

For those interested, the dataset is accessible here: <https://www.kaggle.com/danofer/compass>.

At first glance, the task seems straightforward. A binary classification with no data gaps, so, why not dive right in? However, the conundrum surfaces when one realizes the profound consequences our ML models can have on real human lives. The mantle of responsibility is upon us, as ML engineers and data practitioners, to not just craft efficient models but also to ensure the outcomes are inherently “just.”

Throughout this case study, we'll endeavor to delineate the multifaceted nature of “fairness.” While multiple definitions are available, the crux lies in discerning which notion of fairness aligns with the specific domain at hand. By unfolding various fairness perspectives, we aim to elucidate their intended interpretations.

NOTE

This case study is illustrative and should not be misconstrued as an exhaustive statistical analysis or a critique of America's criminal justice framework. Rather, it's an endeavor to spotlight potential biases in datasets and champion techniques to enhance fairness in our ML algorithms.

Without further ado, let's dive right into our dataset:

```
import pandas as pd
import numpy as np
compas_data = pd.read_csv('../data/compas-scores-two-years.csv')
compas_data.head()
```

Figure 15.1 shows the first five rows of our dataset:

	id	name	first	last	compas_screening_date	sex	dob	age	age_cat	race	...
0	1	miguel hernandez	miguel	hernandez	2013-08-14	Male	1947-04-18	69	Greater than 45	Other	...
1	3	kevon dixon	kevon	dixon	2013-01-27	Male	1982-01-22	34	25 - 45	African-American	...
2	4	ed philo	ed	philo	2013-04-14	Male	1991-05-14	24	Less than 25	African-American	...
3	5	marcu brown	marcu	brown	2013-01-13	Male	1993-01-21	23	Less than 25	African-American	...
4	6	bouthy pierrelouis	bouthy	pierrelouis	2013-03-26	Male	1973-01-22	43	25 - 45	Other	...

Figure 15.1 – An initial view of the COMPAS dataset

This unveils certain sensitive data concerning individuals previously incarcerated in Broward County, Florida. The key label here is `two_year_recid`, which addresses the binary query: “Did this individual get incarcerated again within 24 months of release?”

The 2016 ProPublica investigation, which scrutinized the fairness of the COMPAS algorithm and its foundational data, placed significant emphasis on the decile scores allotted to each subject. A decile score partitions data into 10 equal segments, similar in concept to percentiles. Essentially, an individual is ranked between 1 and 10, where each score denotes a segment of the population based on a specific metric. To illustrate, a decile score of 3 suggests that 70% of the subjects pose a higher risk of re-offending (scoring between 4 and 10) while 20% pose a lower risk (scoring 1 or 2). Conversely, a score of 7 would indicate that 30% have a heightened recidivism rate (scores of 8-10), whereas 60% are considered at a lower risk (scoring between 1 and 6).

Subsequent analyses showcased certain disparities in decile score allocation, particularly concerning race. Upon evaluating score distribution, clear racial biases emerge, as follows:

```
compas_data.groupby('race')['decile_score'].value_counts(
    normalize=True
).unstack().plot(
    kind='bar', figsize=(20, 7),
    title='Decile Score Histogram by Race', ylabel='% with Decile Score'
)
```

Figure 15.2 shows the resulting graph:

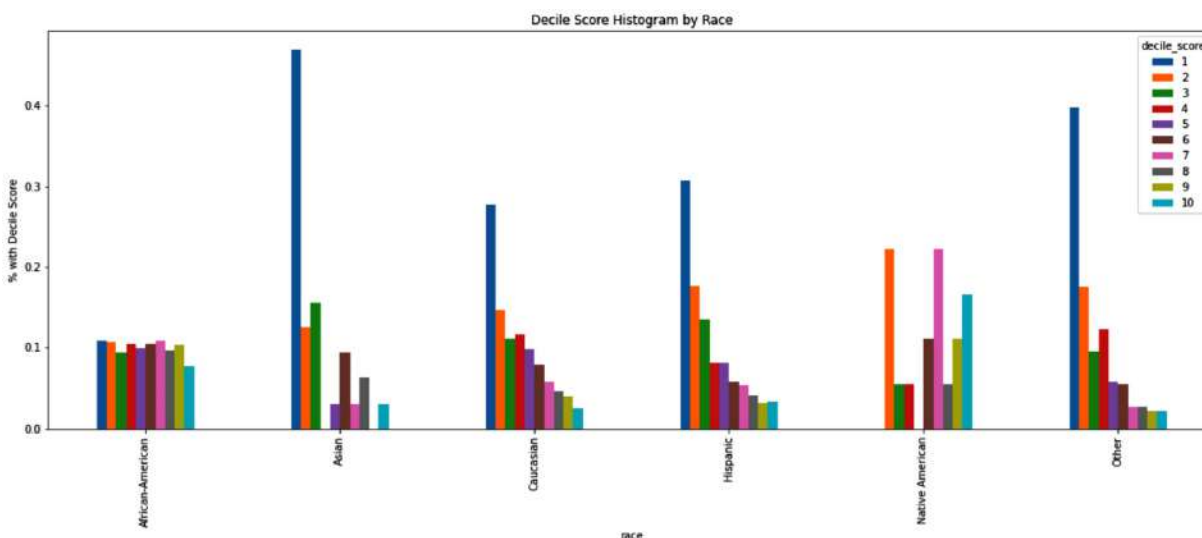


Figure 15.2 – The racial variances in decile score distribution are evident

We could delve deeper into how the ProPublica investigation interpreted its findings, but our interest lies in constructing a binary classifier from the data, setting aside the pre-existing decile scores.

Understanding the task/outlining success

The core of our investigation is binary classification. Our mission can be encapsulated in the question: “Considering various attributes of an individual, can we predict the likelihood of them re-offending, both efficiently and impartially?”

The notion of efficiency is straightforward. We have an arsenal of metrics such as accuracy, precision, and AUC to evaluate model efficacy. But when we discuss “impartiality,” we need to acquaint ourselves with novel concepts and metrics. Before delving into bias and fairness quantification, we should conduct some preliminary data exploration.

Preliminary data exploration

The intention is to predict the `two_year_recid` label using the dataset’s features. Specifically, the features that we’re working with are as follows:

- **sex** – Binary: “Male” or “Female”
- **age** – Numeric value indicating years
- **race** – Categorical
- **juv_fel_count** – Numeric value denoting prior juvenile felonies
- **juv_misd_count** – Numeric value denoting previous juvenile misdemeanors
- **juv_other_count** – Numeric value representing other juvenile convictions

- **priors_count** – Numeric value indicating earlier criminal offenses
- **c_charge_degree** – Binary: “F” indicating felony and “M” indicating misdemeanor

The target variable is as follows:

- **two_year_recid** – Binary, indicating whether the individual re-offended within two years

It’s worth noting that we possess three distinct columns detailing juvenile offenses. We might consider merging these columns into one that represents the total count of juvenile offenses. Given our goal of crafting a precise and unbiased model, let’s inspect the recidivism distribution based on race. By categorizing the dataset by race and analyzing recidivism rates, it’s evident that there are varying baseline rates across racial groups:

```
compas_df.groupby('race')['two_year_recid'].describe()
```

Figure 15.3 shows the resulting matrix of descriptive statistics:

	count	mean	std	min	25%	50%	75%	max
race								
African-American	3696.0	0.514340	0.499862	0.0	0.0	1.0	1.0	1.0
Asian	32.0	0.281250	0.456803	0.0	0.0	0.0	1.0	1.0
Caucasian	2454.0	0.393643	0.488657	0.0	0.0	0.0	1.0	1.0
Hispanic	637.0	0.364207	0.481585	0.0	0.0	0.0	1.0	1.0
Native American	18.0	0.555556	0.511310	0.0	0.0	1.0	1.0	1.0
Other	377.0	0.352785	0.478472	0.0	0.0	0.0	1.0	1.0

Figure 15.3 – Recidivism descriptive statistics categorized by race; distinctive disparities in recidivism rates across racial groups are visible

We also observe limited representation of two racial groups: Asian and Native American. This skewed representation may lead to biased inferences. For context, Asians comprise about 4% of the Broward County, Florida, population, but only about 0.44% of this dataset. In this study, we’ll recategorize individuals from Asian or Native American groups as **other** to address the data imbalances. This allows for a more balanced class distribution:

```
# Modify the race category for educational purposes and to address imbalance in the
dataset
compas_df.loc[compas_df['race'].isin(['Native American', 'Asian']), 'race'] =
'Other' # Adjust "Asian" and "Native American" categories to "Other"
compas_df.groupby('race')['two_year_recid'].value_counts(
    normalize=True
).unstack().plot(
    kind='bar', figsize=(10, 5), title='Recidivism Rates Classified by Race'
) # Visualize Recidivism Rates across the refined racial groups
```

Figure 15.4 shows the resulting bar plot highlighting the differences in recidivism by race:

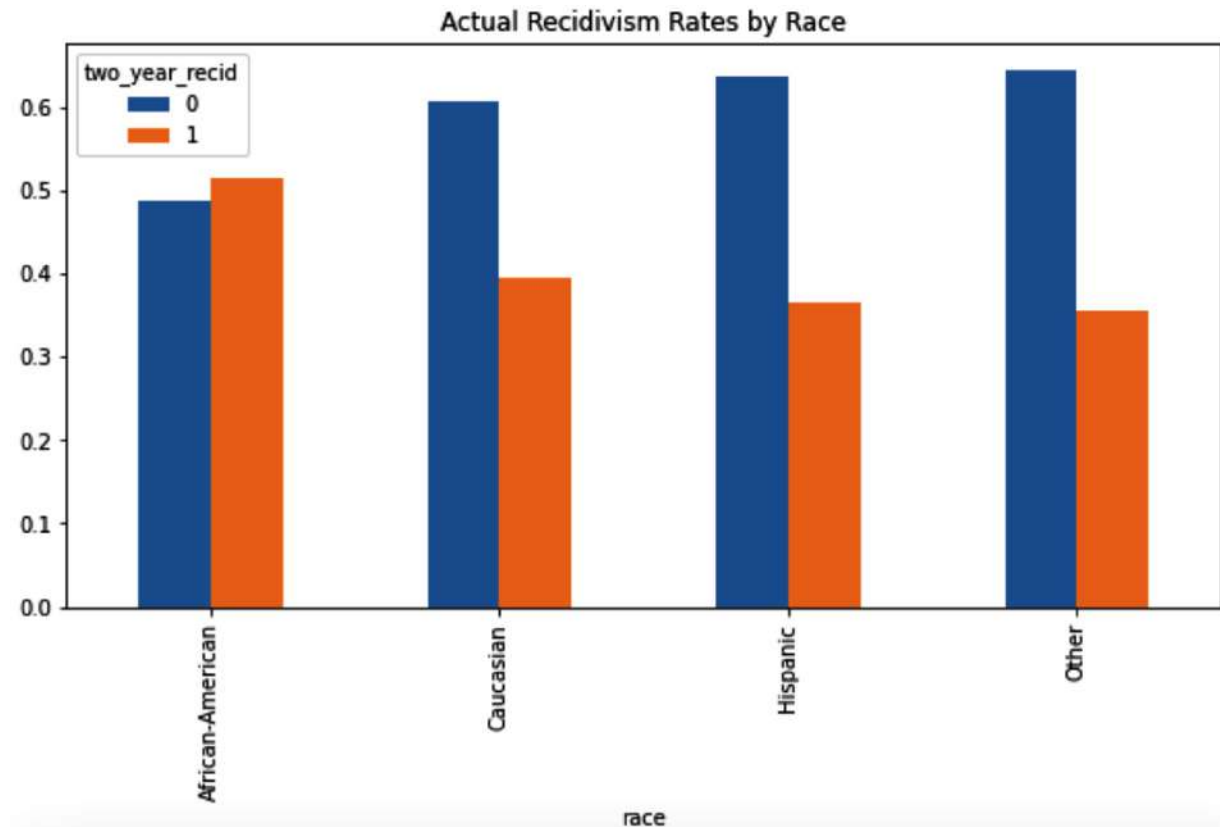


Figure 15.4 – A bar graph illustrating recidivism rates per racial category

Our findings reveal a higher recidivism rate among **African-American** individuals compared to **Caucasian**, **Hispanic**, and **Other** groups. The underlying reasons are multifaceted and beyond this study's scope. However, it's crucial to note the nuanced disparities in rates.

NOTE

We could have analyzed gender biases, as evident differences exist between male and female representations. For this study's objectives, we'll emphasize racial biases.

Advancing further, let's analyze other dataset attributes:

```
compas_df['c_charge_degree'].value_counts(normalize=True).plot(
    kind='bar', title='% of Charge Degree', ylabel='', xlabel='Charge Degree'
)
```

We possess a binary charge degree attribute that, after conversion to a Boolean format, should be readily usable (this plot is shown in *Figure 15.5*):

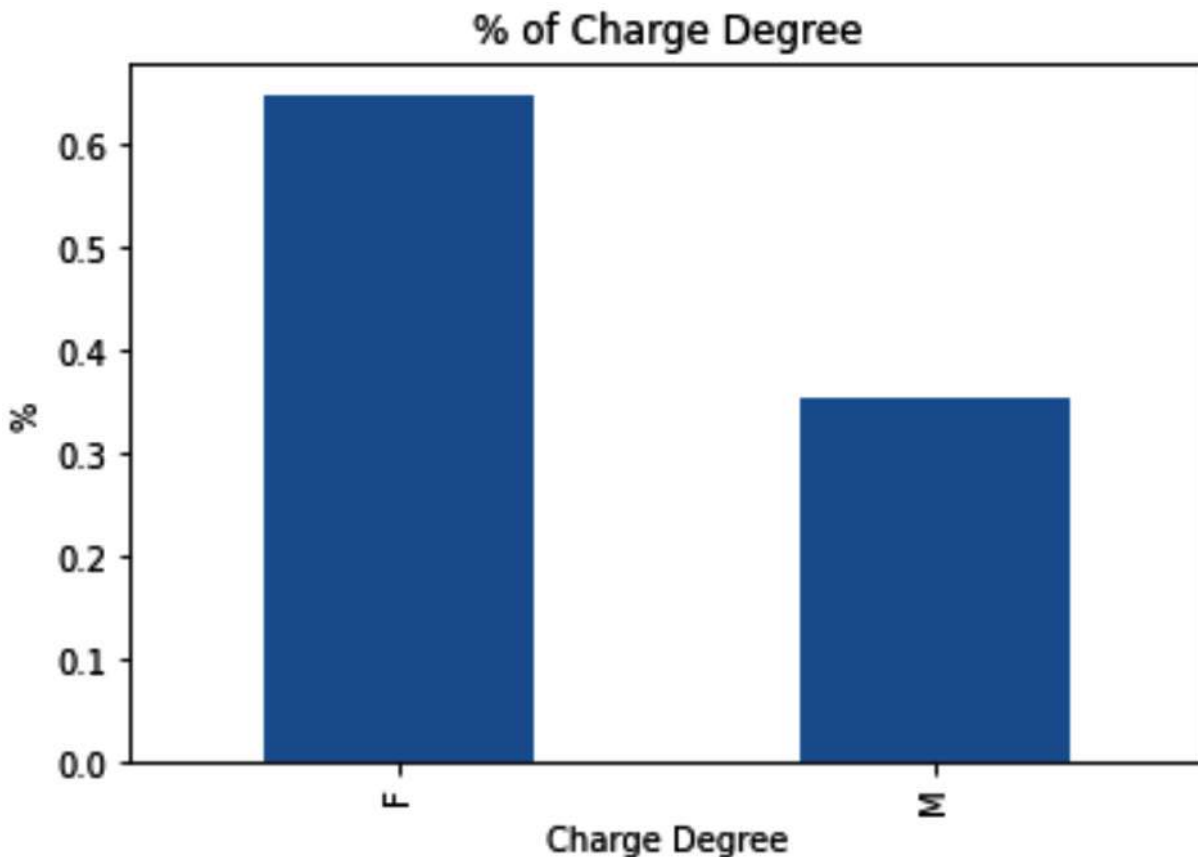


Figure 15.5 – A breakdown of our dataset depicting felonies versus misdemeanors

Approximately 65% of charges are felonies, with the remaining being misdemeanors.

Preparing the data for modeling

Having understood the nuances of the bias and fairness definitions, it's crucial that we give the same attention to our data preparation process. This involves not just the technical transformations but also a thoughtful consideration of the implications of these transformations on fairness.

Feature engineering

We've already touched on a few points during EDA, such as combining the three juvenile crime columns. However, before jumping into that, it's crucial to note that any transformations we apply to our data can introduce or exacerbate biases. Let's take a detailed look.

Combining juvenile crime data

Combining the juvenile offenses into a single feature is logical for model simplicity. However, this can potentially introduce bias if the three types of juvenile crimes have different recidivism implications

based on race. By lumping them together, we could be over-simplifying these implications. Always be wary of such combinations:

```
# feature construction, add up our three juv columns and remove the original features
compas_df['juv_count'] = compas_df[['juv_fel_count', 'juv_misd_count',
'juv_other_count']].sum(axis=1)
compas_df[['juv_fel_count', 'juv_misd_count', 'juv_other_count',
'juv_count']].describe()
```

The resulting matrix can be shown in *Figure 15.6*:

	juv_fel_count	juv_misd_count	juv_other_count	juv_count
count	7214.000000	7214.000000	7214.000000	7214.000000
mean	0.067230	0.090934	0.109371	0.267535
std	0.473972	0.485239	0.501586	0.952763
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	20.000000	13.000000	17.000000	21.000000

Figure 15.6 – A look at our new columns

One-hot encoding categorical features

We need to convert our categorical variables such as `sex`, `race`, and `c_charge_degree` into a numerical format. Here, using a method such as one-hot encoding can be appropriate. However, it's essential to remember that introducing too many binary columns can exacerbate issues in fairness if the model gives undue importance to a particular subgroup:

```
dummies = pd.get_dummies(compas_df[['sex', 'race', 'c_charge_degree']],
drop_first=True)
compas_df = pd.concat([compas_df, dummies], axis=1)
```

Standardizing skewed features

We can easily see that `age` and `priors_count` are right-skewed using the following code block and figure. Standardizing these features can help our model train better. Using methods such as log-transform or square root can be useful:

```
# Right skew on Age
compas_df['age'].plot(
    title='Histogram of Age', kind='hist', xlabel='Age', figsize=(10, 5)
)
# Right skew on Priors as well
compas_df['priors_count'].plot(
    title='Histogram of Priors Count', kind='hist', xlabel='Priors', figsize=(10, 5)
)
```

Figure 15.7 shows us our two distributions and, more importantly, how much our data is skewed:

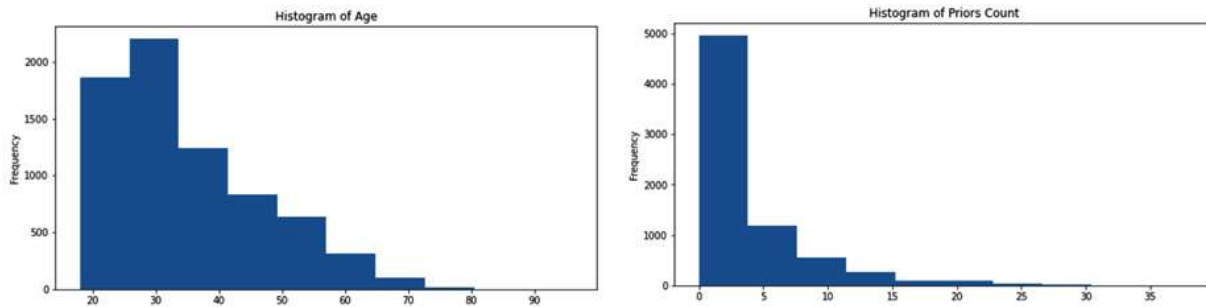


Figure 15.7 – Skewed age and priors data can affect our final predictions

If we want to transform our numerical features, we can use a scikit-learn pipeline to run some feature transformations, such as in the following code block:

```
We can use a scikit-learn pipeline to run a standard scaler like so:
numerical_features = ["age", "priors_count"]
numerical_transformer = Pipeline(steps=[
    ('scale', StandardScaler())
])
```

With a transformer in hand, such as the one defined in the preceding code block, we can begin to address skewed data in real time in our ML pipelines.

Final thoughts

Remember, while we strive to achieve the best model performance, it's crucial to constantly revisit the fairness aspect. Addressing fairness isn't a one-time task but rather an iterative process that involves refining the model, re-evaluating fairness metrics, and ensuring that our model decisions are as impartial as possible. Our ultimate aim is to ensure the equitable treatment of all subgroups while making accurate predictions.

Text embeddings using pretrained models and OpenAI

In the realm of **natural language processing (NLP)**, the quest for effectively converting textual information into mathematical representations, often referred to as embeddings, has always been paramount. Embeddings allow machines to “understand” and process textual content, bridging the gap

between human language and computational tasks. In our previous NLP chapters, we dived deep into the creation of text embeddings and witnessed the transformative power of **large language models (LLMs)** such as BERT in capturing the nuances of language.

Enter OpenAI, a forefront entity in the field of artificial intelligence research. OpenAI has not only made significant contributions to the LLM landscape but has also provided various tools and engines to foster advancements in embedding technology. In this study, we're going to embark on a detailed exploration of text embeddings using OpenAI's offerings.

By embedding paragraphs from this textbook, we'll demonstrate the efficacy of OpenAI's embeddings in answering natural language queries. For instance, a seemingly whimsical question such as "How many horns does a flea have?" can be efficiently addressed by scanning through the embedded paragraphs, showcasing the prowess of semantic search.

Setting up and importing necessary libraries

Before we dive into the heart of this case study, it's essential to set our environment right. We need to ensure we have the appropriate libraries imported for the tasks we'll perform. This case study introduces a couple of new packages:

```
import os
import openai
import numpy as np
from urllib.request import urlopen
from openai.embeddings_utils import get_embedding
from sentence_transformers import util
```

Let's break down our imports:

- **os**: Essential for interacting with the operating system – in our case, to fetch the API key.
- **openai**: The official OpenAI library, which will grant us access to various models and utilities.
- **numpy**: A fundamental package for scientific computing in Python. Helps in manipulating large data and arrays.
- **urlopen**: Enables us to fetch data from URLs, which will be handy when we're sourcing our text data.
- **get_embedding**: A utility from OpenAI to convert text to embeddings.
- **sentence_transformers.util**: Contains helpful utilities for semantic searching, a cornerstone of our case study.

Once our environment is set up, the next step is to configure our connection to the OpenAI service:

```
openai.api_key = os.environ['OPENAI_API_KEY']
ENGINE = 'text-embedding-ada-002'
```

Here, we're sourcing our API key from our environment variables. It's a secure way to access sensitive keys without hardcoding them. The chosen engine for our embeddings is **text-embedding-ada-002**.

Data collection – fetching the textbook data

For this study, we're analyzing a textbook about insects. Let's fetch and process this data:

```
text =
urlopen('https://www.gutenberg.org/cache/epub/10834/pg10834.txt').read().decode()
documents = list(filter(lambda x: len(x) > 100, text.split('\r\n\r\n')))
print(f'There are {len(documents)} documents/paragraphs')
```

Here, we're downloading the text from its source, splitting it into paragraphs, and ensuring we only keep the more content-rich ones (those longer than 100 characters). We end up with 79 paragraphs in this example.

Converting text to embeddings

The core of our analysis lies in converting text data to embeddings. Let's achieve this:

```
question_embedding = np.array(get_embedding(QUESTION))
embeddings=[get_embedding(document) for document in documents]
embeddings = np.array(embeddings)
```

We loop through each document, convert it into an embedding using our specified engine, and store the embeddings in a **numpy** array for efficient operations.

Querying – searching for relevant information

With our data transformed, let's pose a natural language query and find the most relevant document using our vector embedding. We are using a kind of nearest-neighbor algorithm, as we have seen:

```
QUESTION = 'How many horns does a flea have?'
question_embedding = np.array(get_embedding(QUESTION, engine=ENGINE))
hits = util.semantic_search(question_embedding, embeddings, top_k=1)[0]
print(f'Question: {QUESTION}\n')
for i, hit in enumerate(hits):
    print(f'Document {i + 1} Cos_Sim
{hit["score"]:.3f}:\n\n{documents[hit["corpus_id"]]}\n')
print('\n')
```

We encode our question into an embedding, and then use semantic search to find the closest matching document from our dataset. The result provides us with insights into our query. With this structure, we've transformed our code into a more instructive, step-by-step guide that should be more accessible and understandable.

Concluding thoughts – the power of modern pre-trained models

In the rapidly evolving world of ML and AI, what we've witnessed in this case study is just a small taste of the vast potential of modern pre-trained models. Here's a brief contemplation on their profound impact:

- **Unprecedented efficiency:** Gone are the days when we had to train models from scratch for every new task. Pre-trained models, fine-tuned for specific tasks, have removed significant barriers in terms of time, computation, and resources. With a few lines of code, we were able to access and harness the power of models that have trained on vast amounts of data, a task that would've been monumental just a decade ago.
- **Broadened accessibility:** Not only do pre-trained models save time, but they also democratize access to cutting-edge AI technology. Developers, researchers, and hobbyists without extensive ML backgrounds or access to massive compute resources can now embark on AI projects with ease.
- **Rapid prototyping:** The ability to quickly spin up models and test ideas allows for a more iterative and innovative approach to problem-solving. This rapid prototyping is especially important in industries that require quick turnarounds or where the first-mover advantage is crucial.
- **Versatility and scalability:** The models we use today, such as OpenAI's embedding engines, are versatile. Whether you're building a semantic search engine, a recommendation system, or any other application that requires understanding context, these models can be your cornerstone. As your project grows, these models can scale with you, ensuring consistent performance.

In conclusion, the landscape of AI has been revolutionized by the advent of pre-trained models. Their power and efficiency underscore a new era where building advanced AI prototypes and projects is no longer a distant dream but an easily attainable reality. As technology continues to advance, it's exciting to ponder what further innovations lie on the horizon and how they will shape our interconnected world.

Summary

As we reach the conclusion of this comprehensive case study chapter, it's important to highlight that the journey doesn't end here. The power of modern ML and AI is vast and ever-growing, and there is always more to learn, explore, and create.

Our official GitHub repository serves as a central hub, housing not only the code and detailed explanations from this case study but also an extensive collection of additional resources, examples, and even more intricate case studies:

- **More case studies:** Dive deeper into the world of ML with an array of case studies spanning various domains and complexities. Each case study is meticulously crafted to provide you with hands-on experience, guiding you through different challenges and solutions in the AI landscape.
- **Comprehensive code examples:** The repository is rich with code examples that complement the case studies and explanations provided. These examples are designed to be easily understandable and executable, allowing you to grasp the practical aspects of the concepts discussed.
- **Interactive learning:** Engage with interactive notebooks and applications that provide a hands-on approach to learning, helping solidify your understanding of key concepts and techniques.

- **Community and collaboration:** Join a community of learners and contributors. The repository is an open space for collaboration, questions, and discussions. Your participation helps create a vibrant learning environment, fostering growth and innovation.
- **Continuous updates and additions:** The field of ML is dynamic, and our repository reflects that. Stay updated with the latest trends, techniques, and case studies by regularly checking back for new content and updates.

The road to mastering ML is a journey, not a destination. The repository is designed to be your companion on this journey, providing you with the tools, knowledge, and community support needed to thrive in the AI world.

Looking forward, we are excited about the future developments in ML and AI. We are committed to updating our resources, adding new case studies, and continually enhancing the learning experience for everyone.

Thank you for choosing to learn with us, and we hope that the resources provided serve as a springboard for your future endeavors in AI and ML. Here's to exploring the unknown, solving complex problems, and creating a smarter, more connected world together!

Index

As this ebook edition doesn't have fixed pagination, the page numbers below are hyperlinked for reference only, based on the printed edition of this book.

A

A/B test [109](#)

adversarial debiasing [253](#)

algorithmic bias [248](#), [249](#)

 aggregation bias [250](#)

 disparate impact [249](#)

 disparate treatment [249](#)

 emerging techniques [257](#)

 measurement bias [249](#)

 measuring [251](#)

 mitigating [252](#)

 mitigating, during data preprocessing [253](#)

 mitigating, during model in-processing [253](#)

 mitigating, during model postprocessing [254](#)

 pre-existing bias [249](#)

 proxy bias [250](#)

 sample bias [249](#)

 significance, of fairness [252](#)

 types [248](#), [249](#)

 unaddressed bias, consequences [251](#)

algorithmic bias, sources

 aggregation bias [250](#)

 historical bias [250](#)

proxy bias [250](#)

representation or sample bias [250](#)

alternative hypothesis [141](#)

architectural governance [264](#), [275](#)

dimensions [276](#)

pillars [275](#), [276](#)

transformative architectural principles [276](#)

arithmetic mean [114](#)

arithmetic symbols [58](#)

B

bar charts [157-159](#)

Bayesian approach

versus Frequentist approach [73-79](#)

Bayes' theorem [85-89](#)

applications [89](#)

applications, example [89-92](#)

BERT

TL with [237](#), [238](#)

BERT's pre-training

decoding [233](#)

MLM [233](#), [234](#)

NSP [234](#), [235](#)

bias [275](#)

bias, in GPT-2 [255-257](#)

bias, in LLMs [254](#)

GPT-2 [255](#)

big data [10](#)

binary classifier [83](#), [84](#)

binomial random variable [98](#)

binomial random variable, examples

 fundraising meetings [99](#)

 restaurant openings [99](#), [100](#)

box plots [160-163](#)

bracket notation [43](#)

broadened accessibility [290](#)

C

categorical variables

 hypothesis testing [148](#)

causation [164](#)

central limit theorem [138](#)

centroid [211](#)

chi-square goodness of fit test [148](#)

 assumptions [148](#), [149](#)

 example [149](#), [150](#)

 using, scenarios [148](#)

chi-square test for association/independence [150](#)

 assumptions [151](#), [152](#)

chi-square tests [148](#)

classification models [178](#)

clustering [179](#), [211](#)

coefficient of variation

 employee salary example [119](#)

coffee shop data example, qualitative and quantitative data [19-21](#)

comma-separated value (CSV) file [22](#)

communication

 significance [153](#), [154](#)

- COMPAS dataset, case study [279-281](#)
 - data preparation, for modeling [285](#)
 - preliminary data exploration [282-285](#)
 - task/outlining success [282](#)
- compound events [76-79](#)
- computer programming [6, 7](#)
 - python, using [7](#)
- concept drift [258, 275](#)
- conditional probability of A given B ($P(A|B)$) [79](#)
- confidence interval [138-141](#)
- confounding factor [112](#)
- confusion matrices [83](#)
- confusion matrix [83](#)
- continuous data [24](#)
- continuous random variable [103-106](#)
- continuous variables
 - predicting, with linear regression [184-186](#)
- continuous variables, with linear regression
 - causation [187](#)
 - correlation, versus causation [186](#)
 - metrics [189-193](#)
 - predictors, adding [187, 188](#)
- correlation [126](#)
 - versus causation [164-166](#)
- correlation coefficients [126, 127](#)
- curse of dimensionality (COD) [219](#)

D

- data [2](#)

data drift [258](#)

feature drift [258](#)

label drift [259](#)

sources [259](#)

data exploration [39](#), [40](#)

DataFrames [42](#)

questions, guiding [40](#)

Series object [43](#)

Yelp [40-42](#)

DataFrames [42](#)

data governance [264](#)

audit preparedness [268](#)

change management [268](#)

current hurdles [265](#)

documentation and cataloging [267](#)

impact analysis [268](#)

mastering [265](#)

regulatory compliance [268](#)

troubleshooting and analysis [269](#)

data ingestion [266](#)

data integration [267](#)

data levels [24](#)

interval level [27](#)

nominal level [24](#)

ordinal level [25](#)

ratio level [31](#)

data management [266](#)

data mining [9](#)

data model [6](#)

data, obtaining [108](#)

experimental [108](#), [109](#)

observational [108](#), [109](#)

data path [268](#)

data preparation for modeling, COMPAS dataset case study

feature engineering [285](#)

juvenile crime data, combining [286](#)

one-hot encoding categorical features [286](#)

skewed features, standardizing [287](#)

data quality [267](#)

upholding [268](#), [269](#)

data science [1](#), [35](#), [36](#)

COVID-19, predicting with machine learning [3](#), [4](#)

need for [3](#)

steps, overview [36](#), [37](#)

terminology [2](#), [3](#)

data science, case studies [10](#)

government paper pushing, automating [11](#), [12](#)

job description [12-15](#)

data science, steps

data, exploring [38](#)

data, modeling [38](#)

data, obtaining [37](#)

interesting question, asking [37](#)

results, communicating and visualizing [39](#)

data science Venn diagram [4-6](#)

computer programming [6](#), [7](#)

domain knowledge [9](#)

math [6](#)

single tweet, parsing [8](#)

data scientist

causation [168](#)

data types

quantitative, versus qualitative data [19](#)

structured, versus unstructured data [18](#)

data visualization

bar charts [157-159](#)

box plots [160-163](#)

histograms [159](#), [160](#)

identifying [154](#)

line graphs [156](#), [157](#)

scatter plot [154](#), [155](#)

data warehouses [267](#)

decision trees [204](#)

dummy variables [207-210](#)

purity, measuring [204](#)

Titanic dataset, exploring [205-207](#)

deep learning (DL) [175](#)

deep learning (DL) model [238](#)

dimension reduction [179](#), [221](#)

implementing ways [221](#)

discrete data [23](#)

discrete random variables [93-98](#)

types [98](#)

discrete random variables, types

binomial random variable [98](#), [99](#)

continuous random variable [103-106](#)

geometric random variable [100](#), [101](#)

Poisson random variable [102](#)

domain knowledge [9](#)

dot product [59](#), [60](#)

drift, mitigating [259](#)

context [260](#)

continuous monitoring [260](#)

feedback systems, implementing [260](#)

model adaptation techniques [261](#)

regular model retraining [260](#)

dummy variables [207-210](#)

E

empirical rule [128](#), [129](#)

exam scores example [129](#)

empty set [64](#)

entity resolution (ER) [267](#)

event [72](#)

experiment [109](#)

experimental [109-111](#)

experimental units [109](#)

exploratory data analysis (EDA) [9](#), [155](#), [211](#)

exponent [60-63](#)

F

fairness-aware algorithms

emerging techniques [257](#)

false positive [147](#)

farness through awareness method [253](#)

farness through unawareness method [253](#)

feature drift [258](#)

feature extraction [211](#), [219-227](#)

Frequentist approach

versus Bayesian approach [73-79](#)

G

geometric mean [31](#)

geometric random variable [100](#), [101](#)

weather example [101](#), [102](#)

Gini index [204](#)

Google Cloud Storage (GCS) [265](#)

GPT

TL with [237](#), [238](#)

H

histograms [159](#), [160](#)

hypothesis testing

for categorical variables [148](#)

hypothesis tests [141](#)

conducting [142](#), [143](#)

I

image-based models [232](#)

inductive TL (ITL) [236](#)

in-processing techniques [252](#)

intersection [64](#), [65](#)

interval level [27](#)

examples [28](#)

mathematical operations [28](#)

measures of center [28](#), [29](#)

measures of variation [29](#)

standard deviation [29](#), [30](#)

J

Jaccard measure [65](#)

K

K and cluster validation

optimal number, selecting [217](#)

key performance indicator (KPI) [168](#)

k-means clustering [211](#)

illustrative example [212-216](#)

L

label drift [259](#)

labeled data [175](#)

large language models (LLMs) [230](#), [288](#)

Learning Fair Representations (LFR) [257](#)

likelihood [87](#)

Likert [26](#)

Likert scale [96](#)

linear algebra [60](#), [66](#)

linear regression [125](#)

continuous variables, predicting with [184-186](#)

line graphs [156](#), [157](#)

Local Interpretable Model-Agnostic Explanations (LIME) [257](#), [270](#)

logarithm [61-63](#)

M

machine learning (ML) [6](#), [9](#), [36](#), [172](#)

caveats [173](#), [174](#)

facial recognition [172](#), [173](#)

reinforcement learning (RL) [181-183](#)

SML [182](#)

supervised learning (SL) [175](#), [176](#)

types [175](#)

types, overview [182](#)

unsupervised learning (UL) [179-181](#)

unsupervised ML (UML) [183](#)

usage [174](#)

using, to predict COVID-19 [3](#), [4](#)

machine learning (ML), governance [264](#), [269](#)

facets, of ML development [274](#)

model deployment and monitoring [275](#)

model interpretability [270-274](#)

pillars [270](#)

magnitude of set [63](#)

masked language modeling (MLM) [233](#), [234](#)

math [6](#)

using [6](#)

matrix [57](#)

matrix multiplication [66](#), [67](#)

performing [67-69](#)

Mean Absolute Error (MAE) [189](#)

mean of variable [94](#)

Mean Squared Error (MSE) [189](#)

measure of center [113](#), [114](#)

measures of relative standing [120-125](#)

measures of variation [114-119](#)

median [114](#)

methods, for measuring bias

- confusion matrix [251](#)

- counterfactual analysis [251](#)

- disparate impact analysis [251](#)

- equality of odds [251](#)

- equality of opportunity [251](#)

- fairness through awareness [251](#)

model drift [258](#)

- concept drift [258](#)

- prediction drift [258](#)

- sources [259](#)

models [6](#)

N

Naïve Bayes algorithm [85](#)

naïve Bayes classification

- features [195](#), [196](#)

- metrics [197-204](#)

- performing [195-197](#)

natural language processing (NLP) [255](#), [288](#)

natural logarithm [62](#)

neural network (NN) [38](#), [175](#)

next sentence prediction (NSP) [234](#), [235](#)

nominal level [24](#)

- data [25](#)

examples [24](#)

mathematical operations [25](#)

measure of center [25](#)

normalizing constant [87](#)

notation [72](#)

null accuracy rate [198](#)

null hypothesis [141](#)

null model [193](#)

null set [64](#)

numpy array [56](#)

O

observational [109](#)

one-sample t-tests [143](#), [144](#)

assumptions [144-146](#)

example [144](#)

one-tailed test [145](#)

ordinal level [25](#)

examples [26](#)

Likert [26](#)

mathematical operations [26](#)

measures of center [27](#)

overfitting [191](#)

P

pandas

filtering in [45-47](#)

parameter [108](#)

personally identifiable information (PII) [40](#)

platform as a service (PaaS) [276](#)

point estimates [131-136](#)

issues [138](#)

Poisson distribution [132](#)

Poisson random variable [102](#)

example [102](#), [103](#)

population [107](#)

posterior [87](#)

postprocessing techniques [252](#)

prediction drift [258](#)

pre-processing [18](#)

preprocessing techniques [252](#)

pre-trained model [230](#)

BERT's pre-training [233](#)

fine-tuning, for text classification [238-240](#)

image-based models [232](#)

text-based models [232](#), [233](#)

TL [235](#)

usage [231](#)

using, benefits [230](#), [231](#)

principal component analysis (PCA) [219-227](#)

prior [87](#)

prior probability [87](#)

probability [72](#), [73](#)

concepts [71](#), [72](#)

rules, utilizing [79](#)

probability density function (PDF) [103](#)

probability mass function (PMF) [93](#), [99](#)

probability, rules

- addition rule [80](#)

- complementary events [82](#), [83](#)

- independence [82](#)

- multiplication rule [81](#), [82](#)

- mutual exclusivity [80](#)

probability sampling [111](#)

procedure [71](#)

proper subset [64](#)

Q

qualitative data [19](#)

- exploration, tips [44](#)

qualitative data, exploration tips

- filtering, in pandas [45-47](#)

- nominal-level columns [44](#), [45](#)

- ordinal-level columns [48](#), [49](#)

- Titanic dataset [49-53](#)

quantitative data [19](#)

quantitative variables [156](#)

R

random sampling [111](#), [112](#)

random variable [92](#), [93](#)

- discrete random variable [93-98](#)

random variables [85](#)

rapid prototyping [290](#)

ratio level [31](#)

- examples [31](#)

- issues [32](#)
- mathematical operations [31](#)
- measures of center [31](#)
- regression models [178](#)
- reinforcement learning (RL) [181](#), [182](#)
 - advantages [183](#)
 - disadvantages [183](#)
 - applying, in various domains [181](#)
- relative frequency [74](#)
- Root MSE (RMSE) [189](#)

S

- sample mean [132](#)
- sample space [72](#)
- sampling data [111](#)
 - probability sampling [111](#)
 - unequal probability sampling [113](#)
- sampling distributions [136-138](#)
- scalar [59](#)
- scatter plots [154](#), [155](#)
- semi-structured data [19](#)
- sentiment analysis (SA) [203](#), [258](#)
- Series object [43](#)
- set theory [63-66](#)
- SHapley Additive exPlanations (SHAP) [257](#)
- Silhouette Coefficient [217](#), [218](#)
- Simpson's paradox [166-168](#)
- software as a service (SaaS) [266](#)
- square matrix [57](#)

- standard deviation [115](#)
- standard normal distribution [103](#)
- statistical model [9](#)
- statistics [107](#), [108](#)
- statistics, measuring [113](#)
 - coefficient of variation [119](#)
 - measure of center [113](#), [114](#)
 - measures of relative standing [120-125](#)
 - measures of variation [114-118](#)
- stopping criterion [214](#)
- structured data [2](#), [18](#)
 - examples [18](#)
- subset [64](#)
- summation [58](#)
- superset [64](#)
- supervised learning (SL) [175](#), [176](#)
 - deciding, between classification and regression [179](#)
 - heart attack prediction [176-178](#)
 - method [203](#)
 - types [178](#), [179](#)
- Supervised ML (SML) models [175](#), [182](#)
 - advantages [183](#)
 - disadvantage [183](#)
- synthetic minority oversampling (SMOTE) technique [253](#)

T

- text-based models [232](#), [233](#)
- text classification
 - pre-trained model, fine-tuning for [238-240](#)

text embeddings, with pretrained models and OpenAI [288](#)

data collection, fetching from textbook data [289](#)

data querying [290](#)

libraries, importing [288](#), [289](#)

libraries, setting up [288](#), [289](#)

text, converting to embeddings [289](#)

Titanic dataset [49-53](#)

exploring [205-207](#)

transductive TL (TTL) [237](#)

transfer learning (TL) [235](#)

examples [238-244](#)

process [236](#)

types [236](#)

with BERT [237](#), [238](#)

with GPT [237](#), [238](#)

transfer learning (TL), types

inductive TL (ITL) [236](#)

transductive TL (TTL) [237](#)

unsupervised TL (UTL) [237](#)

two-tailed test [145](#)

Type I error [84](#), [147](#)

Type II error [84](#), [147](#)

U

unequal probability sampling [113](#)

union [65](#)

unprecedented efficiency [290](#)

unstructured data [2](#), [18](#)

examples [18](#)

unsupervised learning (UL) [179-181](#), [210](#)

 k-means clustering [211](#)

 Silhouette Coefficient [217](#), [218](#)

 usage, considerations [210](#), [211](#)

unsupervised ML (UML) [211](#)

 advantages [183](#)

 disadvantage [183](#)

unsupervised TL (UTL) [237](#)

V

vector [56](#)

verbal communication [168-170](#)

Visual Geometry Group (VGG) [232](#)

W

world alcohol consumption data example [21-23](#)

World Health Organization's (WHO's) [157](#)

World Health Organization (WHO) [21](#)

Y

Yelp [40-42](#)



Packtpub.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at packtpub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packtpub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

<packt>



2ND EDITION

Building Data Science Applications with FastAPI

Develop, manage, and deploy efficient machine
learning applications with Python



FRANÇOIS VORON

Building Data Science Applications with FastAPI - Second Edition

François Voron

ISBN: 978-1-83763-274-9

- Explore the basics of modern Python and async I/O programming
- Get to grips with basic and advanced concepts of the FastAPI framework
- Deploy a performant and reliable web backend for a data science application
- Integrate common Python data science libraries into a web backend
- Integrate an object detection algorithm into a FastAPI backend
- Build a distributed text-to-image AI system with Stable Diffusion
- Add metrics and logging and learn how to monitor them

<packt>



3RD EDITION

Python Deep Learning

Understand how deep neural networks work and
apply them to real-world tasks



IVAN VASILEV

Python Deep Learning - Third Edition

Ivan Vasilev

ISBN: 978-1-83763-850-5

- Establish theoretical foundations of deep neural networks
- Understand convolutional networks and apply them in computer vision applications
- Become well versed with natural language processing and recurrent networks
- Explore the attention mechanism and transformers
- Apply transformers and large language models for natural language and computer vision
- Implement coding examples with PyTorch, Keras, and Hugging Face Transformers
- Use MLOps to develop and deploy neural network models

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share Your Thoughts

Now you've finished *Principles of Data Science*, we'd love to hear your thoughts! If you purchased the book from Amazon, please [click here to go straight to the Amazon review page](#) for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below



<https://packt.link/free-ebook/9781837636303>

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly