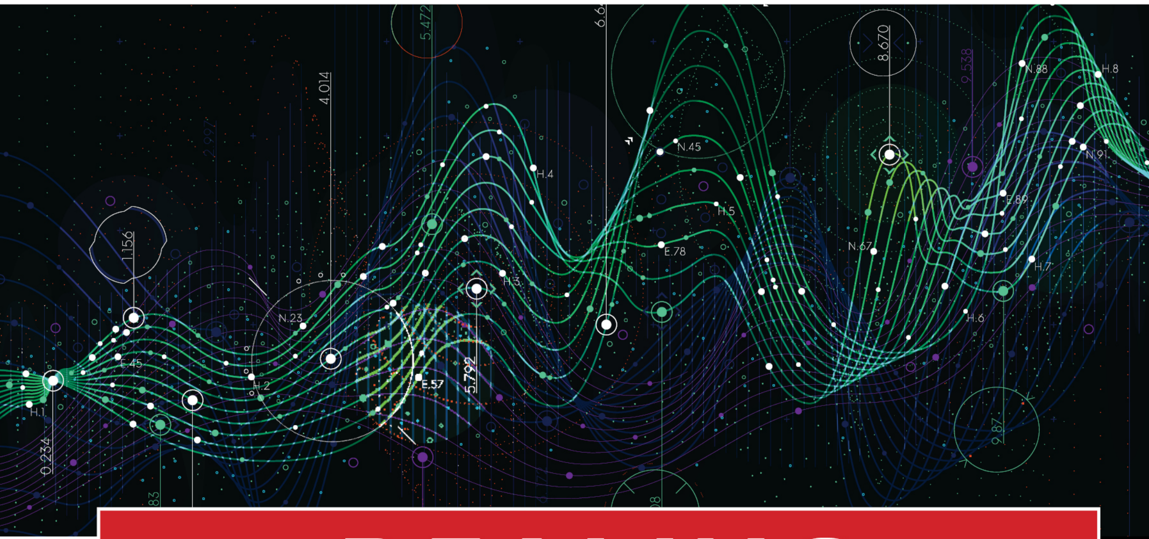

DEALING WITH DATA

POCKET PRIMER

O. CAMPESATO

Dealing with Data

Pocket Primer

Dealing with Data

Pocket Primer

Oswald Campesato

Mercury Learning and Information
Dulles, Virginia
Boston, Massachusetts
New Delhi

Publisher: David Pallai

Mercury Learning and Information
22841 Quicksilver Drive
Dulles, VA 20166
info@merclearning.com
www.merclearning.com
800-232-0223

O. Campesato. *Dealing with Data Pocket Primer.*
ISBN: 978-1-683928-201

Library of Congress Control Number: 2022934840
222324321 This book is printed on acid-free paper in the United States of America.

I'd like to dedicate this book to my parents
– may this bring joy and happiness into their lives.

CONTENTS

PREFACE

What Is the Value Proposition for This Book?

This book contains a fast-paced introduction to as much relevant information about dealing with data that can be reasonably included in a book this size. You will be exposed to statistical concepts, data-related techniques, features of `Pandas`, `SQL`, `NLP` topics, and data visualization.

Keep in mind that some topics are presented in a cursory manner, which is for two main reasons. First, it's important that you be exposed to these concepts. In some cases, you will find topics that might pique your interest, and hence motivate you to learn more about them through self-study; in other cases, you will probably be satisfied with a brief introduction. In other words, you will decide whether or not to delve into more detail regarding the topics in this book.

Second, a full treatment of all the topics that are covered in this book would significantly increase the size of this book, and few people are interested in reading technical tomes.

The Target Audience

This book is intended primarily for people who plan to become data scientists as well as anyone who needs to perform data cleaning tasks. This book is also intended to reach an international audience of readers with highly diverse backgrounds in various age groups. Hence, this book uses standard English rather than colloquial expressions that might be confusing to those readers. As you know, many people learn by different types of imitation; which includes reading, writing, or hearing new material. This book takes these points into consideration in order to provide a comfortable and meaningful learning experience for the intended readers.

What Will I Learn From This Book?

The first chapter briefly introduces basic probability and then discusses basic concepts in statistics, such as the mean, variance, and standard deviation, as well as other concepts. Then you will learn about more advanced concepts, such as Gini impurity, entropy, cross entropy, and KL divergence. You will also learn about different types of distance metrics and Bayesian inference.

Chapter 2 delves into processing different data types in a dataset, along with normalization, standardization, and handling missing data. You will learn about outliers and how to detect them via z-scores and quantile transformation. You will also learn about `SMOTE` for handling imbalanced datasets.

Chapter 3 introduces `Pandas`, which is a very powerful `Python` library that enables you to read the contents of `CSV` files (and other text files) into data frames (somewhat analogous to `Excel` spreadsheets), where you can programmatically slice-and-dice the data to conform to your requirements.

Since large quantities of data are stored in the form structured data in relational databases, Chapter 4 introduces you to `SQL` concepts and also how to perform basic operations in `MySQL`, such as working with databases.

Chapter 5 covers database topics such as managing database tables and illustrates how to populate them with data. You will also see examples of `SQL` statements that select rows of data from a collection of database tables.

Chapter 6 introduces you to `NLP` and how to perform tasks such as tokenization and removing stop words and punctuation, followed by stemming and lemmatization.

The final chapter of this book delves into data visualization with `Matplotlib`, `Seaborn`, and an example of a rendering graphics effects in `Bokeh`.

Why Are the Code Samples Primarily in Python?

Most of the code samples are short (usually less than one page and sometimes less than half a page), and if need be, you can easily and quickly copy/paste the code into a new Jupyter notebook. For the `Python` code samples that reference a `CSV` file, you do not need any additional code in the corresponding Jupyter notebook to access the `CSV` file. Moreover, the code samples execute quickly, so you won't need to avail yourself of the free GPU that is provided in Google Colaboratory.

If you do decide to use Google Colaboratory, you can easily copy/paste the `Python` code into a notebook, and also use the upload feature to upload existing Jupyter notebooks. Keep in mind the following point: if the `Python` code references a `CSV` file, make sure that you include the appropriate code snippet (as explained in Chapter 1) to access the `CSV` file in the corresponding Jupyter notebook in Google Colaboratory.

Do I Need to Learn the Theory Portions of This Book?

Once again, the answer depends on the extent to which you plan to become involved in data analytics. For example, if you plan to study machine learning, then you will probably learn how to create and train a model, which is a task that is performed after data cleaning tasks. In general, you will probably need to learn everything that you encounter in this book if you are planning to become a machine learning engineer.

Getting the Most From This Book

Some programmers learn well from prose, others learn well from sample code (and lots of it), which means that there's no single style that can be used for everyone.

Moreover, some programmers want to run the code first, see what it does, and then return to the code to delve into the details (and others use the opposite approach).

Consequently, there are various types of code samples in this book: some are short, some are long, and other code samples "build" from earlier code samples.

What Do I Need to Know for This Book?

Current knowledge of Python 3.x is the most helpful skill. Knowledge of other programming languages (such as Java) can also be helpful because of the exposure to programming concepts and constructs. The less technical knowledge that you have, the more diligence will be required in order to understand the various topics that are covered.

If you want to be sure that you can grasp the material in this book, glance through some of the code samples to get an idea of how much is familiar to you and how much is new.

Does This Book Contain Production-Level Code Samples?

The primary purpose of the code samples in this book is to show you Python-based libraries for solving a variety of data-related tasks in conjunction with acquiring a rudimentary understanding of statistical concepts. Clarity has higher priority than writing more compact code that is more difficult to understand (and possibly more prone to bugs). If you decide to use any of the code in this book in a production website, you ought to subject that code to the same rigorous analysis as the other parts of your code base.

What Are the Nontechnical Prerequisites for This Book?

Although the answer to this question is more difficult to quantify, it's very important to have strong desire to learn about data analytics, along with the motivation and discipline to read and understand the code samples.

How Do I Set Up a Command Shell?

If you are a Mac user, there are three ways to do so. The first method is to use `Finder` to navigate to `Applications` > `Utilities` and then double click on the `Utilities` application. Next, if you already have a command shell available, you can launch a new command shell by typing the following command:

```
open /Applications/Utilities/Terminal.app
```

A second method for Mac users is to open a new command shell on a Macbook from a command shell that is already visible simply by clicking `command+n` in that command shell, and your Mac will launch another command shell.

If you are a PC user, you can install Cygwin (open source *https://cygwin.com/*) that simulates bash commands, or use another toolkit such as MKS (a commercial product). Please read the online documentation that describes the download and installation process. Note that custom aliases are not automatically set if they are defined in a file other than the main start-up file (such as .bash_login).

Companion Files

All the code samples and figures in this book may be obtained by writing to the publisher at info@merclearning.com.

What Are the "Next Steps" After Finishing This Book?

The answer to this question varies widely, mainly because the answer depends heavily on your objectives. If you are interested primarily in `NLP`, then you can learn more advanced concepts, such as attention, transformers, and the `BERT`-related models.

If you are primarily interested in machine learning, there are some subfields of machine learning, such as deep learning and reinforcement learning (and deep reinforcement learning) that might appeal to you. Fortunately, there are many resources available, and you can perform an internet search for those resources. Keep in mind the different aspects of machine learning that pertain to you will vary as the needs of a machine learning engineer, data scientist, manager, student, or software developer are all different.

CHAPTER 1

Introduction to Probability and Statistics

This chapter introduces you to concepts in probability as well as to an assortment of statistical terms and algorithms.

The first section of this chapter starts with a discussion of probability, how to calculate the expected value of a set of numbers (with associated probabilities), and the concept of a random variable (discrete and continuous), and a short list of some well-known probability distributions.

The second section of this chapter introduces basic statistical concepts, such as *mean*, *median*, *mode*, *variance*, and *standard deviation*, along with simple examples that illustrate how to calculate these terms. You will also learn about the terms `RSS`, `TSS`, `R^2`, and `F1` score.

The third section of this chapter introduces Gini Impurity, entropy, perplexity, cross-entropy, and KL divergence. You will also learn about skewness and kurtosis.

The fourth section explains covariance and correlation matrices and how to calculate eigenvalues and eigenvectors.

The fifth section explains principal component analysis (PCA), which is a well-known dimensionality reduction technique. The final section introduces you to Bayes' Theorem.

WHAT IS A PROBABILITY?

If you have ever performed a science experiment in one of your classes, you might remember that measurements have some uncertainty. In general, we assume that there is a correct value, and we endeavor to find the best estimate of that value.

When we work with an event that can have multiple outcomes, we try to define the probability of an outcome as the chance that it will occur, which is calculated as follows:

```
p(outcome)  = # of times outcome occurs/(total number of
outcomes)
```

For example, in the case of a single balanced coin, the probability of tossing a head "H" equals the probability of tossing a tail "T":

```
p(H) = 1/2 = p(T)
```

The set of probabilities associated with the outcomes {H, T} is shown in the set `P`:

```
P = {1/2, 1/2}
```

Some experiments involve replacement while others involve non-replacement. For example, suppose that an urn contains 10 red balls and 10 green balls. What is the probability that a randomly selected ball is red? The answer is 10/(10+10) = 1/2. What is the probability that the second ball is also red?

There are two scenarios with two different answers. If each ball is selected with replacement, then each ball is returned to the urn after selection, which means that the urn always contains 10 red balls and 10 green balls. In this case, the answer is 1/2 * 1/2 = 1/4. In fact, the probability of any event is independent of all previous events.

On the other hand, if balls are selected without replacement, then the answer is 10/20 * 9/19. As you undoubtedly know, card games are also examples of selecting cards without replacement.

One other concept is called *conditional probability*, which refers to the likelihood of the occurrence of event `E1` given that event `E2` has occurred. A simple example is the following statement:

```
"If it rains (E2), then I will carry an umbrella (E1)."
```

Calculating the Expected Value

Consider the following scenario involving a well-balanced coin: whenever a head appears, you earn $1 and whenever a tail appears, you earn $1 dollar. If you toss the coin 100 times, how much money do you expect to earn? Since you will earn $1 regardless of the outcome, the expected value (in fact, the guaranteed value) is 100.

Now consider this scenario: whenever a head appears, you earn $1 and whenever a tail appears, you earn 0 dollars. If you toss the coin 100 times, how much money do you expect to earn? You probably determined the value 50 (which is the correct answer) by making a quick mental calculation. The more formal derivation of the value of E (the expected earning) is here:

```
E = 100 *[1 * 0.5 + 0 * 0.5] = 100 * 0.5 = 50
```

The quantity **1** * 0.5 + **0** * 0.5 is the amount of money you expected to earn during each coin toss (half the time you earn $1 and half the time you earn

0 dollars), and multiplying this number by 100 is the expected earning after 100 coin tosses. Also note that you might never earn $50: the actual amount that you earn can be any integer between 1 and 100 inclusive.

As another example, suppose that you earn $3 whenever a head appears, and you *lose* $1.50 dollars whenever a tail appears. Then the expected earning `E` after 100 coin tosses is shown here:

```
E = 100 *[3 * 0.5 - 1.5 * 0.5] = 100 * 1.5 = 150
```

We can generalize the preceding calculations as follows. Let P = `{p1, . . . ,pn}` be a probability distribution, which means that the values in P are nonnegative and their sum equals 1. In addition, let R = `{R1, . . . ,Rn}` be a set of rewards, where reward Ri is received with probability `pi`. Then the expected value E after N trials is shown here:

```
E = N * [SUM pi*Ri]
```

In the case of a single balanced die, we have the following probabilities:

```
p(1) = 1/6
p(2) = 1/6
p(3) = 1/6
p(4) = 1/6
p(5) = 1/6
p(6) = 1/6
P = {1/6, 1/6, 1/6, 1/6, 1/6, 1/6}
```

As a simple example, suppose that the earnings are {3, 0, −1, 2, 4, −1} when the values 1, 2, 3, 4, 5, 6, respectively, appear when tossing the single die. Then after 100 trials our expected earnings are calculated as follows:

```
E = 100 * [3 + 0 + -1 + 2 + 4 + -1]/6 = 100 * 3/6 = 50
```

In the case of two balanced dice, we have the following probabilities of rolling 2, 3, . . . , or 12:

```
p(2) = 1/36
p(3) = 2/36
...
p(12) = 1/36
P = {1/36,2/36,3/36,4/36,5/36,6/36,5/36,4/36,3/36,2/36,1/36}
```

RANDOM VARIABLES

A random variable is a variable that can have multiple values, where each value has an associated probability of occurrence. For example, if we let `X` be a random variable whose values are the outcomes of tossing a well-balanced die, then the values of `X` are the numbers in the set {1, 2, 3, 4, 5, 6}. Moreover, each of those values can occur with equal probability (which is 1/6).

In the case of two well-balanced dice, let `x` be a random variable whose values can be any of the numbers in the set {2, 3, 4, . . . , 12}. Then the associated probabilities for the different values for `x` are listed in the previous section.

Discrete versus Continuous Random Variables

The preceding section contains examples of *discrete* random variables because the list of possible values is either finite or countably infinite (such as the set of integers). As an aside, the set of rational numbers is also countably infinite, but the set of irrational numbers and also the set of real numbers are both uncountably infinite (proofs are available online). As pointed out earlier, the associated set of probabilities must form a probability distribution, which means that the probability values are nonnegative and their sum equals 1.

A *continuous* random variable is a random variable whose values can be *any* number in an interval, which can be an uncountably infinite number of distinct values. For example, the amount of time required to perform a task is represented by a continuous random variable.

A continuous random variable also has a probability distribution that is represented as a continuous function. The constraint for such a variable is that the area under the curve (which is sometimes calculated via a mathematical integral) equals 1.

Well-Known Probability Distributions

There are many probability distributions, and some of the well-known probability distributions are listed here:

- Gaussian distribution
- Poisson distribution
- chi-squared distribution
- binomial distribution

The *Gaussian distribution* is named after Karl F Gauss, and it is sometimes called the normal distribution or the Bell curve. The Gaussian distribution is symmetric: the shape of the curve on the left of the mean is identical to the shape of the curve on the right side of the mean. As an example, the distribution of IQ scores follows a curve that is similar to a Gaussian distribution.

Furthermore, the frequency of traffic at a given point in a road follows a Poisson distribution (which is not symmetric). Interestingly, if you count the number of people who go to a public pool based on 5° (Fahrenheit) increments of the ambient temperature, followed by 5° decrements in temperature, that set of numbers follows a Poisson distribution.

Perform an Internet search for each of the bullet items in the preceding list and you will find numerous articles that contain images and technical details about these (and other) probability distributions.

This concludes the brief introduction to probability, and the next section delves into the concepts of mean, median, mode, and standard deviation.

FUNDAMENTAL CONCEPTS IN STATISTICS

This section contains several subsections that discuss the mean, median, mode, variance, and standard deviation. Feel free to skim (or skip) this section if you are already familiar with these concepts. As a starting point, we will suppose that we have a set of numbers X = `{x1, ..., xn}` that can be positive, negative, integer-valued or decimal values.

The Mean

The *mean* of the numbers in the set `X` is the average of the values. For example, if the set `X` consists of `{-10,35,75,100}`, then the mean equals (−10 + 35 + 75 + 100)/4 = 50. If the set `X` consists of `{2,2,2,2}`, then the mean equals (2 + 2 + 2 + 2)/4 = 2. As you can see, the mean value is not necessarily one of the values in the set.

Keep in mind that the mean is sensitive to outliers. For example, the mean of the set of numbers {1, 2, 3, 4} is 2.5, whereas the mean of the set of number {1, 2, 3, 4, 1000} is 202. Since the formulas for the variance and standard deviation involve the mean of a set of numbers, both of these terms are also more sensitive to outliers.

The Median

The *median* of the numbers (sorted in increasing or decreasing order) in the set `X` is the middle value in the set of values, which means that half the numbers in the set are less than the median and half the numbers in the set are greater than the median. For example, if the set `X` consists of `{-10,35,75,100}`, then the *median* equals 55 because 55 is the average of the two numbers 35 and 75. As you can see, half the numbers are less than 55 and half the numbers are greater than 55. If the set `X` consists of `{2,2,2,2}`, then the *median* equals 2.

By contrast, the median is much less sensitive to outliers than the mean. For example, the median of the set of numbers {1, 2, 3, 4} is 2.5, and the median of the set of numbers {1, 2, 3, 4, 1000} is 3.

The Mode

The *mode* of the numbers (sorted in increasing or decreasing order) in the set `X` is the most frequently occurring value, which means that there can be more than one such value. If the set `X` consists of `{2,2,2,2}`, then the *mode* equals 2.

If `X` is the set of numbers `{2,4,5,5,6,8}`, then the number 5 occurs twice and the other numbers occur only once, so the *mode* equals 5.

If `X` is the set of numbers `{2,2,4,5,5,6,8}`, then the numbers 2 and 5 occur twice and the other numbers occur only once, so the *mode* equals 2 and 5. A set that has two modes is called bimodal, and a set that has more than two modes is called multimodal.

One other scenario involves sets that have numbers with the same frequency and they are all different. In this case, the mode does not provide meaningful information, and one alternative is to partition the numbers into subsets and then select the largest subset. For example, if set `X` has the values `{1,2,15,16,17,25,35,50}`, we can partition the set into subsets whose elements are in range that are multiples of ten, which results in the subsets {1, 2}, {15, 16, 17}, {25}, {35}, and {50}. The largest subset is {15, 16, 17}, so we could select the number 16 as the mode.

As another example, if set `X` has the values `{-10,35,75,100}`, then partitioning this set does not provide any additional information, so it is probably better to work with either the mean or the median.

The Variance and Standard Deviation

The *variance* is the sum of the squares of the difference between the numbers in `X` and the mean `mu` of the set `X`, divided by the number of value in X, as shown here:

```
variance = [SUM (xi - mu)**2 ] / n
```

For example, if the set `X` consists of `{-10,35,75,100}`, then the *mean* equals (–10 + 35 + 75 + 100)/4 = 50, and the variance is computed as follows:

```
variance = [(-10-50)**2 + (35-50)**2 + (75-50)**2 + (100-50)**2]/4
         = [60**2 + 15**2 + 25**2 + 50**2]/4
         = [3600 + 225 + 625 + 2500]/4
         = 6950/4 = 1,737
```

The standard deviation `std` is the square root of the variance:

```
std = sqrt(1737) = 41.677
```

If the set `X` consists of `{2,2,2,2}`, then the *mean* equals (2 + 2 + 2 + 2)/4 = 2, and the variance is computed as follows:

```
variance = [(2-2)**2 + (2-2)**2 + (2-2)**2 + (2-2)**2]/4
          = [0**2 + 0**2 + 0**2 + 0**2]/4
          = 0
```

The standard deviation `std` is the square root of the variance:

```
std = sqrt(0) = 0
```

Population, Sample, and Population Variance

The population specifically refers to the entire set of entities in a given group, such as the entire population of a country, the people over 65 in the United States, or the number of first-year students in a university.

However, in many cases statistical quantities are calculated on samples instead of an entire population. Thus, a sample is a much smaller subset of the given population. See the central limit theorem regarding the distribution of the mean of a set of sample of a population (which need not be a population with a Gaussian distribution).

If you want to learn about techniques for sampling data, here is a list of three different techniques that you can investigate:

- stratified sampling
- cluster sampling
- quota sampling

One other important point: the population variance is calculated by multiplying the sample variance by `n/(n-1)`, as shown here:

```
population variance = [n/(n-1)]*variance
```

Chebyshev's Inequality

Chebyshev's inequality provides a very simple way to determine the minimum percentage of data that lies within k standard deviations. Specifically, this inequality states that for any positive integer k greater than 1, the amount of data in a sample that lies within k standard deviations is at least 1 - 1/k**2. For example, if k = 2, then at least 1 - 1/2**2 = 3/4 of the data must lie within 2 standard deviations.

The interesting part of this inequality is that it is been mathematically proven to be true; that is, it is not an empirical or heuristic-based result. An extensive description regarding Chebyshev's inequality (including some advanced mathematical explanations) can be found here:

https://en.wikipedia.org/wiki/Chebyshev%27s_inequality

What Is a P-Value?

The null hypothesis states that there is no correlation between a dependent variable (such as `y`) and an independent variable (such as `x`). The `p-value` is used to reject the null hypothesis if the `p-value` is small enough (< 0.005) which indicates a higher significance. The threshold value for `p` is typically 1% or 5%.

There is no straightforward formula for calculating `p-values`, which are values that are always between 0 and 1. In fact, `p-values` are statistical quantities to evaluate the null hypothesis, and they can be calculated using `p-value` tables or via spreadsheet/statistical software.

THE MOMENTS OF A FUNCTION (OPTIONAL)

The previous sections describe several statistical terms that is sufficient for the material in this book. However, several of those terms can be viewed from the perspective of different moments of a function.

In brief, the *moments of a function* are measures that provide information regarding the shape of the graph of a function. In the case of a probability distribution, the first four moments are defined below:

- The mean is the first central moment.
- The variance is the second central moment.
- The skewness (discussed later) is the third central moment.
- The kurtosis (discussed later) is the fourth central moment.

More detailed information (including the relevant integrals) regarding moments of a function is available here:

https://en.wikipedia.org/wiki/Moment_(mathematics)#Variance

What Is Skewness?

Skewness is a measure of the asymmetry of a probability distribution. A Gaussian distribution is symmetric, which means that its skew value is zero (it is not exactly zero, but close enough for our purposes). In addition, the skewness of a distribution is the *third* moment of the distribution.

A distribution can be skewed on the left side or on the right side. A *left-sided* skew means that the long tail is on the left side of the curve, with the following relationships:

```
mean < median < mode
```

A *right-sided* skew means that the long tail is on the right side of the curve, with the following relationships (compare with the left-sided skew):

```
mode < median < mean
```

If need be, you can transform skewed data to a normally distributed dataset using one of the following techniques (which depends on the specific use-case):

- exponential transform
- log transform
- power transform

Perform an online search for more information regarding the preceding transforms and when to use each of these transforms.

What Is Kurtosis?

Kurtosis is related to the skewness of a probability distribution, in the sense that both of them assess the asymmetry of a probability distribution. The kurtosis of a distribution is a scaled version of the *fourth* moment of the distribution, whereas its skewness is the *third* moment of the distribution. Note that the kurtosis of a univariate distribution equals 3.

If you are interested in learning about additional kurtosis-related concepts, you can perform an online search for information regarding mesokurtic, leptokurtic, and platykurtic types of so-called "excess kurtosis."

DATA AND STATISTICS

This section contains various subsections that briefly discuss some of the challenges and obstacles that you might encounter when working with datasets. This section and subsequent sections introduce you to the following concepts:

- correlation versus causation
- bias-variance tradeoff
- types of bias
- central limit theorem
- statistical inferences

First, keep in mind that statistics typically involves data *samples*, which are subsets of observations of a population. The goal is to find well-balanced samples that provide a good representation of the entire population.

Although this goal can be very difficult to achieve, it is also possible to achieve highly accurate results with a very small sample size. For example, the Harris poll in the United States has been used for decades to analyze political trends. This poll computes percentages that indicate the favorability rating of political candidates, and it is usually within 3.5% of the correct percentage values. What is remarkable about the Harris poll is that its sample size is a mere 4,000 people that are from the U.S. population that is greater than 325,000,000 people.

Another aspect to consider is that each sample has a mean and variance, which do not necessarily equal the mean and variance of the actual population. However, the expected value of the sample mean and variance equal the mean and variance, respectively, of the population.

The Central Limit Theorem

Samples of a population have an interesting property. Suppose that you take a set of samples `{S1, S3, ... , Sn}` of a population and you calculate the mean of those samples, which is `{m1, m2, ..., mn}`. The central limit theorem is a remarkable result: given a set of samples of a population and the mean value of those samples, the distribution of the mean values can be approximated by a Gaussian distribution. Moreover, as the number of samples increases, the approximation becomes more accurate.

Correlation versus Causation

In general, datasets have some features (columns) that are more significant in terms of their set of values, and some features only provide additional information that does not contribute to potential trends in the dataset. For example, the passenger names in the list of passengers on the Titanic are unlikely to affect the survival rate of those passengers, whereas the gender of the passengers is likely to be an important factor.

In addition, a pair of significant features may also be "closely coupled" in terms of their values. For example, a real estate dataset for a set of houses will contain the number of bedrooms and the number of bathrooms for each house in the dataset. As you know, these values tend to increase together and also decrease together. Have you ever seen a house that has ten bedrooms and one bathroom, or a house that has ten bathrooms and one bedroom? If you did find such a house, would you purchase that house as your primary residence?

The extent to which the values of two features change is called their correlation, which is a number between −1 and 1. Two "perfectly" correlated features have a correlation of 1, and two features that are not correlated have a correlation of 0. In addition, if the values of one feature decrease when the values of another feature increase, and vice versa, then their correlation is closer to −1 (and might also equal −1).

However, causation between two features means that the values of one feature can be used to calculate the values of the second feature (within some margin of error).

Keep in mind this fundamental point about machine learning models: they can provide correlation but they cannot provide causation.

Statistical Inferences

Statistical thinking relates processes and statistics, whereas *statistical inference* refers to the process by which the inferences that you make regarding a population. Those inferences are based on statistics that are derived from samples of the population. The validity and reliability of those inferences depend on random sampling in order to reduce bias. There are various metrics that you can calculate to help you assess the validity of a model that has been trained on a particular dataset.

STATISTICAL TERMS RSS, TSS, R^2, AND F1 SCORE

Statistics is extremely important in machine learning, so it is not surprising that many concepts are common to both fields. Machine learning relies on a number of statistical quantities in order to assess the validity of a model, some of which are listed here:

- `RSS`
- `TSS`
- `R^2`

The term *RSS* is the "residual sum of squares" and the term *TSS* is the "total sum of squares." Moreover, these terms are used in regression models.

As a starting point so we can simplify the explanation of the preceding terms, suppose that we have a set of points `{(x1,y1), ..., (xn,yn)}` in the Euclidean plane. In addition, we will define the following quantities:

- `(x,y) is any point in the dataset`
- `y is the y-coordinate of a point in the dataset`
- `y_ is the mean of the y-values of the points in the dataset`
- `y_hat is the y-coordinate of a point on a best-fitting line`

To be clear, `(x,y)` is a point in the *dataset*, whereas `(x,y_hat)` is the corresponding point that lies on the *best fitting line*. With these definitions in mind, the definitions of `RSS`, `TSS`, and `R^2` are listed here (n equals the number of points in the dataset):

```
RSS = (y - y_hat)**2/n
TSS = (y - y_bar)**2/n
R^2 = 1 - RSS/TSS
```

We also have the following inequalities involving `RSS`, `TSS`, and `R^2`:

```
0 <= RSS
RSS <= TSS
0 <= RSS/TSS <= 1
0 <= 1 - RSS/TSS <= 1
0 <= R^2 <= 1
```

When RSS is close to 0, then RSS/TSS is also close to zero, which means that R^2 is close to 1. Conversely, when RSS is close to TSS, then RSS/TSS is close to 1, and R^2 is close to 0. In general, a larger R^2 is preferred (i.e., the model is closer to the data points), but a lower value of R^2 is not necessarily a bad score.

What Is an F1 Score?

In machine learning, an F1 score is for models that are evaluated on a feature that contains categorical data, and the `p-value` is useful for machine learning in general. An F1 score is a measure of the accuracy of a test, and it is defined as the harmonic mean of precision and recall. Here are the relevant formulas, where `p` is the precision and `r` is the recall:

```
p = (True Positive)/(True Positive + False Positive)
r = (True Positive)/(True Positive + False Negative)

F1-score  = 1/[((1/r) + (1/p))/2]
          = 2*[p*r]/[p+r]
```

The best value of an F1 score is 0 and the worse value is 0. Keep in mind that an F1 score is for categorical classification problems, whereas the `R^2` value is typically for regression tasks (such as linear regression).

GINI IMPURITY, ENTROPY, AND PERPLEXITY

These concepts are useful for assessing the quality of a machine learning model and the latter pair are useful for dimensionality reduction algorithms.

Before we discuss the details of Gini impurity, suppose that P is a set of nonnegative numbers {p1, p2, . . . , pn} such that the sum of all the numbers in the set P equals 1. Under these two assumptions, the values in the set P comprise a probability distribution, which we can represent with the letter p.

Now suppose that the set K contains a total of M elements, with k1 elements from class S1, k2 elements from class S2, . . . , and kn elements from class Sn. Compute the fractional representation for each class as follows:

```
p1 = k1/M, p2 = k2/M, ..., pn = kn/M
```

As you can surmise, the values in the set {p1, p2, ..., pn} form a probability distribution. We're going to use the preceding values in the following subsections.

What Is Gini Impurity?

The *Gini impurity* is defined as follows, where {p1,p2,…,pn} is a probability distribution:

```
Gini = 1 – [p1*p1 + p2*p2 + . . . + pn*pn]
     = 1 – SUM pi*pi (for all i, where 1<=i<=n)
```

Since each pi is between 0 and 1, then pi*pi <= pi, which means that:

```
1 = p1 + p2 + . . . + pn
  >= p1*p1 + p2*p2 + . . . + pn*pn
   = Gini impurity
```

Since the Gini impurity is the sum of the squared values of a set of probabilities, the Gini impurity cannot be negative. Therefore, we have derived the following result:

```
0 <= Gini impurity <= 1
```

What Is Entropy?

A formal definition: *entropy* is a measure of the expected ("average") number of bits required to encode the outcome of a random variable. The calculation for the entropy H (the letter E is reserved for Einstein's formula) as defined via the following formula:

```
H = (-1)*[p1*log p1 + p2 * log p2 + . . . + pn * log pn]
  = (-1)* SUM [pi * log(pi)] (for all i, where 1<=i<=n)
```

Calculating Gini Impurity and Entropy Values

For our first example, suppose that we have three classes A and B and a cluster of 10 elements with 8 elements from class A and 2 elements from class B. Therefore p1 and p2 are 8/10 and 2/10, respectively. We can compute the Gini score as follows:

```
Gini = 1 – [p1*p1 + p2*p2]
     = 1 – [64/100 + 04/100]
     = 1 - 68/100
     = 32/100
     = 0.32
```

We can also calculate the entropy for this example as follows:

```
Entropy = (-1)*[p1 * log p1 + p2 * log p2]
        = (-1)*[0.8 * log 0.8 + 0.2 * log 0.2]
        = (-1)*[0.8 * (-0.322) + 0.2 * (-2.322)]
        = 0.8 * 0.322 + 0.2 * 2.322
        = 0.7220
```

For our second example, suppose that we have three classes A, B, C, and a cluster of 10 elements with 5 elements from class A, 3 elements from class B, and 2 elements from class C. Therefore `p1`, `p2`, and `p3` are 5/10, 3/10, and 2/10, respectively. We can compute the Gini score as follows:

```
Gini = 1 – [p1*p1 + p2*p2 + p3*p3]
     = 1 – [25/100 + 9/100 + 04/100]
     = 1 - 38/100
     = 62/100
     = 0.62
```

We can also calculate the entropy for this example as follows:

```
Entropy = (-1)*[p1 * log p1 + p2 * log p2]
        = (-1)*[0.5*log0.5 + 0.3*log0.3 + 0.2*log0.2]
        = (-1)*[-1 + 0.3*(-1.737) + 0.2*(-2.322)]
        = 1 + 0.3*1.737 + 0.2*2.322
        = 1.9855
```

In both examples the Gini impurity is between 0 and 1. However, while the entropy is between 0 and 1 in the first example, it is greater than 1 in the second example (which was the rationale for showing you two examples).

Keep in mind that a set whose elements belong to the same class has Gini impurity equal to 0 and also its entropy equal to 0. For example, if a set has 10 elements that belong to class S1, then:

```
Gini = 1 – SUM pi*pi
     = 1 - p1*p1
     = 1 – (10/10)*(10/10)
     = 1 – 1 = 0

Entropy = (-1)*SUM pi*log pi
        = (-1) * p1*log pi
        = (-1) * (10/10) * log(10/10)
        = (-1)*1*0  = 0
```

Multidimensional Gini Index

The Gini index is a one-dimensional index that works well because the value is uniquely defined. However, when working with multiple factors, we need a multidimensional index. Unfortunately, the multidimensional Gini index (MGI) is not unique defined. While there have been various attempts to define an MGI that has unique values, they tend to be non-intuitive and mathematically much more complex. More information about MGI is here:

https://link.springer.com/chapter/10.1007/978-981-13-1727-9_5

What Is Perplexity?

Suppose that `q` and `p` are two probability distributions, and `{x1, x2, . . . , xN}` is a set of sample values that is drawn from a model whose probability distribution is `p`. In addition, suppose that `b` is a positive integer (it is usually equal to 2). Now define the variable `S` as the following sum (logarithms are in base `b` not 10):

```
S = (-1/N) * [log q(x1) + log q(x2) + . . . + log q(xN)]
  = (-1/N) * SUM log q(xi)
```

The formula for the perplexity PERP of the model `q` is `b` raised to the power `S`, as shown here:

```
PERP = b^S
```

If you compare the formula for entropy with the formula for `S`, you can see that the formulas as similar, so the perplexity of a model is somewhat related to the entropy of a model.

CROSS-ENTROPY AND KL DIVERGENCE

Cross-entropy is useful for understanding machine learning algorithms, and frameworks such as TensorFlow, which supports multiple APIs that involve cross-entropy. *KL divergence* is relevant in machine learning, deep learning, and reinforcement learning.

As an interesting example, consider the credit assignment problem, which involves assigning credit to different elements or steps in a sequence. For example, suppose that users arrive at a Web page by clicking on a previous page, which was also reached by clicking on yet another Web page. Then on the final Web page users click on an ad. How much credit is given to the first and second Web pages for the selected ad? You might be surprised to discover that one solution to this problem involves KL divergence.

What Is Cross-Entropy?

The following formulas for logarithms are presented here because they are useful for the derivation of cross-entropy in this section:

- `log (a * b) = log a + log b`
- `log (a / b) = log a - log b`
- `log (1 / b) = (-1) * log b`

In a previous section you learned that for a probability distribution `P` with values `{p1,p2, ..., pn}`, its entropy is `H` defined as follows:

```
H(P) = (-1)*SUM pi*log(pi)
```

Now we will introduce another probability distribution `Q` whose values are `{q1,q2, ..., qn}`, which means that the entropy `H` of `Q` is defined as follows:

```
H(Q) = (-1)*SUM qi*log(qi)
```

Now we can define the cross-entropy CE of Q and P as follows (notice the `log qi` and `log pi` terms and recall the formulas for logarithms in the previous section):

```
CE(Q,P) = SUM (pi*log qi) - SUM (pi*log pi)
        = SUM (pi*log qi - pi*log pi)
        = SUM pi*(log qi - log pi)
        = SUM pi*(log qi/pi)
```

What Is KL Divergence?

Now that entropy and cross-entropy have been discussed, we can easily define the KL divergence of the probability distributions Q and P as follows:

```
KL(P||Q) = CE(P,Q) - H(P)
```

The definitions of entropy `H`, cross-entropy CE, and KL divergence in this chapter involve discrete probability distributions `P` and `Q`. However, these concepts have counterparts in continuous probability density functions. The mathematics involves the concept of a Lebesgue measure on Borel sets (which is beyond the scope of this book) that are described here:

https://en.wikipedia.org/wiki/Lebesgue_measure

https://en.wikipedia.org/wiki/Borel_set

In addition to KL divergence, there is also Jenson-Shannon divergence, also called JS divergence, which was developed by Johan Jensen and Claude Shannon (who defined the formula for entropy). JS divergence is based on KL divergence, and it has some differences: JS divergence is symmetric and a true metric, whereas KL divergence is neither (as noted in Chapter 4). More information regarding JS divergence is available here:

https://en.wikipedia.org/wiki/Jensen–Shannon_divergence

What Is Their Purpose?

The Gini impurity is often used to obtain a measure of the homogeneity of a set of elements in a decision tree. The entropy of that set is an alternative to its Gini impurity, and you will see both of these quantities used in machine learning models.

The perplexity value in NLP is one way to evaluate language models, which are probability distributions over sentences or texts. This value provides an estimate for the encoding size of a set of sentences.

Cross-entropy is used in various methods in the TensorFlow framework, and the KL divergence is used in various algorithms, such as the dimensionality reduction algorithm t-SNE. For more information about any of these terms, perform an online search and you will find numerous online tutorials that provide more detailed information.

COVARIANCE AND CORRELATION MATRICES

This section explains two important matrices: the covariance matrix and the correlation matrix. Although these are relevant for PCA (principal component analysis) that is discussed later in this chapter, these matrices are not specific to PCA, which is the rationale for discussing them in a separate section. If you are familiar with these matrices, feel free to skim through this section.

The Covariance Matrix

As a reminder, the statistical quantity called the variance of a random variable X is defined as follows:

```
variance(x) = [SUM (x - xbar)*(x-xbar)]/n
```

A covariance matrix C is an nxn matrix whose values on the main diagonal are the variance of the variables X1, X2, . . . , Xn. The other values of C are the covariance values of each pair of variables Xi and Xj.

The formula for the covariance of the variables X and Y is a generalization of the variance of a variable, and the formula is shown here:

```
covariance(X, Y) = [SUM (x - xbar)*(y-ybar)]/n
```

Notice that you can reverse the order of the product of terms (multiplication is commutative), and therefore the covariance matrix C is a symmetric matrix:

```
covariance(X, Y) = covariance(Y,X)
```

Suppose that a comma separated values (CSV) file contains four numeric features, all of which have been scaled appropriately, and we will call them x1, x2, x3, and x4. Then the covariance matrix C is a 4x4 square matrix that is defined with the following entries (pretend that there are outer brackets on the left side and the right side to indicate a matrix):

```
cov(x1,x1) cov(x1,x2) cov(x1,x3) cov(x1,x4)
cov(x2,x1) cov(x2,x2) cov(x2,x3) cov(x2,x4)
cov(x3,x1) cov(x3,x2) cov(x3,x3) cov(x3,x4)
cov(x4,x1) cov(x4,x2) cov(x4,x3) cov(x4,x4)
```

Note that the following is true for the diagonal entries in the preceding covariance matrix C:

```
var(x1,x1) = cov(x1,x1)
var(x2,x2) = cov(x2,x2)
var(x3,x3) = cov(x3,x3)
var(x4,x4) = cov(x4,x4)
```

In addition, C is a symmetric matrix, which is to say that the transpose of matrix C (rows become columns and columns become rows) is identical to the matrix C. The latter is true because (as you saw in the previous section) cov(x,y) = cov(y,x) for any feature x and any feature y.

Covariance Matrix: An Example

Suppose we have the two-column matrix A defined as follows:

```
        x   y
A = | 1   1  | <= 6x2 matrix
    | 2   1  |
    | 3   2  |
    | 4   2  |
    | 5   3  |
    | 6   3  |
```

The mean x_bar of column x is (1+2+3+4+5+6)/6 = 3.5, and the mean y_bar of column y is (1+1+2+2+3+3)/6 = 2. Now subtract x_bar from column x and subtract y_bar from column y and we get matrix B, as shown here:

```
B = | -2.5 -1  | <= 6x2 matrix
    | -1.5 -1  |
    | -0.5  0  |
    |  0.5  0  |
    |  1.5  1  |
    |  2.5  1  |
```

Let Bt indicate the transpose of the matrix B (i.e., switch columns with rows and rows with columns) which means that Bt is a 2 x 6 matrix, as shown here:

```
Bt = |-2.5 -1.5 -0.5 0.5, 1.5, 2.5|
     |-1   -1    0   0    1    1  |
```

The covariance matrix C is the product of Bt and B, as shown here:

```
C = Bt * B = | 15.25 4 |
             |  4    8 |
```

Note that if the units of measure of features x and y do not have a similar scale, then the covariance matrix is adversely affected. In this case, the solution is simple: use the correlation matrix, which defined in the next section.

The Correlation Matrix

As you learned in the preceding section, if the units of measure of features x and y do not have a similar scale, then the covariance matrix is adversely affected. The solution involves the correlation matrix, which equals the covariance values cov(x,y) divided by the standard deviation stdx and stdy of x and y, respectively, as shown here:

```
corr(x,y) = cov(x,y)/[stdx * stdy]
```

The correlation matrix no longer has units of measure, and we can use this matrix to find the eigenvalues and eigenvectors.

Now that you understand how to calculate the covariance matrix and the correlation matrix, you are ready for an example of calculating eigenvalues and eigenvectors.

Eigenvalues and Eigenvectors

According to a well-known theorem in mathematics (whose proof you can find online), the eigenvalues of a real-valued symmetric matrix are real numbers. Consequently, the eigenvectors of `C` are vectors in a Euclidean vector space (not a complex vector space).

Before we continue, a non-zero vector `x'` is an eigenvector of the matrix `C` if there is a nonzero scalar lambda such that `C*x' = lambda * x'.`

Now suppose that the eigenvalues of `C` are `b1`, `b2`, `b3`, and `b4`, in decreasing numeric order from left-to-right, and that the corresponding eigenvectors of `C` are the vectors `w1`, `w2`, `w3`, and `w4`. Then the matrix M that consists of the column vectors `w1`, `w2`, `w3`, and `w4` represents the principal components.

CALCULATING EIGENVECTORS: A SIMPLE EXAMPLE

As a simple illustration of calculating eigenvalues and eigenvectors, suppose that the square matrix C is defined as follows:

```
C = | 1  3 |
    | 3  1 |
```

Let `I` denote the 2 × 2 identity matrix, and let `b'` be an eigenvalue of `C`, which means that there is an eigenvector `x'` such that:

```
C* x' = b' * x', or
(C-b*I)*x' = 0 (the right side is a 2x1 vector)
```

Since `x'` is nonzero, that means the following is true (where `det` refers to the *determinant* of a matrix):

```
det(C-b*I) = det |1-b 3   | = (1-b)*(1-b)-9 = 0
                 |3   1-b|
```

We can expand the quadratic equation in the preceding line to get:

```
det(C-b*I) = (1-b)*(1-b) - 9
           = 1 - 2*b + b*b - 9
           = -8 - 2*b + b*b
           = b*b - 2*b - 8
```

Use the quadratic formula (or perform factorization by visual inspection) to determine that the solution for `det(C-b*I) = 0 is b = -2 or b = 4.` Next, substitute `b` = -2 into `(C-b*I)x' = 0` and we get the following result:

```
|1-(-2) 3      | |x1| = |0|
|3      1-(-2)| |x2|    |0|
```

The preceding reduces to the following identical equations:

```
3*x1 + 3*x2 = 0
3*x1 + 3*x2 = 0
```

The general solution is `x1 = -x2`, and we can choose any nonzero value for `x2`, so we will set `x2` = 1 (any nonzero value will do just fine), which yields `x1` = -1. Therefore, the eigenvector [-1, 1] is associated with the eigenvalue -2. In a similar fashion, if `x'` is an eigenvector whose eigenvalue is 4, then [1, 1] is an eigenvector.

Notice that the eigenvectors [-1, 1] and [1, 1] are orthogonal because their inner product is zero, as shown here:

```
[-1,1] * [1,1] = (-1)*1 + (1)*1 = 0
```

In fact, the set of eigenvectors of a square matrix (whose eigenvalues are real) are always orthogonal, regardless of the dimensionality of the matrix.

Gauss Jordan Elimination (Optional)

This simple technique enables you to find the solution to systems of linear equations "in place," which involves a sequence of arithmetic operations to transform a given matrix to an identity matrix.

The following example combines the Gauss-Jordan elimination technique (which finds the solution to a set of linear equations) with the *bookkeeper's method*, which determines the inverse of an invertible matrix (its determinant is nonzero).

This technique involves two adjacent matrices: the left-side matrix is the initial matrix and the right-side matrix is an identity matrix. Next, perform various linear operations on the left-side matrix to reduce it to an identity matrix: the matrix on the right side equals its inverse. For example, consider the following pair of linear equations whose solution is x = 1 and y = 2:

```
2*x + 2*y = 6
4*x - 1*y = 2
```

Step 1: create a 2×2 matrix with the coefficients of `x` in column 1 and the coefficients of `y` in column two, followed by the 2×2 identity matrix, and finally a column from the numbers on the right of the equals sign:

```
| 2  2 | 1 0 | 6|
| 4 -1 | 0 1 | 2|
```

Step 2: add (-2) times the first row to the second row:

```
| 2  2  | 1  0 |6   |
| 0  -5 | -2 1 |-10|
```

Step 3: divide the second row by 5:

```
| 2   2  | 1      0 |6    |
| 0  -1  | -2/5 1/5 |-10/5|
```

Step 4: add 2 times the second row to the first row:

```
| 2   0  |   1/5 2/5 |2 |
| 0  -1  | -2/5 1/5 |-2|
```

Step 5: divide the first row by 2:

```
| 1   0  | -2/10 2/10 |1 |
| 0  -1  | -2/5   1/5 |-2|
```

Step 6: multiply the second row by (-1):

```
| 1   0  | -2/10 2/10 |1|
| 0   1  |   2/5  -1/5 |2|
```

As you can see, the left-side matrix is the 2×2 identity matrix, the right-side matrix is the inverse of the original matrix, and the right-most column is the solution to the original pair of linear equations (x = 1 and y = 2).

PCA (PRINCIPAL COMPONENT ANALYSIS)

PCA is a linear dimensionality reduction technique for determining the most important features in a dataset. This section discusses PCA because it is a very popular technique that you will encounter frequently. Other techniques are more efficient than PCA, so later on it is worthwhile to learn other dimensionality reduction techniques as well.

Keep in mind the following points regarding the PCA technique:

- PCA is a variance-based algorithm.
- PCA creates variables that are linear combinations of the original variables.
- The new variables are all pair-wise orthogonal.
- PCA can be a useful preprocessing step before clustering.
- PCA is generally preferred for data reduction.

PCA can be useful for variables that are strongly correlated. If most of the coefficients in the correlation matrix are smaller than 0.3, PCA is not helpful. PCA provides some advantages: less computation time for training a model (for example, using only five features instead of one hundred features), a simpler model, and the ability to render the data visually when two or three features are selected. Here is a key point about PCA:

PCA calculates the eigenvalues and the eigenvectors of the covariance (or correlation) matrix C.

If you have four or five components, you will not be able to display them visually, but you could select subsets of three components for visualization, and perhaps gain some additional insight into the dataset.

The PCA algorithm involves the following sequence of steps:

1. Calculate the correlation matrix (from the covariance matrix) C of a dataset.
2. Find the eigenvalues of C.
3. Find the eigenvectors of C.
4. Cnstruct a new matrix that comprises the eigenvectors.

The covariance matrix and correlation matrix were explained in a previous section. You also saw the definition of eigenvalues and eigenvectors, along with an example of calculating eigenvalues and eigenvectors.

The eigenvectors are treated as column vectors that are placed adjacent to each other in decreasing order (from left-to-right) with respect to their associated eigenvectors.

PCA uses the variance as a measure of information: the higher the variance, the more important the component. In fact, just to jump ahead slightly: PCA determines the eigenvalues and eigenvectors of a covariance matrix (discussed in a previous section), and constructs a new matrix whose columns are eigenvectors, ordered from left-to-right in a sequence that matches the corresponding sequence of eigenvalues: the left-most eigenvector has the largest eigenvalue, the next eigenvector has the second-largest eigenvalue, and continuing in this fashion until the right-most eigenvector (which has the smallest eigenvalue).

Alternatively, there is an interesting theorem in linear algebra: if `C` is a symmetric matrix, then there is a diagonal matrix `D` and an orthogonal matrix `P` (the columns are pair-wise orthogonal, which means their pair-wise inner product is zero), such that the following holds:

```
C = P * D * Pt (where Pt is the transpose of matrix P)
```

In fact, the diagonal values of `D` are eigenvalues, and the columns of `P` are the corresponding eigenvectors of the matrix `C`.

Fortunately, we can use `NumPy` and `Pandas` to calculate the mean, standard deviation, covariance matrix, correlation matrix, as well as the matrices `D` and `P` in order to determine the eigenvalues and eigenvectors.

A point of interest is that any positive definite square matrix has real-valued eigenvectors, which also applies to the covariance matrix `C` because it is a real-valued symmetric matrix.

The New Matrix of Eigenvectors

The previous section described how the matrices `D` and `P` are determined. The left-most eigenvector of `D` has the largest eigenvalue, the next eigenvector has the second-largest eigenvalue, and so forth. Therefore, the eigenvector with the highest eigenvalue is the principal component of the dataset. The eigenvector with the second-highest eigenvalue is the second principal component, and so forth. You specify the number of principal components that you want via the `n_components` hyperparameter in the PCA class of Sklearn (discussed briefly in Chapter 7).

As a simple and minimalistic example, consider the following code block that uses PCA for a (somewhat contrived) dataset:

```
import numpy as np
from sklearn.decomposition import PCA
data = np.array([[-1,-1], [-2,-1], [-3,-2], [1,1], [2,1], [3,2]])
pca = PCA(n_components=2)
pca.fit(X)
```

Note that a trade-off here: we greatly reduce the number of components, which reduces the computation time and the complexity of the model, but we also lose some accuracy. However, if the unselected eigenvalues are small, we lose only a small amount of accuracy.

Now we will use the following notation:

NM denotes the matrix with the new principal components.

NMt is the transpose of NM.

PC is the matrix of the subset of selected principal components.

SD is the matrix of scaled data from the original dataset.

SDt is the transpose of SD.

Then the matrix NM is calculated via the following formula:

```
NM = PCt * SDt
```

Although PCA is a very nice technique for dimensionality reduction, keep in mind the following limitations of PCA:

- It is less suitable for data with nonlinear relationships.
- It is less suitable for special classification problems.

A related algorithm is called kernel PCA, which is an extension of PCA that introduces a nonlinear transformation so you can still use the PCA approach.

WELL-KNOWN DISTANCE METRICS

There are several similarity metrics available, such as item similarity metrics, `Jaccard` (user-based) similarity, and cosine similarity (which is used to compare vectors of numbers). The following subsections introduce you to these similarity metrics.

Another well-known distance metric is the so-called "taxicab" metric, which is also called the Manhattan distance metric. Given two points A and B in a rectangular grid, the taxicab metric calculates the distance between two points by counting the number of "blocks" that must be traversed in order to reach B from A (the other direction has the same taxicab metric value). For example, if you need to travel two blocks north and then three blocks east in a rectangular grid, then the Manhattan distance is 5.

In fact, there are various other metrics available, which you can learn about by searching Wikipedia. In the case of NLP, the most commonly used distance metric is calculated via the cosine similarity of two vectors, and it is derived from the formula for the inner ("dot") product of two vectors.

Pearson Correlation Coefficient

Pearson correlation coefficient is the `Pearson` correlation between two vectors. Given random variables `X` and `Y`, and the following terms:

```
std(X)   = standard deviation of X
std(Y)   = standard deviation of Y
cov(X,Y) = covariance of X and Y
```

Then the `Pearson` correlation coefficient `rho(X,Y)` is defined as follows:

```
               cov(X,Y)
rho(X,Y) =  -------------
             std(X)*std(Y)
```

Keep in mind that the `Pearson` correlation coefficient is limited to items of the same type. More information about the `Pearson` correlation coefficient is here:

https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Jaccard Index (or Similarity)

The `Jaccard` similarity is based on the number of users which have rated item A and B divided by the number of users who have rated either A or B. `Jaccard` similarity is based on unique words in a sentence and is unaffected by duplicates, whereas cosine similarity is based on the length of all word vectors (which changes when duplicates are added). The choice between cosine similarity and `Jaccard` similarity depends on whether or not word duplicates are important.

The following `Python` method illustrates how to compute the `Jaccard` similarity of two sentences:

```
def get_jaccard_sim(str1, str2):
  set1 = set(str1.split())
  set2 = set(str2.split())
  set3 = set1.intersection(set2)
  # (size of intersection) / (size of union):
  return float(len(set3)) / (len(set1) + len(set2) - len(set3))
```

`Jaccard` similarity can be used in situations involving Boolean values, such as product purchases (true/false), instead of numeric values. More information is available here:

https://en.wikipedia.org/wiki/Jaccard_index

Local Sensitivity Hashing (Optional)

If you are familiar with hash algorithms, you know that they are algorithms that create a hash table that associate items with a value. The advantage of hash tables is that the lookup time to determine whether or not an item exists in the hash table is constant. Of course, it is possible for two items to "collide," which means that they both occupy the same bucket in the hash table. In this case, a bucket can consist of a list of items that can be searched in more or less constant time. If there are too many items in the same bucket, then a different hashing function can be selected to reduce the number of collisions. The goal of a hash table is to minimize the number of collisions.

The local sensitivity hashing (LSH) algorithm hashes similar input items into the same "buckets." In fact, the goal of LSH is to maximize the number of collisions, whereas traditional hashing algorithms attempt to minimize the number of collisions.

Since similar items end up in the same buckets, LSH is useful for data clustering and nearest neighbor searches. Moreover, LSH is a dimensionality reduction technique that places data points of high dimensionality closer together in a lower-dimensional space, while simultaneously preserving the relative distances between those data points.

More details about LSH are available here:

https://en.wikipedia.org/wiki/Locality-sensitive_hashing

TYPES OF DISTANCE METRICS

Nonlinear dimensionality reduction techniques can also have different distance metrics. For example, linear reduction techniques can use the Euclidean distance metric (based on the Pythagorean theorem). However, you need to use a different distance metric to measure the distance between two points on a sphere (or some other curved surface). In the case of NLP, the cosine similarity metric is used to measure the distance between word embeddings (which are vectors of floating point numbers that represent words or tokens).

Distance metrics are used for measuring physical distances, and some well-known distance metrics are as follows:

- Euclidean distance
- Manhattan distance
- Chebyshev distance

The Euclidean algorithm also obeys the "triangle inequality," which states that for any triangle in the Euclidean plane, the sum of the lengths of any pair of sides must be greater than the length of the third side.

In spherical geometry, you can define the distance between two points as the arc of a great circle that passes through the two points (always selecting the smaller of the two arcs when they are different).

In addition to physical metrics, the following algorithms implement the concept of *edit distance* (the distance between strings):

- Hamming distance
- Jaro–Winkler distance
- Lee distance
- Levenshtein distance
- Mahalanobis distance metric
- Wasserstein metric

The Mahalanobis metric is based on an interesting idea: given a point P and a probability distribution D, this metric measures the number of standard deviations that separate point P from distribution D. More information about Mahalanobis is available here:

https://en.wikipedia.org/wiki/Mahalanobis_distance

In the branch of mathematics called topology, a metric space is a set for which distances between all members of the set are defined. Various metrics are available (such as the Hausdorff metric), depending on the type of topology.

The Wasserstein metric measures the distance between two probability distributions over a metric space X. This metric is also called the *earth mover's metric* for the following reason: given two unit piles of dirt, it is the measure of the minimum cost of moving one pile on top of the other pile.

KL divergence bears some superficial resemblance to the Wasserstein metric. However, there are some important differences between them. Specifically, the Wasserstein metric has the following properties:

1. It is a metric.
2. It is symmetric.
3. It satisfies the triangle inequality.

Constrastingly, KL divergence has the following properties:

1. It is not a metric (it is a divergence).
2. It is not symmetric: KL(P, Q) != KL(Q, P).
3. It does not satisfy the triangle inequality.

Note that JS (Jenson-Shannon) divergence (which is based on KL divergence) is a true metric, which would enable a more meaningful comparison with other metrics (such as the Wasserstein metric). A comparison of the Wasserstein metric and KL divergence can be found here:

https://stats.stackexchange.com/questions/295617/what-is-the-advantages-of-wasserstein-metric-compared-to-kullback-leibler-diverg

More information regarding the Wasstertein metric is available here: *https://en.wikipedia.org/wiki/Wasserstein_metric*

WHAT IS BAYESIAN INFERENCE?

Bayesian inference is an important technique in statistics that involves statistical inference and Bayes' theorem to update the probability for a hypothesis as more information becomes available. Bayesian inference is often called Bayesian probability, and it is important in dynamic analysis of sequential data.

Bayes' Theorem

Given two sets A and B, we will define the following numeric values (all of them are between 0 and 1):

```
P(A) = probability of being in set A
P(B) = probability of being in set B
P(Both) = probability of being in A intersect B
P(A|B) = probability of being in A (given you're in B)
P(B|A) = probability of being in B (given you're in A)
```

Then the following formulas are also true:

```
P(A|B) = P(Both)/P(B) (#1)
P(B|A) = P(Both)/P(A) (#2)
```

Multiply the preceding pair of equations by the term that appears in the denominator and we get these equations:

```
P(B)*P(A|B) = P(Both) (#3)
P(A)*P(B|A) = P(Both) (#4)
```

Now set the left-side of equations #3 and #4 equal to each another and that gives us this equation:

```
P(B)*P(A|B) = P(A)*P(B|A) (#5)
```

Divide both sides of #5 by P(B) and we get this well-known equation:

```
P(A|B) = P(A)*P(A|B)/P(B) (#6)
```

Some Bayesian Terminology

In the previous section, we derived the following relationship:

```
P(h|d) = (P(d|h) * P(h)) / P(d)
```

There is a name for each of the four terms in the preceding equation, as discussed below.

First, the *posterior probability* is `P(h|d)`, which is the probability of hypothesis `h` given the data `d`.

Second, `P(d|h)` is the probability of data `d` given that the hypothesis `h` was true.

Third, the *prior probability* of `h` is `P(h)`, which is the probability of hypothesis `h` being true (regardless of the data).

Finally, `P(d)` is the probability of the data (regardless of the hypothesis)

We are interested in calculating the posterior probability of P(h|d) from the prior probability p(h) with P(D) and P(d|h).

What Is MAP?

The maximum a posteriori (MAP) hypothesis is the hypothesis with the highest probability, which is the maximum probable hypothesis. This can be written as follows:

```
MAP(h) = max(P(h|d))
```

or:

```
MAP(h) = max((P(d|h) * P(h)) / P(d))
```

or:

```
MAP(h) = max(P(d|h) * P(h))
```

Why Use Bayes' Theorem?

Bayes' theorem describes the probability of an event based on the prior knowledge of the conditions that might be related to the event. If we know the conditional probability, we can use Bayes' rule to find out the reverse probabilities. The previous statement is the general representation of the Bayes' rule.

SUMMARY

This chapter started with a discussion of probability, expected values, and the concept of a random variable. Then you learned about some basic statistical concepts, such as mean, median, mode, variance, and standard deviation. Next, you learned about the terms RSS, TSS, R^2, and F1 score. In addition, you got an introduction to the concepts of skewness, kurtosis, Gini impurity, entropy, perplexity, cross-entropy, and KL divergence.

Next, you learned about covariance and correlation matrices and how to calculate eigenvalues and eigenvectors. Then you were introduced to the dimensionality reduction technique known as PCA (principal component analysis), after which you learned about Bayes' theorem.

CHAPTER 2

WORKING WITH DATA

This chapter introduces you to various data types that you will encounter in datasets, how to scale data values, techniques for detecting outliers, and several ways for handling missing data values.

The first part of this chapter contains an overview of different types of data, and an explanation of how to normalize and standardize a set of numeric values by calculating the mean and standard deviation of a set of numbers. You will see how to map categorical data to a set of integers and how to perform a one-hot encoding.

The second part of this chapter discusses outliers, anomalies, missing data, and various techniques for handling these scenarios. The third section discusses imbalanced data and several techniques, such as `SMOTE`, to deal with imbalanced classes in a dataset.

The fourth section contains details regarding the bias-variance tradeoff and various types of statistical bias, and also discusses ways to evaluate classifiers, such as `LIME` and `ANOVA`.

As you will soon see, this chapter provides a high-level view of concepts that will help you work with datasets that require preprocessing before using them to train machine learning models. While the code samples reference APIs from `Python` libraries (such as `Numpy` and `Pandas`), the APIs are intuitive: `mean()` for calculating the mean of a set of numbers, `std()` for calculating the standard deviation of a set of numbers.

However, the code sample that involves `Sklearn` is marked “optional” because it uses the `EllipticEnvelope` class in `sklearn.covariance`, whose functionality is not intuitive (yet good to be aware of for future study).

DEALING WITH DATA: WHAT CAN GO WRONG?

In a perfect world, all datasets are in pristine condition, with no extreme values, no missing values, and no erroneous values. Every feature value is captured correctly, with no chance for any confusion. Moreover, no conversion is required between date formats, currency values, or languages because of the one universal standard that defines the correct formats and acceptable values for every possible set of data values.

Although the preceding scenario can occur, it's more likely that you need techniques to process data in order to handle various types of inconsistencies. Even after you manage to create a clean and robust dataset, other issues can arise, such as data drift that is described in the next section.

In fact, the task of cleaning data is not necessarily complete even after a machine learning model is deployed to a production environment. For instance, an online system that gathers terabytes or petabytes of data on a daily basis can contain skewed values that in turn adversely affect the performance of the model. Such adverse effects can be revealed through the changes in the metrics that are associated with the production model.

What Is Data Drift?

The value of data is based on its accuracy, its relevance, and its age. *Data drift* refers to data that has become less relevant—in some cases this happens over a period of time, and in other cases it is because some data is no longer relevant because of feature-related changes in an application.

For example, online purchasing patterns in 2010 are probably not as relevant as data from 2020 because of various factors (such as the profile of different types of customers). Another example involves an inventory of cell phones: discontinued models have a diminished value in such a system. Keep in mind that there might be multiple factors that can influence data drift in a specific dataset.

Two techniques for handling data drift are *domain classifier* and the *black-box shift* detector, both of which are discussed here:

https://blog.dataiku.com/towards-reliable-mlops-with-drift-detectors

Data drift is one of three types of drift, and all three types are listed here:

- concept drift
- data drift
- upstream data changes

Perform an online search to find more information about these types of drift.

WHAT ARE DATASETS?

In simple terms, a dataset is a source of data (such as a text file) that contains rows and columns of data. Each row is typically called a "data point," and each column is called a "feature." A dataset can be a `CSV` (comma separated values), `TSV` (tab separated values), `Excel` spreadsheet, a table in an `RDMBS`, a document in a `NoSQL` database, the output from a Web Service, and so forth. As you will see, someone needs to analyze the dataset to determine which features are the most important and which features can be safely ignored in order to train a model with the given dataset.

A dataset can vary from very small (a couple of features and 100 rows) to very large (more than 1,000 features and more than one million rows). If you are unfamiliar with the problem domain, then you might struggle to determine the most important features in a large dataset. In this situation, you might need a "domain expert" who understands the importance of the features, their inter-dependencies (if any), and whether or not the data values for the features are valid.

In addition, there are algorithms (called dimensionality reduction algorithms) that can help you determine the most important features, such as `PCA` (principal component analysis), which is outside the scope of this book.

Data Preprocessing

Data preprocessing is the initial step that involves validating the contents of a dataset, which involves making decisions about missing and incorrect data values:

- dealing with missing data values
- cleaning "noisy" text-based data
- removing HTML tags
- dealing with emojis/emoticons
- filtering data
- grouping data
- handling currency and date formats

Cleaning data is a subset of data wrangling that involves removing unwanted data as well as handling missing data. In the case of text-based data, you might need to remove `HTML` tags, punctuation, and so forth. In the case of numeric data, it is possible that alphabetic characters are mixed together with numeric data. However, a dataset with numeric features might have incorrect values or missing values (discussed later). In addition, calculating the minimum, maximum, mean, median, and standard deviation of the values of a feature obviously pertain only to numeric values.

After the preprocessing step is completed, data wrangling is performed. *Data wrangling* refers to the transformation of data into a new format. For example, you might have to combine data from multiple sources into a single dataset. Perhaps you need to convert between different units of measurement (e.g., date formats, currency values, etc.) so that the data values can be represented in a consistent manner in a dataset.

In case you didn't already know, currency and data values are part of something that is called `i18n` (internationalization), whereas `L10n` (localization) targets a specific nationality, language, or region. Hard-coded values (such as text strings) can be stored as resource strings in a file that's often called a resource bundle, where each string is referenced via a code. Each language has its own resource bundle.

DATA TYPES

If you have written computer programs, then you know that explicit data types exist in many programming languages, such as C, C++, Java, TypeScript, and so forth. Some programming languages, such as JavaScript and awk, do not require initializing variables with an explicit type: the type of a variable is inferred dynamically via an implicit type system (i.e., one that is not directly exposed to a developer).

In machine learning, datasets can contain features that have different types of data, such as a combination of one or more of the following:

- numeric data (integer/floating point and discrete/continuous)
- character/categorical data (different languages)
- date-related data (different formats)
- currency data (different formats)
- binary data (yes/no, 0/1, and so forth)
- nominal data (multiple unrelated values)
- ordinal data (multiple and related values)

Consider a dataset that contains real estate data, which can have as many as 30 columns (or even more), often with the following features:

- the number of bedrooms in a house: numeric value and a discrete value
- the number of square feet: a numeric value and (probably) a continuous value
- the name of the city: character data
- the construction date: a date value
- the selling price: a currency value and probably a continuous value
- the "for sale" status: binary data (either "yes" or "no")

An example of nominal data is the seasons in a year: although many countries have four distinct seasons, some countries have two distinct seasons. However, keep in mind that seasons can be associated with different temperature ranges (summer versus winter). An example of ordinal data is an employee pay grade: 1 = entry level, 2 = one year of experience, and so forth. Another example of nominal data is a set of colors, such as {Red, Green, Blue}.

An example of binary data is the pair {Male, Female}, and some datasets contain a feature with these two values. If such a feature is required for training a model, first convert {Male, Female} to a numeric counterpart, such as {0, 1}. Similarly, if you need to include a feature whose values are the previous set of colors, you can replace {Red, Green, Blue} with the values {0, 1, 2}. Categorical data is discussed in more detail later in this chapter.

PREPARING DATASETS

If you have the good fortune to inherit a dataset that is in pristine condition, then data cleaning tasks (discussed later) are vastly simplified: in fact, it might not be necessary to perform *any* data cleaning for the dataset. However, if you need to create a dataset that combines data from multiple datasets that contain different formats for dates and currency, then you need to perform a conversion to a common format.

If you need to train a model that includes features that have categorical data, then you need to convert that categorical data to numeric data. For instance, the Titanic dataset contains a feature called "gender," which is either male or female. As you will see later in this chapter, Pandas makes it extremely simple to "map" male to 0 and female to 1.

Discrete Data versus Continuous Data

As a simple rule of thumb, discrete data is a set of values that can be counted whereas continuous data must be measured. Discrete data can reasonably fit in a drop-down list of values, but there is no exact value for making such a determination. One person might think that a list of 500 values is discrete, whereas another person might think it is continuous.

For example, the list of provinces of Canada and the list of states of the United States are discrete data values, but is the same true for the number of countries in the world (roughly 200) or for the number of languages in the world (more than 7,000)?

Alternatively, values for temperature, humidity, and barometric pressure are considered continuous. Currency is also treated as continuous, even though there is a measurable difference between two consecutive values. The smallest unit of currency for U.S. currency is one penny, which is 1/100th of a dollar (accounting-based measurements use the "mil," which is 1/1,000th of a dollar).

Continuous data types can have subtle differences. For example, someone who is 200 centimeters tall is twice as tall as someone who is 100 centimeters tall; similarly for 100 kilograms versus 50 kilograms. However, temperature is different: 80° Fahrenheit is not twice as hot as 40° Fahrenheit.

Furthermore, keep in mind that the word *continuous* has a different meaning in mathematics, and the definition is not necessarily the same as *continuous* in machine learning. In the former, a continuous variable (e.g., in the 2D Euclidean plane) can have an uncountably infinite number of values. On the other hand, a feature in a dataset that can have more values that can be "reasonably" displayed in a drop down list is treated *as though* it is a continuous variable.

For instance, values for stock prices are discrete: they must differ by at least a penny (or some other minimal unit of currency), which is to say, it is meaningless to say that the stock price changes by one-millionth of a penny. However, since there are "so many" possible stock values, it is treated as a continuous variable. The same comments apply to car mileage, ambient temperature, barometric pressure, and so forth.

"Binning" Continuous Data

With the previous section in mind, the concept of *binning* refers to subdividing a set of values into multiple intervals, and then treating all the numbers in the same interval as though they had the same value.

As a simple example, suppose that a feature in a dataset contains the age of people in a dataset. The range of values is approximately between 0 and 120, and we could "bin" them into 12 equal intervals, where each consists of 10 values: 0 through 9, 10 through 19, 20 through 29, and so forth.

However, partitioning the values of people's age as described in the preceding paragraph can be problematic. Suppose that person A, person B, and person C are 29, 30, and 39, respectively. Then person A and person B are probably more similar to each other than person B and person C, but because of the way in which the ages are partitioned, B is classified as closer to C than to A. In fact, binning can increase Type I errors (false positive) and Type II errors (false negative), as discussed in this blog post (along with some alternatives to binning):

https://medium.com/@peterflom/why-binning-continuous-data-is-almost-always-a-mistake-ad0b3a1d141f

As another example, using quartiles is even more coarse-grained than the earlier age-related binning example. The issue with binning pertains to the consequences of classifying people in different bins, even though they are in close proximity to each other. For instance, some people struggle financially because they earn a meager wage, and they are disqualified from financial assistance because their salary is higher than the cut-off point for receiving any assistance.

Scaling Numeric Data via Normalization

A range of values can vary significantly and it is important to note that they often need to be scaled to a smaller range, such as values in the range [−1, 1] or [0, 1], which you can do via the `tanh` function or the `sigmoid` function, respectively.

For example, measuring a person's height in meters involves a range of values between 0.50 meters and 2.5 meters (in the vast majority of cases), whereas measuring height in centimeters ranges between 50 centimeters and 250 centimeters: these two units differ by a factor of 100. A person's weight in kilograms generally varies between 5 kilograms and 200 kilograms, whereas measuring weight in grams differs by a factor of 1,000. Distances between objects can be measured in meters or in kilometers, which also differ by a factor of 1,000.

In general, use units of measure so that the data values in multiple features all belong to a similar range of values. In fact, some machine learning algorithms require scaled data, often in the range of [0, 1] or [−1, 1]. In addition to the `tanh` and sigmoid function, there are other techniques for scaling data, such as "standardizing" data (think Gaussian distribution) and "normalizing" data (linearly scaled so that the new range of values is in [0, 1]).

The following examples involve a floating point variable `X` with different ranges of values that will be scaled so that the new values are in the interval [0, 1].

Example 1: if the values of `X` are in the range [0, 2], then `X`/2 is in the range [0, 1].

Example 2: if the values of `X` are in the range [3, 6], then `X`-3 is in the range [0, 3], and (`X`-3)/3 is in the range [0, 1].

Example 3: if the values of `X` are in the range [−10, 20], then `X` +10 is in the range [0, 30], and (`X` +10)/30 is in the range of [0, 1].

In general, suppose that `X` is a random variable whose values are in the range `[a,b]`, where `a < b`. You can scale the data values by performing two steps:

```
Step 1: X-a is in the range [0,b-a]
Step 2: (X-a)/(b-a) is in the range [0,1]
```

If `X` is a random variable that has the values `{x1, x2, x3, . . . , xn}`, then the formula for normalization involves mapping each `xi` value to `(xi – min)/(max – min)`, where `min` is the minimum value of `X` and `max` is the maximum value of X.

As a simple example, suppose that the random variable `X` has the values `{-1, 0, 1}`. Then `min` and `max` are 1 and −1, respectively, and the normalization of `{-1, 0, 1}` is the set of values `{(-1-(-1))/2, (0-(-1))/2, (1-(-1))/2}`, which equals `{0, 1/2, 1}`.

Scaling Numeric Data via Standardization

The standardization technique involves finding the mean `mu` and the standard deviation `sigma`, and then mapping each `xi` value to `(xi – mu)/sigma`. Recall the following formulas:

```
mu = [SUM (x) ]/n
variance(x) = [SUM (x – xbar)*(x-xbar)]/n
sigma = sqrt(variance)
```

As a simple illustration of standardization, suppose that the random variable `X` has the values {−1, 0, 1}. Then `mu` and `sigma` are calculated as follows:

```
mu       = (SUM xi)/n = (-1 + 0 + 1)/3 = 0

variance = [SUM (xi- mu)^2]/n
         = [(-1-0)^2 + (0-0)^2 + (1-0)^2]/3
         = 2/3

sigma    = sqrt(2/3) = 0.816 (approximate value)
```

Therefore, the standardization of `{-1, 0, 1}` is `{-1/0.816, 0/0.816, 1/0.816}`, which in turn equals the set of values `{-1.2254, 0, 1.2254}`.

As another example, suppose that the random variable `X` has the values `{-6, 0, 6}`. Then `mu` and `sigma` are calculated as follows:

```
mu       = (SUM xi)/n = (-6 + 0 + 6)/3 = 0

variance = [SUM (xi- mu)^2]/n
         = [(-6-0)^2 + (0-0)^2 + (6-0)^2]/3
         = 72/3
         = 24

sigma    = sqrt(24) = 4.899 (approximate value)
```

Therefore, the standardization of `{-6, 0, 6}` is `{-6/4.899, 0/4.899, 6/4.899}`, which in turn equals the set of values `{-1.2247, 0, 1.2247}`.

In the preceding two examples, the mean equals 0 in both cases but the variance and standard deviation are significantly different. One other point: the normalization of a set of values *always* produces a set of numbers between 0 and 1.

Alternately, the standardization of a set of values can generate numbers that are less than –1 and greater than 1: this will occur when `sigma` is less than the minimum value of every term `|mu - xi|`, where the latter is the absolute value of the difference between `mu` and each `xi` value. In the preceding example, the minimum difference equals 1, whereas `sigma` is 0.816, and therefore the largest standardized value is greater than 1.

Scaling Numeric Data via Robust Standardization

The *robust standardization* technique is a variant of standardization that computes the mean `mu` and the standard deviation `sigma` based on a subset of values. Specifically, use only the values that are between the 25th percentile and 75th percentile, which ignores the first and fourth quartiles that might contain outliers. Now we will define the following variables:

```
X25 = 25th percentile
X75 = 75th percentile
XM = mean of {Xi} values
XR = robust standardization
```

Then `XR` is computed according to the following formula:

```
XR = (Xi - XM)/(X75 - X25)
```

The preceding technique is also called IQR, which is an acronym for *interquartile range*, and you can see a sample calculation here:

https://en.wikipedia.org/wiki/Interquartile_range

What to Look for in Categorical Data

This section contains various suggestions for handling inconsistent data values, and you can determine which ones to adopt based on any additional factors that are relevant to your particular task. For example, consider dropping columns that have very low cardinality (equal to or close to 1), as well as numeric columns with zero or very low variance.

Next, check the contents of categorical columns for inconsistent spellings or errors. A good example pertains to the gender category, which can consist of a combination of the following values:

```
male
Male
female
Female
m
f
M
F
```

The preceding categorical values for gender can be replaced with two categorical values (unless you have a valid reason to retain some of the other values). Moreover, if you are training a model whose analysis involves a single gender, then you need to determine which rows (if any) of a dataset must be excluded. Also check categorical data columns for redundant or missing whitespaces.

Check for data values that have multiple data types, such as a numerical column with numbers as numerals and some numbers as strings or objects. Also ensure consistent data formats: numbers as integer or floating numbers and ensure that dates have the same format (for example, do not mix `mm/dd/yyyy` date formats with another date format, such as `dd/mm/yyyy`).

Mapping Categorical Data to Numeric Values

Character data is often called categorical data, examples of which include people's names, home or work addresses, email addresses, and so forth. Many types of categorical data involve short lists of values. For example, the days of the week and the months in a year involve seven and twelve distinct values, respectively. Notice that the days of the week have a relationship: each day has a previous day and a next day, and similarly for the months of a year.

However, the colors of an automobile are independent of each other: the color red is not "better" or "worse" than the color blue. Note that cars of a certain color can have a statistically higher number of accidents, but we will not address this case here.

There are several well-known techniques for mapping categorical values to a set of numeric values. A simple example where you need to perform this conversion involves the gender feature in the Titanic dataset. This feature is one of the relevant features for training a machine learning model. The gender feature has {M,F} as its set of values. As you will see later in this chapter, Pandas makes it very easy to convert the set of values {M,F} to the set of values {0, 1}.

Another mapping technique involves mapping a set of categorical values to a set of consecutive integer values. For example, the set {Red, Green, Blue} can be mapped to the set of integers {0, 1, 2}. The set {Male, Female} can be mapped to the set of integers {0, 1}. The days of the week can be mapped to {0, 1, 2, 3, 4, 5, 6}. Note that the first day of the week depends on the country: in some cases it is Sunday and in other cases it is Monday.

Another technique is called *one-hot encoding*, which converts each value to a *vector* (check Wikipedia if you need a refresher regarding vectors). Thus, {Male, Female} can be represented by the vectors `[1,0]` and `[0,1]`, and the colors {Red, Green, Blue} can be represented by the vectors `[1,0,0]`, `[0,1,0]`, and `[0,0,1]`. If you vertically "line up" the two vectors for gender, they form a 2×2 identity matrix, and doing the same for the colors will form a 3×3 identity matrix.

If you vertically "line up" the two vectors for gender, they form a 2×2 identity matrix, and doing the same for the colors {R,G,B} will form a 3x3 identity matrix, as shown here:

```
[1,0,0]
[0,1,0]
[0,0,1]
```

If you are familiar with matrices, you probably noticed that the preceding set of vectors looks like the 3×3 identity matrix. In fact, this technique generalizes in a straightforward manner. Specifically, if you have n distinct categorical values, you can map each of those values to one of the vectors in an nxn identity matrix.

As another example, the set of titles `{"Intern," "Junior," "Mid-Range," "Senior," "Project Leader," "Dev Manager"}` have a hierarchical relationship in terms of their salaries (which can also overlap, but we'll gloss over that detail for now).

Another set of categorical data involves the season of the year: `{"Spring," "Summer," "Autumn," "Winter"}` and while these values are generally independent of each other, there are cases in which the season is significant. For example, the values for the monthly rainfall, average temperature, crime rate, or foreclosure rate can depend on the season, month, week, or even the day of the year.

If a feature has a large number of categorical values, then a one-hot encoding will produce many additional columns for each data point. Since the majority of the values in the new columns equal 0, this can increase the sparsity of the dataset, which in turn can result in more overfitting and therefore adversely affect the accuracy of machine learning algorithms that you adopt during the training process.

Another solution is to use a sequence-based solution in which N categories are mapped to the integers 1, 2, . . . , N. Another solution involves examining the row frequency of each categorical value. For example, suppose that N equals 20, and there are 3 categorical values occur in 95% of the values for a given feature. You can try the following:

1. Assign the values 1, 2, and 3 to those three categorical values.
2. Assign numeric values that reflect the relative frequency of those categorical values.
3. Assign the category "OTHER" to the remaining categorical values.
4. Delete the rows that whose categorical values belong to the 5%.

Working With Dates

The format for a calendar dates varies among different countries, and this belongs to something called *localization* of data (not to be confused with `i18n`, which is data internationalization). Some examples of date formats are shown here (and the first four are probably the most common):

```
MM/DD/YY
MM/DD/YYYY
DD/MM/YY
DD/MM/YYYY
YY/MM/DD
M/D/YY
D/M/YY
YY/M/D
MMDDYY
DDMMYY
YYMMDD
```

If you need to combine data from datasets that contain different date formats, then converting the disparate date formats to a single common date format will ensure consistency.

Working With Currency

The format for currency depends on the country, which includes different interpretations for a "," and "." in currency (and decimal values in general). For example, `1,124.78` equals "one thousand one hundred twenty four point seven eight" in the United States, whereas `1.124,78` has the same meaning in Europe (i.e., the "." symbol and the "," symbol are interchanged).

If you need to combine data from datasets that contain different currency formats, then you probably need to convert all the disparate currency formats to a single common currency format. There is another detail to consider: currency exchange rates can fluctuate on a daily basis, which in turn can affect the calculation of taxes, late fees, and so forth. Although you might be fortunate enough where you will not have to deal with these issues, it is still worth being aware of them.

WORKING WITH OUTLIERS AND ANOMALIES

In simplified terms, an *outlier* is an abnormal data value that is outside the range of "normal" values. For example, a person's height in centimeters is typically between 30 centimeters and 250 centimeters. Therefore, a data point (e.g., a row of data in a spreadsheet) with a height of 5 centimeters or a height of 500 centimeters is an outlier. The consequences of these outlier values are unlikely to involve a significant financial or physical loss (although they could have an adverse effect on the accuracy of a trained model).

Anomalies are also outside the "normal" range of values (just like outliers), and they are typically more problematic than outliers: anomalies can have more "severe" consequences than outliers. For example, consider the scenario in which someone who lives in California suddenly makes a credit card purchase in New York. If the person is on vacation (or a business trip), then the purchase is an outlier (it is "outside" the typical purchasing pattern), but it is not an issue. However, if that person was in California when the credit card purchase was made, then it is most likely to be credit card fraud, as well as an anomaly.

Unfortunately, there is no simple way to decide how to deal with anomalies and outliers in a dataset. Although you can drop rows that contain outliers, keep in mind that doing so might deprive the dataset—and therefore the trained model—of valuable information. You can try modifying the data values (as will be discussed), but again, this might lead to erroneous inferences in the trained model. Another possibility is to train a model with the dataset that contains anomalies and outliers, and then train a model with a dataset from which the anomalies and outliers have been removed. Compare the two results and see if you can infer anything meaningful regarding the anomalies and outliers.

Outlier Detection/Removal

Although the decision to keep or drop outliers is your decision to make, there are some techniques available that help you detect outliers in a dataset. Here is a short list of techniques, along with a very brief description and links for additional information:

- trimming
- winsorizing
- minimum covariance determinant
- local outlier factor
- Huber and ridge
- isolation forest (tree-based algorithm)
- one-class SVM

Perhaps *trimming* is the simplest technique (apart from dropping outliers), which involves removing rows whose feature value is in the upper 5% range or the lower 5% range. *Winsorizing* the data is an improvement over trimming: set the values in the top 5% range equal to the maximum value in the 95th percentile, and set the values in the bottom 5% range equal to the minimum in the 5th percentile.

The *Minimum Covariance Determinant* is a covariance-based technique, and a `Python`-based code sample that uses this technique is downloadable here:

https://scikit-learn.org/stable/modules/outlier_detection.html

The *Local Outlier Factor* (LOF) technique is an unsupervised technique that calculates a local anomaly score via the kNN (k Nearest Neighbor) algorithm. Documentation and short code samples that use LOF are here:

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html

Two other techniques involve the *Huber* and the *ridge* classes, both of which are included as part of `Sklearn`. Huber error is less sensitive to outliers because it is calculated via linear loss, similar to MAE (mean absolute error). A code sample that compares Huber and ridge is downloadable here:

https://scikit-learn.org/stable/auto_examples/linear_model/plot_huber_vs_ridge.html

You can also explore the Theil-Sen estimator and RANSAC that are "robust" against outliers, and additional information is here:

https://scikit-learn.org/stable/auto_examples/linear_model/plot_theilsen.html

https://en.wikipedia.org/wiki/Random_sample_consensus

Four algorithms for outlier detection are discussed here:

https://www.kdnuggets.com/2018/12/four-techniques-outlier-detection.html

One other scenario involves "local" outliers. For example, suppose that you use `kMeans` (or some other clustering algorithm) and determine that a value that is an outlier with respect to one of the clusters. While this value is not necessarily an "absolute" outlier, detecting such a value might be important for your use case.

FINDING OUTLIERS WITH NUMPY

Although we have not discussed the `Numpy` library, we will use the `Numpy` `array()` method, the `mean()` method, and the `std()` method in this section, all of which have intuitive functionality.

Listing 2.1 displays the contents of `numpy_outliers1.py` that illustrates how to use `Numpy` methods to find outliers in an array of numbers.

LISTING 2.1: numpy_outliers1.py

```
import numpy as np

arr1 = np.array([2,5,7,9,9,40])
print("values:",arr1)

data_mean = np.mean(arr1)
data_std  = np.std(arr1)
print("data_mean:",data_mean)
print("data_std:" ,data_std)
print()
```

```
multiplier = 1.5
cut_off = data_std * multiplier
lower = data_mean - cut_off
upper = data_mean + cut_off
print("lower cutoff:",lower)
print("upper cutoff:",upper)
print()

outliers = [x for x in arr1 if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
print("outliers:",outliers)
```

Listing 2.1 starts by defining a `Numpy` array of numbers and then calculates the mean and standard deviation of those numbers. The next block of code initializes two numbers that represent the upper and lower values that are based on the value of the `cut_off` variable. Any numbers in the array `arr1` that lie to the left of the lower value or to the right of the upper value are treated as outliers.

The final section of code in Listing 2.1 initializes the variable `outliers` with the numbers that are determined to be outliers, and those values are printed. Now launch the code in Listing 2.1 and you will see the following output:

```
values: [ 2  5  7  9  9 40]
data_mean: 12.0
data_std: 12.754084313139327

lower cutoff: -7.131126469708988
upper cutoff: 31.13112646970899

Identified outliers: 1
outliers: [40]
```

The preceding code sample specifies a hard-coded value in order to calculate the upper and lower range values.

Listing 2.2 is an improvement in that you can specify a set of values from which to calculate the upper and lower range values, and the new block of code is shown in bold.

LISTING 2.2: numpy_outliers2.py

```
import numpy as np

arr1 = np.array([2,5,7,9,9,40])
print("values:",arr1)

data_mean = np.mean(arr1)
data_std  = np.std(arr1)
print("data_mean:",data_mean)
print("data_std:" ,data_std)
print()

multipliers = np.array([0.5,1.0,1.5,2.0,2.5,3.0])
for multiplier in multipliers:
  cut_off = data_std * multiplier
  lower, upper = data_mean - cut_off, data_mean + cut_off
  print("=> multiplier:  ",multiplier)
  print("lower cutoff:",lower)
  print("upper cutoff:",upper)

  outliers = [x for x in df['data'] if x < lower or x > upper]
  print('Identified outliers: %d' % len(outliers))
  print("outliers:",outliers)
  print()
```

Listing 2.2 contains a block of new code that initializes the variable `multipliers` as an array of numeric values that are used for finding outliers. Although you will probably use a value of 2.0 or larger on a real dataset, this range of numbers can give you a better sense of detecting outliers. Now launch the code in Listing 2.2 and you will see the following output:

```
values: [ 2  5  7  9  9 40]
data_mean: 12.0
data_std: 12.754084313139327

lower cutoff: -7.131126469708988
upper cutoff: 31.13112646970899

Identified outliers: 1
outliers: [40]
=> multiplier:   0.5
lower cutoff: 5.622957843430337
upper cutoff: 18.377042156569665
Identified outliers: 3
outliers: [2, 5, 40]

=> multiplier:   1.0
lower cutoff: -0.7540843131393267
upper cutoff: 24.754084313139327
Identified outliers: 1
outliers: [40]

=> multiplier:   1.5
lower cutoff: -7.131126469708988
upper cutoff: 31.13112646970899
Identified outliers: 1
outliers: [40]

=> multiplier:   2.0
lower cutoff: -13.508168626278653
upper cutoff: 37.50816862627865
Identified outliers: 1
outliers: [40]

=> multiplier:   2.5
lower cutoff: -19.88521078284832
upper cutoff: 43.88521078284832
Identified outliers: 0
outliers: []
```

```
=> multiplier:   3.0
lower cutoff: -26.262252939417976
upper cutoff: 50.26225293941798
Identified outliers: 0
outliers: []
```

FINDING OUTLIERS WITH PANDAS

Although we discuss `Pandas` in Chapter 3, the code in this section only involves a very simple `Pandas` data frame, the `mean()` method, and the `std()` method.

Listing 2.3 displays the contents of `pandas_outliers1.py` that illustrates how to use `Pandas` to find outliers in an array of numbers.

LISTING 2.3: pandas_outliers1.py

```
import pandas as pd

df = pd.DataFrame([2,5,7,9,9,40])
df.columns = ["data"]

data_mean = df['data'].mean()
data_std  = df['data'].std()
print("data_mean:",data_mean)
print("data_std:" ,data_std)
print()

multiplier = 1.5
cut_off = data_std * multiplier
lower, upper = data_mean - cut_off, data_mean + cut_off
print("lower cutoff:",lower)
print("upper cutoff:",upper)
print()

outliers = [x for x in df['data'] if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
print("outliers:",outliers)
```

Listing 2.3 starts by defining a `Pandas` data frame and then calculates the mean and standard deviation of those numbers. The next block of code initializes two numbers that represent the upper and lower values that are based on the value of the `cut_off` variable. Any numbers in the data frame that lie to the left of the lower value or to the right of the upper value are treated as outliers.

The final section of code in Listing 2.3 initializes the variable `outliers` with the numbers that are determined to be outliers, and those values are printed. Now launch the code in Listing 2.3 and you will see the following output:

```
values: [ 2  5  7  9  9 40]
data_mean: 12.0
data_std: 12.754084313139327
```

```
lower cutoff: -7.131126469708988
upper cutoff: 31.13112646970899

Identified outliers: 1
outliers: [40]
```

The preceding code sample specifies a hard-coded value in order to calculate the upper and lower range values.

Listing 2.4 is an improvement in that you can specify a set of values from which to calculate the upper and lower range values, and the new block of code is shown in bold.

LISTING 2.4: pandas_outliers2.py

```
import pandas as pd

#df = pd.DataFrame([2,5,7,9,9,40])
#df = pd.DataFrame([2,5,7,8,42,44])
df = pd.DataFrame([2,5,7,8,42,492])
df.columns = ["data"]
print("=> data values:")
print(df['data'])

data_mean = df['data'].mean()
data_std  = df['data'].std()
print("=> data_mean:",data_mean)
print("=> data_std:" ,data_std)
print()

multipliers = [0.5,1.0,1.5,2.0,2.5,3.0]
for multiplier in multipliers:
  cut_off = data_std * multiplier
  lower, upper = data_mean - cut_off, data_mean + cut_off
  print("=> multiplier:  ",multiplier)
  print("lower cutoff:",lower)
  print("upper cutoff:",upper)

  outliers = [x for x in df['data'] if x < lower or x > upper]
  print('Identified outliers: %d' % len(outliers))
  print("outliers:",outliers)
  print()
```

Listing 2.4 contains a block of new code that initializes the variable `multipliers` as an array of numeric values that are used for finding outliers. Although you will probably use a value of 2.0 or larger on a real dataset, this range of numbers can give you a better sense of detecting outliers. Now launch the code in Listing 2.4 and you will see the following output:

```
=> data values:
0      2
1      5
2      7
3      8
```

```
4     42
5    492
Name: data, dtype: int64
=> data_mean: 92.66666666666667
=> data_std: 196.187325448579

=> multiplier:   0.5
lower cutoff: -5.42699605762283
upper cutoff: 190.76032939095617
Identified outliers: 1
outliers: [492]

=> multiplier:   1.0
lower cutoff: -103.52065878191233
upper cutoff: 288.85399211524566
Identified outliers: 1
outliers: [492]

=> multiplier:   1.5
lower cutoff: -201.6143215062018
upper cutoff: 386.9476548395352
Identified outliers: 1
outliers: [492]

=> multiplier:   2.0
lower cutoff: -299.7079842304913
upper cutoff: 485.0413175638247
Identified outliers: 1
outliers: [492]

=> multiplier:   2.5
lower cutoff: -397.80164695478084
upper cutoff: 583.1349802881142
Identified outliers: 0
outliers: []

=> multiplier:   3.0
lower cutoff: -495.8953096790703
upper cutoff: 681.2286430124036
Identified outliers: 0
outliers: []
```

Calculating Z-Scores to Find Outliers

The `z-score` of a set of numbers is calculated by standardizing those numbers, which involves (a) subtracting their mean from each number, and (b) dividing by their standard deviation. Although you can perform these steps manually, the `Python SciPy` library performs these calculations. If need be, you can install this package with the following command:

```
pip3 install scipy
```

Listing 2.5 displays the contents of `outliers_zscores.py` that illustrates how to find outliers in an array of numbers. As you will see, this code sample relies on convenience methods from `Numpy`, `Pandas`, and `SciPy`.

LISTING 2.5: outliers_zscores.py

```
import numpy as np
import pandas as pd
from scipy import stats

arr1 = np.array([2,5,7,9,9,40])
print("values:",arr1)

df = pd.DataFrame(arr1)

zscores = np.abs(stats.zscore(df))
print("z scores:")
print(zscores)
print()

upper = 2.0
lower = 0.5
print("=> upper outliers:")
print(zscores[np.where(zscores > upper)])
print()

print("=> lower outliers:")
print(zscores[np.where(zscores < lower)])
print()
```

Listing 2.5 starts with several `import` statements, followed by initializing the variable `arr1` as a `Numpy` array of numbers, and then displaying the values in `arr1`. The next code snippet initializes the variable `df` as a data frame that contains the values in the variable `arr1`.

Next, the variable `zscores` is initialized with the z-scores of the elements of the df data frame, as shown here:

```
zscores = np.abs(stats.zscore(df))
```

The next section of code initializes the variables `upper` and `lower`, and the `z-scores` whose values are less than the value of `lower` or greater than the value `upper` are treated as outliers, and those values are displayed. Now launch the code in Listing 2.5 and you will see the following output:

```
values: [2  5  7  9  9 40]
z scores:
[[0.78406256]
 [0.54884379]
 [0.39203128]
 [0.23521877]
 [0.23521877]
 [2.19537517]]

=> upper outliers:
[2.19537517]

=> lower outliers:
[0.39203128 0.23521877 0.23521877]
```

FINDING OUTLIERS WITH SKLEARN (OPTIONAL)

This section is optional because the code involves the `EllipticEnvelope` class in `sklearn.covariance`, which we do not cover in this book. However, it is still worthwhile to peruse the code and compare this code with earlier code samples for finding outliers.

Listing 2.6 displays the contents of `elliptic_envelope_outliers.py` that illustrates how to use `Pandas` to find outliers in an array of numbers.

LISTING 2.6: elliptic_envelope_outliers.py

```
# pip3 install sklearn
from sklearn.covariance import EllipticEnvelope
import numpy as np

# Create a sample normal distribution:
Xdata = np.random.normal(loc=5, scale=2, size=10).reshape(-1, 1)
print("Xdata values:")
print(Xdata)
print()

# instantiate and fit the estimator:
envelope = EllipticEnvelope(random_state=0)
envelope.fit(Xdata)

# create a test set:
test = np.array([0, 2, 4, 6, 8, 10, 15, 20, 25, 30]).reshape(-1, 1)
print("test values:")
print(test)
print()

# predict() returns 1 for inliers and -1 for outliers:
print("envelope.predict(test):")
print(envelope.predict(test))
```

Listing 2.6 starts with several `import` statements and then initializes the variable `Xdata` as a column vector of random numbers from a Gaussian distribution. The next code snippet initializes the variable envelope as an instance of the `EllipticEnvelope` from `Sklearn` (which will determine if there are any outliers in `Xdata`), and then trained on the data values in `Xdata`.

The next portion of Listing 2.6 initializes the variable `test` as a column vector, much like the initialization of `Xdata`. The final portion of Listing 2.6 makes a prediction on the values in the variable test and also displays the results: the value –1 indicates an outlier. Now launch the code in Listing 2.6 and you will see the following output:

```
Xdata values:
[[5.21730452]
 [5.49182377]
 [2.87553776]
 [4.20297013]
 [8.29562026]
 [5.78097977]
 [4.86631006]
 [5.47184212]
 [4.77954946]
 [8.66184028]]

test values:
[[ 0]
 [ 2]
 [ 4]
 [ 6]
 [ 8]
 [10]
 [15]
 [20]
 [25]
 [30]]

envelope.predict(test):
[-1  1  1  1  1 -1 -1 -1 -1 -1]
```

WORKING WITH MISSING DATA

There are various reasons for missing values in a dataset, some of which are as follows:

- Values are unavailable.
- Values were improperly collected.
- Inaccurate data was entered.

Although you might be tempted to *always* replace a missing values with a concrete value, there are situations in which you cannot determine a value. As a simple example, a survey that contains questions for people under 30 will have a missing value for respondents who are over 30, and in this case specifying a value for the missing value would be incorrect. With these details in mind, there are various ways to fill missing values, some of which are listed here:

- Remove the lines with the data if the dataset is large enough *and* there is a high percentage of missing values (50% or larger).
- Fill null variables with 0 for numeric features.
- Use the Imputerclass from the scikit-learn library.
- Fill missing values with the value in an adjacent row.
- Replace missing data with the mean/median/mode value.
- Infer ("impute") the value for missing data.
- Delete rows with missing data.

Once again, the technique that you select for filling missing values is influenced by various factors, such as:

- how you want to process the data
- the type of data involved
- the cause of missing values. (See list at the beginning of this section.)

Although the most common technique involves the mean value for numeric features, someone needs to determine which technique is appropriate for a given feature.

However, if you are not confident that you can impute a reasonable value, consider either deleting the row with a missing value, or training a model with the imputed value and also with the deleted row.

One problem that can arise after removing rows with missing values is that the resulting data set is too small. In this case, consider using `SMOTE` (synthetic minority oversampling technique), which is discussed later in this chapter in order to generate synthetic data.

https://www.kdnuggets.com/2020/09/missing-value-imputation-review.html

Imputing Values: When Is Zero a Valid Value?

In general, replace a missing numeric value with zero is a risky choice: this value is obviously incorrect if the values of a feature are positive numbers between 1,000 and 5,000 (or some other range of positive numbers). For a feature that has numeric values, replacing a missing value with the mean of existing values can be better than the value zero (unless the average equals zero); also consider using the median value. For categorical data, consider using the mode to replace a missing value.

There are situations where you can use the mean of existing values to impute missing values but not the value zero, and vice versa. As a first example, suppose that an attribute contains the height in centimeters of a set of persons. In this case, the mean could be a reasonable imputation, whereas 0 suffers from:

1. It is an invalid value (nobody has height 0).
2. It will skew statistical quantities such as the mean and variance.

You might be tempted to use the mean instead of 0 when the minimum allowable value is a positive number, but use caution when working with highly imbalanced data sets. As a second example, consider a small community of 50 residents in which:

1. Forty-five people have an average annual income of US$ 50,000.
2. Four other residents have an annual income of US$ 10,000,000.
3. One resident has an unknown annual income.

Although the preceding example might seem contrived, it is likely that the median income is preferable to the mean income, and certainly better than imputing a value of 0.

As a third example, suppose that a company generates weekly sales reports for multiple office branches, and a new office has been opened, but has yet to make any sales. In this case, the use of the mean to impute missing values for this branch would produce fictitious results. Therefore, it makes sense to use the value 0 for all sales-related quantities, which will accurately reflect the sales-related status of the new branch.

DEALING WITH IMBALANCED DATASETS

Imbalanced datasets contain at least once class that has many more values than another class in the dataset. For example, if class A has 99% of the data and class B has 1%, which classification algorithm would you use? Unfortunately, classification algorithms do not work so well with this type of imbalanced dataset.

Imbalanced classification involves dealing with imbalanced datasets, and the following list contains several well-known techniques:

- Random resampling rebalances the class distribution.
- Random undersampling deletes examples from the majority class.
- Random oversampling duplicates data in the minority class.
- SMOTE (synthetic minority oversampling technique) to calculate missing values.

Random resampling rebalances the class distribution by resampling the data space.

Alternatively, the random undersampling technique removes samples that belong to the majority class from the dataset, and involves the following:

- randomly remove samples from majority class
- can be performed with or without replacement
- alleviates imbalance in the dataset
- may increase the variance of the classifier
- may discard useful or important samples

However, random undersampling does not work so well with extremely unbalanced datasets, such as a 99% and 1% split into two classes. Moreover, undersampling can result in losing information that is useful for a model.

Random oversampling generates new samples from a minority class: this technique duplicates examples from the minority class.

Another option to consider is the `Python` package `imbalanced-learn` in the `scikit-learn-contrib` project. This project provides various re-sampling techniques for datasets that exhibit class imbalance. More details are available here:

https://github.com/scikit-learn-contrib/imbalanced-learn

Another well-known technique is called `SMOTE`, which involves data augmentation (i.e., synthesizing new data samples). `SMOTE` was initially developed by means of the `kNN` algorithm (other options are available), and it can be an effective technique for handling imbalanced classes. `SMOTE` is discussed in more detail in the next section.

WHAT IS SMOTE?

`SMOTE` (synthetic minority oversampling technique) is a technique for synthesizing new samples for a dataset. This technique is based on linear interpolation:

Step 1: Select two samples that are close in the feature space.
Step 2: Draw a line between the two samples in the feature space.
Step 3: Draw a new sample at a point on the line in Step 2.

A more detailed explanation of the SMOTE algorithm is as follows:

- Step 1: Select a random sample “a” from the minority class.
- Step 2: Find k nearest neighbors for that example.
- Step 3: Select a random neighbor “b” from the nearest neighbors.
- Step 4: Create a line L that connects “a” and “b.”
- Step 5: Randomly select one or more points “c” on line L.

If need be, you can repeat this process for the other (k–1) nearest neighbors in order to distribute the synthetic values more evenly among the nearest neighbors.

SMOTE Extensions

The initial `SMOTE` algorithm is based on the `kNN` classification algorithm, which has been extended in various ways, such as replacing `kNN` with `SVM`. A list of `SMOTE` extensions is as follows:

- selective synthetic sample generation
- borderline-SMOTE (kNN)
- borderline-SMOTE (SVM)
- adaptive synthetic sampling (ADASYN)

Perform an Internet search for more details about these algorithms, and also navigate to the following URL:

https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis

THE BIAS-VARIANCE TRADEOFF

This section is presented from the viewpoint of machine learning, but the concepts of bias and variance are highly relevant outside of machine learning, so it is probably still worthwhile to read this section as well as the previous section.

Bias in machine learning can be due to an error from wrong assumptions in a learning algorithm. High bias might cause an algorithm to miss relevant relations between features and target outputs (underfitting). Prediction bias can occur because of "noisy" data, an incomplete feature set, or a biased training sample.

Error due to bias is the difference between the expected (or average) prediction of your model and the correct value that you want to predict. Repeat the model building process multiple times, and gather new data each time, and also perform an analysis to produce a new model. The resulting models have a range of predictions because the underlying datasets have a degree of randomness. Bias measures the extent to the predictions for these models are from the correct value.

Variance in machine learning is the expected value of the squared deviation from the mean. High variance can/might cause an algorithm to model the random noise in the training data, rather than the intended outputs (also referred to as "overfitting"). Moreover, adding parameters to a model increases its complexity, increases the variance, and decreases the bias.

The point to remember: dealing with bias and variance involves dealing with underfitting and overfitting.

Error due to variance is the variability of a model prediction for a given data point. As before, repeat the entire model building process, and the variance is the extent to which predictions for a given point vary among different "instances" of the model.

If you have worked with datasets and performed data analysis, you already know that finding well-balanced samples can be difficult or highly impractical. Moreover, performing an analysis of the data in a dataset is vitally important, yet there is no guarantee that you can produce a dataset that is 100% "clean."

A *biased statistic* is a statistic that is systematically different from the entity in the population that is being estimated. In more casual terminology, if a data sample "favors" or "leans" toward one aspect of the population, then the sample has bias. For example, if you prefer movies that are comedies more so than any other type of movie, then clearly you are more likely to select a comedy instead of a dramatic movie or a science fiction movie. Thus, a frequency graph of the movie types in a sample of your movie selections will be more closely clustered around comedies.

However, if you have a wide-ranging set of preferences for movies, then the corresponding frequency graph will be more varied, and therefore have a larger spread of values.

As another example, suppose that you have an assignment that involves writing a term paper on a controversial subject that has many opposing viewpoints. Since you want a bibliography that supports your well-balanced term paper that takes into account multiple viewpoints, your bibliography will contain a wide variety of sources.

In other words, your bibliography will have a larger variance and a smaller bias. Contrastingly if most (or all) the references in your bibliography espouses the same point of view, then you will have a smaller variance and a larger bias. (It is an analogy, so it is not a perfect counterpart to bias-versus-variance).

The bias-variance trade-off can be stated in simple terms: in general, reducing the bias in samples can increase the variance, whereas reducing the variance tends to increase the bias.

Types of Bias in Data

In addition to the bias-variance trade-off that is discussed in the previous section, there are several types of bias, some of which are listed here:

- availability bias
- confirmation bias
- false causality
- sunk cost fallacy
- survivorship bias

Availability bias is akin to making a "rule" based on an exception. For example, there is a known link between smoking cigarettes and cancer, but there are exceptions. If you find someone who has smoked three packs of cigarettes on a daily basis for four decades and is still healthy, can you assert that smoking does not lead to cancer?

Confirmation bias refers to the tendency to focus on data that confirms their beliefs and simultaneously ignore data that contradicts a belief.

False causality occurs when you incorrectly assert that the occurrence of a particular event causes another event to occur as well. One of the most well-known examples involves ice cream consumption and violent crime in New York during the summer. Since more people eat ice cream in the summer, that seems to "cause" more violent crime, which is a false causality. Other factors, such as the increase in temperature, may be linked to the increase in crime. However, it is important to distinguish between correlation and causality: the latter is a much stronger link than the former, and it is also more difficult to establish causality instead of correlation.

Sunk cost refers to something (often money) that has been spent or incurred that cannot be recouped. A common example pertains to gambling at a casino: people fall into the pattern of spending more money in order to recoup a substantial amount of money that has already been lost. While there are cases in which people do recover their money, in many (most?) cases people simply incur an even greater loss because they continue to spend their money. Perhaps you have heard of "it is time to cut your losses and walk away."

Survivorship bias refers to analyzing a particular subset of "positive" data while ignoring the "negative" data. This bias occurs in various situations, such as being influenced by individuals who recount their rags-to-riches success story ("positive" data) while ignoring the fate of the people (which is often a very high percentage) who did not succeed (the "negative" data) in a similar quest. So, while it is certainly possible for an individual to overcome many difficult obstacles in order to succeed, is the success rate one in one thousand (or even lower)?

ANALYZING CLASSIFIERS (OPTIONAL)

This section is marked optional because its contents pertain to machine learning classifiers, which is not the focus of this book. However, it is still worthwhile to glance through the material, or perhaps return to this section after you have a basic understanding of machine learning classifiers.

Several well-known techniques are available for analyzing the quality of machine learning classifiers. Two techniques are `LIME` and `ANOVA`, both of which are discussed in the following subsections.

What Is LIME?

`LIME` is an acronym for local interpretable model-agnostic explanations. `LIME` is a model-agnostic technique that can be used with machine learning models. The intuition for this technique is straightforward: make small random changes to data samples and then observe the manner in which predictions change (or not). The intuition involves changing the output (slightly) and then observe what happens to the output.

By way of analogy, consider food inspectors who test for bacteria in truckloads of perishable food. Clearly it is infeasible to test every food item in a truck (or a train car), so inspectors perform "spot checks" that involve testing randomly selected items. In an analogous fashion, `LIME` makes small changes to input data in random locations and then analyzes the changes in the associated output values.

However, there are two caveats to keep in mind when you use `LIME` with input data for a given model:

1. The actual changes to input values are model-specific.
2. This technique works on input that is interpretable.

Examples of interpretable input include machine learning classifiers (such as trees and random forests) and `NLP` techniques such as `BoW`. Noninterpretable input involves "dense" data, such as a word embedding (which is a vector of floating point numbers).

You could also substitute your model with another model that involves interpretable data, but then you need to evaluate how accurate the approximation is to the original model.

What Is ANOVA?

ANOVA is an acronym for *analysis of variance*, which attempts to analyze the differences among the mean values of a sample that's taken from a population. ANOVA enables you to test if multiple mean values are equal. More importantly, ANOVA can assist in reducing Type I (false positive) errors and Type II errors (false negative) errors. For example, suppose that person A is diagnosed with cancer and person B is diagnosed as healthy, and that both diagnoses are incorrect. Then the result for person A is a false positive whereas the result for person B is a false negative. In general, a test result of false positive is *much* preferable to a test result of false negative.

ANOVA pertains to the design of experiments and hypothesis testing, which can produce meaningful results in various situations. For example, suppose that a dataset contains a feature that can be partitioned into several "reasonably" homogenous groups. Next, analyze the variance in each group and perform comparisons with the goal of determining different sources of variance for the values of a given feature. For more information about ANOVA, navigate to the following link:

https://en.wikipedia.org/wiki/Analysis_of_variance

SUMMARY

This chapter started with an explanation of datasets, a description of data wrangling, and details regarding various types of data. Then you learned about techniques for scaling numeric data, such as normalization and standardization. You saw how to convert categorical data to numeric values and how to handle dates and currency.

Then you learned how to work with outliers, anomalies, and missing data, along with various techniques for handling these scenarios. You also learned about imbalanced data and evaluating the use of SMOTE to deal with imbalanced classes in a dataset. In addition, you learned about the bias-variance tradeoff and various types of statistical bias. Finally, you learned about classifiers using the techniques LIME and ANOVA.

CHAPTER 3

INTRODUCTION TO PANDAS

This chapter introduces the `Pandas` library and contains various code samples that illustrate some useful `Pandas` features. As you will see, the title of each section clearly indicates its contents, so you can easily scan this chapter for those sections that contain material that is new to you. This approach will help you make efficient use of your time when you read the contents of this chapter.

The first part of this chapter contains a brief introduction to `Pandas`, followed by code samples that illustrate how to define `Pandas DataFrames` and also display their attributes. Please keep in mind that this chapter is devoted to `Pandas DataFrames`. There is one code block that illustrates how to define a `Pandas Series`, and if you want to learn more about this `Pandas Series`, you can search online for more information.

The second part of this chapter discusses various types of data frames that you can create, such as numeric and Boolean `Data frames`. In addition, you will see examples of creating `Data frames` with `NumPy` functions and random numbers. You will also see examples of converting between `Python` dictionaries and `JSON`-based data, and also how to create a `Pandas DataFrame` from `JSON`-based data.

WHAT IS PANDAS?

`Pandas` is a `Python` package that is compatible with other `Python` packages, such as `NumPy`, `Matplotlib`, and so forth. Install `Pandas` by opening a command shell and invoking this command for `Python` 3.x:

```
pip3 install pandas
```

In many ways the `Pandas` package has the semantics of a spreadsheet, and it also works with various file types, such as `xsl, xml, html, csv` files.

`Pandas` provides a data type called a `Data frame` (similar to a `Python` dictionary) with extremely powerful functionality (similar to the functionality of a spreadsheet).

Pandas DataFrames

In simplified terms, a `Pandas DataFrame` is a two-dimensional data structure, and it is convenient to think of the data structure in terms of rows and columns. `Data frames` can be labeled (rows as well as columns), and the columns can contain different data types. The source of the dataset can be a data file, database tables, web service, and so forth. `Pandas DataFrame` features include:

- data frame methods
- data frame statistics
- grouping, pivoting, and reshaping
- handle missing data
- join data frames

Pandas Operations: In-place or Not?

In `Pandas` makes a copy of the underlying data frame before performing an operation on that data frame. Although you can specify `inplace=True`, Pandas might still create a copy of the data frame. Such behavior can have an impact in code that performs method chaining, which involves executing a "chain" of functions that are connected via a ".", without the need to initialize intermediate variables.

More details regarding Pandas in-place operations can be found here:

https://stackoverflow.com/questions/69969482/why-arent-pandas-operations-in-place

Data Frames and Data Cleaning Tasks

The specific tasks that you need to perform depend on the structure and contents of a dataset. In general, you will perform a workflow with the following steps (not necessarily always in this order), all of which can be performed with a `Pandas DataFrame`:

- Read data into a data frame.
- Display top of data frame.
- Display column data types.
- Display nonmissing values.
- Replace NA with a value.
- Iterate through the columns.
- Compute statistics for each column
- Find missing values.
- Total missing values.
- Calculate percentage of missing values

- Sort table values.
- Print summary information.
- Find columns with > 50% missing values
- Rename columns.

A PANDAS DATAFRAME EXAMPLE

Listing 3.1 displays the contents of `pandas_df.py` that illustrates how to define several `Pandas DataFrames` and display their contents.

LISTING 3.1: pandas_df.py

```
import pandas as pd
import numpy as np

myvector1 = np.array([1,2,3,4,5])
print("myvector1:")
print(myvector1)
print()

mydf1 = pd.Data frame(myvector1)
print("mydf1:")
print(mydf1)
print()

myvector2 = np.array([i for i in range(1,6)])
print("myvector2:")
print(myvector2)
print()

mydf2 = pd.Data frame(myvector2)
print("mydf2:")
print(mydf2)
print()

myarray = np.array([[10,30,20], [50,40,60],[1000,2000,3000]])
print("myarray:")
print(myarray)
print()

mydf3 = pd.Data frame(myarray)
print("mydf3:")
print(mydf3)
print()
```

Listing 3.1 starts with a standard `import` statement for `Pandas` and `NumPy`, followed by the definition of two one-dimensional `NumPy` arrays and a two-dimensional `NumPy` array. The `NumPy` syntax ought to be familiar to you (many basic tutorials are available online). Each `NumPy` variable is followed by a corresponding `Pandas DataFrame mydf1`, `mydf2`, and `mydf3`. Now launch the code in Listing 3.1 and you will see the following output, and you can compare the NumPy arrays with the Pandas DataFrames:

```
myvector1:
[1 2 3 4 5]

mydf1:
   0
0  1
1  2
2  3
3  4
4  5

myvector2:
[1 2 3 4 5]

mydf2:
   0
0  1
1  2
2  3
3  4
4  5

myarray:
[[  10   30   20]
 [  50   40   60]
 [1000 2000 3000]]

mydf3:
      0     1     2
0    10    30    20
1    50    40    60
2  1000  2000  3000
```

By contrast, the following code block illustrates how to define a Pandas Series:

```
names = pd.Series(['SF', 'San Jose', 'Sacramento'])
sizes = pd.Series([852469, 1015785, 485199])
df = pd.Data frame({ 'Cities': names, 'Size': sizes })
print(df)
```

Create a Python file with the preceding code (along with the required import statement), and when you launch that code you will see the following output:

```
    City name    sizes
0          SF   852469
1    San Jose  1015785
2  Sacramento   485199
```

DESCRIBING A PANDAS DATA FRAME

Listing 3.2 displays the contents of `pandas_df_describe.py` that illustrates how to define a `Pandas DataFrame` that contains a 3×3 `NumPy` array of integer values, where the rows and columns of the data frame are labeled. Various other aspects of the data frame are also displayed.

LISTING 3.2: pandas_df_describe.py

```
import numpy as np
import pandas as pd

myarray = np.array([[10,30,20], [50,40,60],[1000,2000,3000]])

rownames = ['apples', 'oranges', 'beer']
colnames = ['January', 'February', 'March']

mydf = pd.Data frame(myarray, index=rownames, columns=colnames)
print("contents of df:")
print(mydf)
print()

print("contents of January:")
print(mydf['January'])
print()

print("Number of Rows:")
print(mydf.shape[0])
print()

print("Number of Columns:")
print(mydf.shape[1])
print()

print("Number of Rows and Columns:")
print(mydf.sh
ape)
print()

print("Column Names:")
print(mydf.columns)
print()

print("Column types:")
print(mydf.dtypes)
print()

print("Description:")
print(mydf.describe())
print()
```

Listing 3.2 starts with two standard `import` statements followed by the variable `myarray`, which is a 3×3 `NumPy` array of numbers. The variables `rownames` and `colnames` provide names for the rows and columns, respectively, of the `Pandas DataFrame mydf`, which is initialized as a `Pandas DataFrame` with the specified data source (i.e., `myarray`).

The first portion of the output below requires a single `print()` statement (which simply displays the contents of `mydf`). The second portion of the output is generated by invoking the `describe()` method that is available for any `NumPy Data frame`. The `describe()` method is very useful: you will see various statistical quantities, such as the mean, standard deviation minimum, and maximum performed column_wise (not row_wise), along with values for the 25th, 50th, and 75th percentiles. The output of Listing 3.2 is here:

```
contents of df:
         January  February  March
apples        10        30     20
oranges       50        40     60
beer        1000      2000   3000

contents of January:
apples       10
oranges      50
beer       1000
Name: January, dtype: int64

Number of Rows:
3

Number of Columns:
3

Number of Rows and Columns:
(3, 3)

Column Names:
Index(['January', 'February', 'March'], dtype='object')

Column types:
January     int64
February    int64
March       int64
dtype: object

Description:
           January     February        March
count     3.000000     3.000000     3.000000
mean    353.333333   690.000000  1026.666667
std     560.386771  1134.504297  1709.073823
min      10.000000    30.000000    20.000000
25%      30.000000    35.000000    40.000000
50%      50.000000    40.000000    60.000000
75%     525.000000  1020.000000  1530.000000
max    1000.000000  2000.000000  3000.000000
```

PANDAS BOOLEAN DATA FRAMES

Pandas supports Boolean operations on Data frames, such as the logical or, the logical and, and the logical negation of a pair of Data frames. Listing 3.3 displays the contents of pandas_boolean_df.py that illustrates how to define a Pandas DataFrame in which rows and columns are Boolean values.

LISTING 3.3: pandas_boolean_df.py

```
import pandas as pd

df1 = pd.Data frame({'a': [1, 0, 1], 'b': [0, 1, 1] }, dtype=bool)
df2 = pd.Data frame({'a': [0, 1, 1], 'b': [1, 1, 0] }, dtype=bool)

print("df1 & df2:")
print(df1 & df2)

print("df1 | df2:")
print(df1 | df2)

print("df1 ^ df2:")
print(df1 ^ df2)
```

Listing 3.3 initializes the Data frames df1 and df2, and then computes df1 & df2, df1 | df2, df1 ^ df2, which represent the logical AND, the logical OR, and the logical negation, respectively, of df1 and df2. The output from launching the code in Listing 3.3 is here:

```
df1 & df2:
       a      b
0  False  False
1  False   True
2   True  False
df1 | df2:
      a     b
0  True  True
1  True  True
2  True  True
df1 ^ df2:
       a      b
0   True   True
1   True  False
2  False   True
```

Transposing a Pandas Data Frame

The T attribute (as well as the transpose function) enables you to generate the transpose of a Pandas DataFrame, similar to a NumPy ndarray.

For example, the following code snippet defines a Pandas DataFrame df1 and then displays the transpose of df1:

```
df1 = pd.Data frame({'a': [1, 0, 1], 'b': [0, 1, 1] }, dtype=int)

print("df1.T:")
print(df1.T)
```

The output is here:

```
df1.T:
   0  1  2
a  1  0  1
b  0  1  1
```

The following code snippet defines `Pandas DataFrames df1 and df2` and then displays their sum:

```
df1 = pd.Data frame({'a' : [1, 0, 1], 'b' : [0, 1, 1] }, dtype=int)
df2 = pd.Data frame({'a' : [3, 3, 3], 'b' : [5, 5, 5] }, dtype=int)

print("df1 + df2:")
print(df1 + df2)
```

The output is here:

```
df1 + df2:
   a  b
0  4  5
1  3  6
2  4  6
```

PANDAS DATA FRAMES AND RANDOM NUMBERS

Listing 3.4 displays the contents of `pandas_random_df.py` that illustrates how to create a `Pandas DataFrame` with random numbers.

LISTING 3.4: pandas_random_df.py

```
import pandas as pd
import numpy as np

df = pd.Data frame(np.random.randint(1, 5, size=(5, 2)),
columns=['a','b'])
df = df.append(df.agg(['sum', 'mean']))

print("Contents of data frame:")
print(df)
```

Listing 3.4 defines the `Pandas DataFrame df` that consists of 5 rows and 2 columns that contain random integers between 1 and 5. Notice that the columns of `df` are labeled “a” and “b.” In addition, the next code snippet appends two rows consisting of the sum and the mean of the numbers in both columns. The output of Listing 3.4 is here:

```
a    b
0      1.0  2.0
1      1.0  1.0
2      4.0  3.0
3      3.0  1.0
4      1.0  2.0
sum   10.0  9.0
mean   2.0  1.8
```

Listing 3.5 displays the contents of `pandas_combine_df.py` that illustrates how to define a `Pandas DataFrame` that is based on two `NumPy` arrays of numbers.

LISTING 3.5: pandas_combine_df.py

```
import pandas as pd
import numpy as np

df = pd.Data frame({'foo1' : np.random.randn(5),
                    'foo2' : np.random.randn(5)})

print("contents of df:")
print(df)

print("contents of foo1:")
print(df.foo1)

print("contents of foo2:")
print(df.foo2)
```

Listing 3.5 defines the `Pandas DataFrame df` that consists of 5 rows and 2 columns (labeled “`foo1`” and “`foo2`”) of random real numbers between 0 and 5. The next portion of Listing 3.5 displays the contents of `df` and `foo1`. The output of Listing 3.5 is here:

```
contents of df:
       foo1      foo2
0  0.274680 _0.848669
1 _0.399771 _0.814679
2  0.454443 _0.363392
3  0.473753  0.550849
4 _0.211783 _0.015014
contents of foo1:
0    0.256773
1    1.204322
2    1.040515
3   _0.518414
4    0.634141
Name: foo1, dtype: float64
contents of foo2:
0   _2.506550
```

```
1   _0.896516
2   _0.222923
3    0.934574
4    0.527033
Name: foo2, dtype: float64
```

CONVERTING CATEGORICAL DATA TO NUMERIC DATA

One common task in machine learning involves converting a feature containing character data into a feature that contains numeric data.

Listing 3.6 displays the contents of `sometext.tsv` that contains labeled data (spam or ham), which is used in the code sample displayed in Listing 3.7.

LISTING 3.6: sometext.tsv

```
type    text
ham     Available only for today
ham     I'm joking with you
spam    Free entry in 2 a wkly comp
ham     U dun say so early hor
ham     I don't think he goes to usf
spam    FreeMsg Hey there
ham     my brother is not sick
ham     As per your request Melle
spam    WINNER!! As a valued customer
```

Listing 3.7 displays the contents of `cat2numeric.py` that illustrates how to replace a text field with a corresponding numeric field.

LISTING 3.7: cat2numeric.py

```
import pandas as pd
import numpy as np

df = pd.read_csv('sometext.tsv', delimiter='\t')

print("=> First five rows (before):")
print(df.head(5))
print("-------------------------")

# map ham/spam to 0/1 values:
df['type'] = df['type'].map( {'ham':0 , 'spam':1} )

print("=> First five rows (after):")
print(df.head(5))
print("-------------------------")
```

Listing 3.7 initializes the data frame `df` with the contents of the `CSV` file `sometext.tsv`, and then displays the contents of the first five rows by invoking `df.head(5)`, which is also the default number of rows to display. The next code snippet in Listing 3.7 invokes the `map()` method to replace occurrences of `ham` with 0 and replace occurrences of `spam` with 1 in the column labeled type, as shown here:

```
df['type'] = df['type'].map( {'ham':0 , 'spam':1} )
```

The last portion of Listing 3.7 invokes the `head()` method again to display the first five rows of the dataset after having renamed the contents of the column type. Launch the code in Listing 3.7 and you will see the following output:

```
=> First five rows (before):
   type                          text
0   ham      Available only for today
1   ham            I'm joking with you
2  spam  Free entry in 2 a wkly comp
3   ham         U dun say so early hor
4   ham  I don't think he goes to usf
-------------------------
=> First five rows (after):
   type                          text
0     0      Available only for today
1     0            I'm joking with you
2     1  Free entry in 2 a wkly comp
3     0         U dun say so early hor
4     0  I don't think he goes to usf
-------------------------
```

As another example, Listing 3.8 displays the contents of `shirts.csv` and Listing 3.9 displays the contents of `shirts.py` that illustrates four techniques for converting categorical data to numeric data.

LISTING 3.8: shirts.csv

```
type,ssize
shirt,xxlarge
shirt,xxlarge
shirt,xlarge
shirt,xlarge
shirt,xlarge
shirt,large
shirt,medium
shirt,small
shirt,small
shirt,xsmall
shirt,xsmall
shirt,xsmall
```

LISTING 3.9: shirts.py

```
import pandas as pd

shirts = pd.read_csv("shirts.csv")
print("shirts before:")
print(shirts)
print()
```

```
# TECHNIQUE #1:
#shirts.loc[shirts['ssize']=='xxlarge','size'] = 4
#shirts.loc[shirts['ssize']=='xlarge', 'size'] = 4
#shirts.loc[shirts['ssize']=='large',  'size'] = 3
#shirts.loc[shirts['ssize']=='medium', 'size'] = 2
#shirts.loc[shirts['ssize']=='small',  'size'] = 1
#shirts.loc[shirts['ssize']=='xsmall', 'size'] = 1

# TECHNIQUE #2:
#shirts['ssize'].replace('xxlarge', 4, inplace=True)
#shirts['ssize'].replace('xlarge',  4, inplace=True)
#shirts['ssize'].replace('large',   3, inplace=True)
#shirts['ssize'].replace('medium',  2, inplace=True)
#shirts['ssize'].replace('small',   1, inplace=True)
#shirts['ssize'].replace('xsmall',  1, inplace=True)

# TECHNIQUE #3:
#shirts['ssize'] = shirts['ssize'].apply({'xxlarge':4, 'xlarge':4,
'large':3, 'medium':2, 'small':1, 'xsmall':1}.get)

# TECHNIQUE #4:
shirts['ssize'] = shirts['ssize'].replace(regex='xlarge', value=4)
shirts['ssize'] = shirts['ssize'].replace(regex='large',  value=3)
shirts['ssize'] = shirts['ssize'].replace(regex='medium', value=2)
shirts['ssize'] = shirts['ssize'].replace(regex='small',  value=1)

print("shirts after:")
print(shirts)
```

Listing 3.9 starts with a code block of six statements that uses direct comparison with strings to make numeric replacements. For example, the following code snippet replaces all occurrences of the string `xxlarge` with the value 4:

```
shirts.loc[shirts['ssize']=='xxlarge','size'] = 4
```

The second code block consists of six statements that use the `replace()` method to perform the same updates, an example of which is shown here:

```
shirts['ssize'].replace('xxlarge', 4, inplace=True)
```

The third code block consists of a single statement that use the `apply()` method to perform the same updates, as shown here:

```
shirts['ssize'] = shirts['ssize'].apply({'xxlarge':4,
'xlarge':4, 'large':3, 'medium':2, 'small':1, 'xsmall':1}.
get)
```

The fourth code block consists of four statements that use regular expressions to perform the same updates, an example of which is shown here:

```
shirts['ssize'] = shirts['ssize'].replace(regex='xlarge', value=4)
```

Since the preceding code snippet matches `xxlarge` as well as `xlarge`, we only need four statements instead of six statements. If you are unfamiliar with regular expressions, you can find articles online that contains an assortment of regular expressions. Now launch the code in Listing 3.9 and you will see the following output:

```
shirts before
     type     size
0   shirt  xxlarge
1   shirt  xxlarge
2   shirt   xlarge
3   shirt   xlarge
4   shirt   xlarge
5   shirt    large
6   shirt   medium
7   shirt    small
8   shirt    small
9   shirt   xsmall
10  shirt   xsmall
11  shirt   xsmall

shirts after:
     type  size
0   shirt     4
1   shirt     4
2   shirt     4
3   shirt     4
4   shirt     4
5   shirt     3
6   shirt     2
7   shirt     1
8   shirt     1
9   shirt     1
10  shirt     1
11  shirt     1
```

MERGING AND SPLITTING COLUMNS IN PANDAS

Listing 3.10 displays the contents of `employees.csv` and Listing 3.11 displays the contents of `emp_merge_split.py` that illustrates how to merge columns and split columns of a `CSV` file.

LISTING 3.10: employees.csv

```
name,year,month
Jane-Smith,2015,Aug
Dave-Smith,2020,Jan
```

```
Jane-Jones,2018,Dec
Jane-Stone,2017,Feb
Dave-Stone,2014,Apr
Mark-Aster,,Oct
Jane-Jones,NaN,Jun
```

LISTING 3.11: emp_merge_split.py

```
import pandas as pd

emps = pd.read_csv("employees.csv")
print("emps:")
print(emps)
print()

emps['year']  = emps['year'].astype(str)
emps['month'] = emps['month'].astype(str)

# separate column for first name and for last name:
emps['fname'],emps['lname'] = emps['name'].str.split("-",1).str

# concatenate year and month with a "#" symbol:
emps['hdate1'] = emps['year'].astype(str)+"#"+emps['month'].astype(str)

# concatenate year and month with a "-" symbol:
emps['hdate2'] = emps[['year','month']].agg('-'.join, axis=1)

print(emps)
print()
```

Listing 3.11 initializes the data frame `df` with the contents of the `CSV` file `employees.csv`, and then displays the contents of the data frame. The next pair of code snippets invoke the `astype()` method to convert the contents of the `year` and `month` columns to strings.

The next code snippet in Listing 3.11 uses the `split()` method to split the name column into the columns `fname` and `lname` that contain the first name and last name, respectively, of each employee's name:

```
emps['fname'],emps['lname'] = emps['name'].str.split("-",1).str
```

The next code snippet concatenates the contents of the year and month string with a "#" character to create a new column called `hdate1`, as shown here:

```
emps['hdate1'] = emps['year'].astype(str)+"#"+emps['month'].astype(str)
```

The final code snippet concatenates the contents of the `year` and `month` string with a "-" to create a new column called `hdate2`, as shown here:

```
emps['hdate2'] = emps[['year','month']].agg('-'.join, axis=1)
```

Now launch the code in Listing 3.11 and you will see the following output: emps:

```
         name    year month
0  Jane-Smith  2015.0   Aug
1  Dave-Smith  2020.0   Jan
2  Jane-Jones  2018.0   Dec
3  Jane-Stone  2017.0   Feb
4  Dave-Stone  2014.0   Apr
5  Mark-Aster     NaN   Oct
6  Jane-Jones     NaN   Jun

         name    year month fname  lname      hdate1      hdate2
0  Jane-Smith  2015.0   Aug  Jane  Smith  2015.0#Aug  2015.0-Aug
1  Dave-Smith  2020.0   Jan  Dave  Smith  2020.0#Jan  2020.0-Jan
2  Jane-Jones  2018.0   Dec  Jane  Jones  2018.0#Dec  2018.0-Dec
3  Jane-Stone  2017.0   Feb  Jane  Stone  2017.0#Feb  2017.0-Feb
4  Dave-Stone  2014.0   Apr  Dave  Stone  2014.0#Apr  2014.0-Apr
5  Mark-Aster     nan   Oct  Mark  Aster     nan#Oct     nan-Oct
6  Jane-Jones     nan   Jun  Jane  Jones     nan#Jun     nan-Jun
```

One other detail regarding the following code snippet:

```
#emps['fname'],emps['lname'] = emps['name'].str.split("-",1).str
```

The following deprecation message is displayed:

```
#FutureWarning: Columnar iteration over characters
#will be deprecated in future releases.
```

COMBINING PANDAS DATAFRAMES

`Pandas` supports the concat() method in `Data frames` in order to concatenate `Data frames`. Listing 3.12 displays the contents of `concat_frames.py` that illustrates how to combine two `Pandas DataFrames`.

LISTING 3.12: concat_frames.py

```
import pandas as pd

can_weather = pd.Data frame({
    "city": ["Vancouver","Toronto","Montreal"],
    "temperature": [72,65,50],
    "humidity": [40, 20, 25]
})

us_weather = pd.Data frame({
    "city": ["SF","Chicago","LA"],
    "temperature": [60,40,85],
    "humidity": [30, 15, 55]
})

df = pd.concat([can_weather, us_weather])
print(df)
```

The first line in Listing 3.12 is an `import` statement, followed by the definition of the `Pandas DataFrames` `can_weather` and `us_weather` that contain weather-related information for cities in Canada and the United States, respectively. The `Pandas DataFrame df` is the concatenation of `can_weather` and `us_weather`. The output from Listing 3.12 is here:

```
0  Vancouver        40           72
1    Toronto        20           65
2   Montreal        25           50
0         SF        30           60
1    Chicago        15           40
2         LA        55           85
```

DATA MANIPULATION WITH PANDAS DATAFRAMES

As a simple example, suppose that we have a two-person company that keeps track of income and expenses on a quarterly basis, and we want to calculate the profit/loss for each quarter, and also the overall profit/loss.

Listing 3.13 displays the contents of `pandas_quarterly_df1.py` that illustrates how to define a `Pandas DataFrame` consisting of income-related values.

LISTING 3.13: pandas_quarterly_df1.py

```
import pandas as pd

summary = {
    'Quarter': ['Q1', 'Q2', 'Q3', 'Q4'],
    'Cost':    [23500, 34000, 57000, 32000],
    'Revenue': [40000, 40000, 40000, 40000]
}

df = pd.Data frame(summary)

print("Entire Dataset:\n",df)
print("Quarter:\n",df.Quarter)
print("Cost:\n",df.Cost)
print("Revenue:\n",df.Revenue)
```

Listing 3.13 defines the variable `summary` that contains hard-coded quarterly information about cost and revenue for our two-person company. In general these hard-coded values would be replaced by data from another source (such as a `CSV` file), so think of this code sample as a simple way to illustrate some of the functionality that is available in `Pandas DataFrames`.

The variable `df` is a `Pandas DataFrame` based on the data in the `summary` variable. The three `print()` statements display the quarters, the cost per quarter, and the revenue per quarter. The output from Listing 3.13 is here:

```
Entire Dataset:
     Cost Quarter  Revenue
0  23500      Q1    40000
1  34000      Q2    60000
```

```
2  57000      Q3    50000
3  32000      Q4    30000
Quarter:
0    Q1
1    Q2
2    Q3
3    Q4
Name: Quarter, dtype: object
Cost:
0    23500
1    34000
2    57000
3    32000
Name: Cost, dtype: int64
Revenue:
0    40000
1    60000
2    50000
3    30000
Name: Revenue, dtype: int64
```

PANDAS DATAFRAMES AND CSV FILES

The code samples in several earlier sections contain hard-coded data inside the `Python` scripts. However, it is also very common to read data from a `CSV` file. You can use the `Python` `csv.reader()` function, the `NumPy` `loadtxt()` function, or the `Pandas` function `read_csv()` function (shown in this section) to read the contents of `CSV` files.

Listing 3.14 displays the contents of the `CSV` file `weather_data.csv` and Listing 3.15 displays the contents of `weather_data.py` that illustrates how to read the `CSV` `weather_data.csv`.

LISTING 3.14: weather_data.csv

```
day,temperature,windspeed,event
7/1/2021,42,16,Rain
7/2/2021,45,3,Sunny
7/3/2021,78,12,Snow
7/4/2021,74,9,Snow
7/5/2021,42,24,Rain
7/6/2021,51,32,Sunny
```

LISTING 3.15: weather_data.py

```
import pandas as pd

df = pd.read_csv("weather_data.csv")

print(df)
print(df.shape)  # rows, columns
print(df.head()) # df.head(3)
print(df.tail())
print(df[1:3])
print(df.columns)
print(type(df['day']))
print(df[['day','temperature']])
print(df['temperature'].max())
```

Listing 3.15 invokes the `Pandas read_csv()` function to read the contents of the `CSV` file `weather_data.csv`, followed by a set of `Python print()` statements that display various portions of the `CSV` file. The output from Listing 3.15 is here:

```
        day  temperature  windspeed    event
0  7/1/2021           42         16   Rain
1  7/2/2021           45          3  Sunny
2  7/3/2021           78         12   Snow
3  7/4/2021           74          9  Snow
4  7/5/2021           42         24   Rain
5  7/6/2021           51         32    Sunny
(6, 4)
        day  temperature  windspeed    event
0  7/1/2021           42         16   Rain
1  7/2/2021           45          3  Sunny
2  7/3/2021           78         12   Snow
3  7/4/2021           74          9  Snow
4  7/5/2021           42         24   Rain
        day  temperature  windspeed    event
1  7/2/2021           45          3  Sunny
2  7/3/2021           78         12   Snow
3  7/4/2021           74          9  Snow
4  7/5/2021           42         24   Rain
5  7/6/2021           51         32    Sunny
        day  temperature  windspeed    event
1  7/2/2021           45          3  Sunny
2  7/3/2021           78         12   Snow
Index(['day', 'temperature', 'windspeed', 'event'], dtype='object')
<class 'pandas.core.series.Series'>
        day  temperature
0  7/1/2021           42
1  7/2/2021           45
2  7/3/2021           78
3  7/4/2021           74
4  7/5/2021           42
5  7/6/2021           51
78
```

In some situations you might need to apply `Boolean` conditional logic to "filter out" some rows of data, based on a conditional condition that's applied to a column value.

Listing 3.16 displays the contents of the `CSV` file `people.csv` and Listing 3.17 displays the contents of `people_pandas.py` that illustrates how to define a `Pandas DataFrame` that reads the `CSV` file and manipulates the data.

LISTING 3.16: people.csv

```
fname,lname,age,gender,country
john,smith,30,m,usa
jane,smith,31,f,france
jack,jones,32,m,france
dave,stone,33,m,italy
sara,stein,34,f,germany
eddy,bower,35,m,spain
```

LISTING 3.17: people_pandas.py

```
import pandas as pd

df = pd.read_csv('people.csv')
df.info()
print('fname:')
print(df['fname'])
print('____________')
print('age over 33:')
print(df['age'] > 33)
print('____________')
print('age over 33:')
myfilter = df['age'] >  33
print(df[myfilter])
```

Listing 3.17 populates the `Pandas DataFrame df` with the contents of the `CSV` file `people.csv`. The next portion of Listing 3.17 displays the structure of `df`, followed by the first names of all the people. The next portion of Listing 3.17 displays a tabular list of six rows containing either "True" or "False" depending on whether a person is over 33 or at most 33, respectively. The final portion of Listing 3.17 displays a tabular list of two rows containing all the details of the people who are over 33. The output from Listing 3.17 is here:

```
myfilter = df['age'] >  33
<class 'pandas.core.frame.Data frame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
fname      6 non_null object
lname      6 non_null object
age        6 non_null int64
gender     6 non_null object
country    6 non_null object
dtypes: int64(1), object(4)
memory usage: 320.0+ bytes
fname:
0    john
1    jane
2    jack
3    dave
```

```
4    sara
5    eddy
Name: fname, dtype: object
____________
age over 33:
0    False
1    False
2    False
3    False
4     True
5     True
Name: age, dtype: bool
____________
age over 33:
  fname  lname  age gender country
4  sara  stein   34      f  france
5  eddy  bower   35      m  france
```

Useful Options for the Pandas read_csv() Function

Skip the initial header information contained in the first row with this code snippet:

```
df = pd.read_csv("data.csv", header=None)
```

The following code snippet shows you how to save a `Pandas DataFrame` to a `CSV` file *without* including the indices:

```
df.to_csv("data.csv", sep=",", index=False)
```

If need be, you can remove the first line immediately following the header row with this code snippet:

```
my_dataset = pd.read_csv("dataset.csv", skiprows=1, low_memory=False)
```

Skip the first three rows of a CSV file:

```
df = pd.read_csv("data.csv", skiprows=3, header=None)
```

Skip a range of rows that are specified by index:

```
df = pd.read_csv("data.csv", skiprows=[0,2])
```

Reading Selected Rows From CSV Files

You have seen `Pandas`-based examples of reading the entire contents of `CSV` files into a `Pandas DataFrame`, and then selecting subsets of those rows for additional processing. In this section, you will see how to read portions of `CSV` files, which eliminates the need to drop redundant rows from `Pandas DataFrames`.

This technique involves reading portions of `CSV` files by specifying the `chunksize` parameter. This is a useful method for large datasets: `Pandas` will process the dataset in sequential chunks without reading the entire file into memory. Although the CSV dataset in this example is very small, you now know how to specify this parameter.

Listing 3.18 displays the contents of `fruits.csv` that is referenced in Listing 3.19 that retrieves a subset of rows from `fruits.csv`.

LISTING 3.18: fruits.csv

```
name,month,day
avocado,Aug,13
persimmon,Jul,28
apples,Sept,25
oranges,Aug,30
bananas,Dec,20
cantelope,Nov,18
```

Listing 3.19 displays the contents of `pandas_csv1.py` that illustrates how to read a subset of rows from a CSV file based on some conditional logic.

LISTING 3.19: pandas_csv1.py

```
import pandas as pd

csv_file="fruits.csv"

df1 = pd.read_csv(csv_file)
print("df1 set of rows:")
print(df1)
print()

df1 = pd.read_csv(csv_file, chunksize=10000000)
df2 = pd.concat((item.query("name == 'oranges'") for item
in df1), ignore_index=True)
print("df2:")
print(df2)
print()
```

Listing 3.19 initialized the string variable `csv_file` with the value of `fruits.csv`, populates the `Pandas DataFrame df1` with the contents of `fruits.csv`, and then displays the contents of `df1`.

The next portion of Listing 3.19 initializes the `Pandas DataFrame df2` with the subset of rows in `df1` whose `name` attribute equals `oranges`. Launch the code in Listing 3.19 and you will see the following output:

```
df1 set of rows:
        name month  day
0    avocado   Aug   13
1  persimmon   Jul   28
2     apples  Sept   25
3    oranges   Aug   30
4    bananas   Dec   20
5  cantelope   Nov   18
```

```
df2 rows with oranges:
      name month  day
0  oranges   Aug   30
```

Listing 3.20 displays the contents of `pandas_schema1.py` that illustrates how to read a subset of rows from a `CSV` file based on some conditional logic.

LISTING 3.20: pandas_schema1.py

```
import pandas as pd

csv_file="emp_ages.csv"

schema = { "age": int }

df1 = pd.read_csv(csv_file, dtype=schema, chunksize=10000000)
df2 = pd.concat((item.query("'age' >= 45") for item in
df1), ignore_index=True)

print("df2 ages at least 45:")
print(df2)
```

Listing 3.20 initializes the string variable `csv_file` with the value `emp_ages.csv` and then initializes the string variable schema with a `JSON`-based string. The next code snippet initializes the `Pandas DataFrames` `df1` with the contents of the `CSV` file `emp_ages.csv`.

Next, the `Pandas DataFrame` `df2` is initialized with the subset of rows in `df1` whose `age` attribute is at least 45. The back quotes in this code snippet are required when you specify an attribute that has an embedded whitespace. Launch the code in Listing 3.20 and you will see the following output:

```
df2 ages at least 45:
  fname  lname  age
0  Jane  Jones   65
1  Jane  Jones   65
2  Dave  Stone   45
3  Mark  Aster   53
4  Jane  Jones   58
```

Listing 3.21 displays the contents of `pandas_schema2.py` that illustrates how to read a subset of rows from a `CSV` file based on some conditional logic.

LISTING 3.21: pandas_schema2.py

```
import pandas as pd

csv_file="emp_ages.csv"

schema = { "age": int, "fname":str}
df1 = pd.read_csv(csv_file, dtype=schema, chunksize=10000000)
```

```
df2 = pd.concat((item.query("age >= 45 | age < 20") for
item in df1), ignore_index=True)

print("df2 ages at least 45 or less than 20:")
print(df2)
```

Listing 3.21 extends the code in Listing 3.20 by specifying a compound condition for the rows in the `Pandas DataFrame df2`, which involves the rows in `df1` whose `age` attribute is at least 45 *or* the rows in `df1` whose `age` attribute is *less than* 20. Launch the code in Listing 3.21 and you will see the following output:

```
df2 ages at least 45 or less than 20:
  fname  lname  age
0  Dave  Smith   10
1  Jane  Jones   65
2  Jane  Jones   65
3  Dave  Stone   45
4  Mark  Aster   53
5  Jane  Jones   58
```

PANDAS DATAFRAMES AND EXCEL SPREADSHEETS

Listing 3.22 displays the contents of `write_people_xlsx.py` that illustrates how to read data from a `CSV` file and then create an `Excel` spreadsheet with that data.

LISTING 3.22: write_people_xlsx.py

```
import pandas as pd

df1 = pd.read_csv("people.csv")
df1.to_excel("people.xlsx")

#optionally specify the sheet name:
#df1.to_excel("people.xlsx", sheet_name='Sheet_name_1')
```

Listing 3.22 initializes the `Pandas DataFrame df1` with the contents of the `CSV` file `people.csv`, and then invokes the `to_excel()` method in order to save the contents of the data frame to the `Excel` spreadsheet `people.xlsx`.

Listing 3.23 displays the contents of `read_people_xlsx.py` that illustrates how to read data from the `Excel` spreadsheet `people.xlsx` and create a `Pandas DataFrame` with that data.

LISTING 3.23: read_people_xlsx.py

```
import pandas as pd

df = pd.read_excel("people.xlsx")
print("Contents of Excel spreadsheet:")
print(df)
```

Listing 3.23 is straightforward: the Pandas DataFrame df is initialized with the contents of the spreadsheet people.xlsx (whose contents are the same as people.csv) via the Pandas function read_excel(). The output from Listing 3.23 is here:

```
df1:
   Unnamed: 0 fname  lname  age gender  country
0           0  john  smith   30      m      usa
1           1  jane  smith   31      f   france
2           2  jack  jones   32      m   france
3           3  dave  stone   33      m    italy
4           4  sara  stein   34      f  germany
5           5  eddy  bower   35      m    spain
```

Useful Options for Reading Excel Spreadsheets

Sometimes you need extra control over the values that you read from an Excel spreadsheet into a Pandas DataFrame, just as you do with CSV files.

Skip the header and the footer in an Excel spreadsheet with this code snippet:

```
df = pd.read_excel("myfile.xls",header=15,skipfooter=_Y_)
```

SELECT, ADD, AND DELETE COLUMNS IN DATA FRAMES

This section contains short code blocks that illustrate how to perform operations on a Data frame that resemble the operations on a Python dictionary. For example, getting, setting, and deleting columns works with the same syntax as the analogous Python dict operations, as shown here:

```
df = pd.Data frame.from_dict(dict([('A',[1,2,3]),('B',[4,5,6])]),
                orient='index', columns=['one', 'two', 'three'])

print(df)
```

The output from the preceding code snippet is here:

```
   one  two  three
A    1    2      3
B    4    5      6
```

Now look at the following operation that appends a new column to the contents of the data frame df:

```
df['four'] = df['one'] * df['two']
print(df)
```

The output from the preceding code block is here:

```
   one  two  three  four
A    1    2      3     2
B    4    5      6    20
```

The following operation squares the contents of a column in the data frame `df`:

```
df['three'] = df['two'] * df['two']
print(df)
```

The output from the preceding code block is here:

```
   one  two  three  four
A    1    2      4     2
B    4    5     25    20
```

The following operation inserts a column of random numbers in index position 1 (which is the second column) in the data frame `df`:

```
import numpy as np
rand = np.random.randn(2)
df.insert(1, 'random', rand)
print(df)
```

The output from the preceding code block is here:

```
   one    random  two  three  four
A    1 -1.703111    2      4     2
B    4  1.139189    5     25    20
```

The following operation appends a new column called flag that contains True or False, based on whether or not the numeric value in the "one" column is greater than 2:

```
import numpy as np
rand = np.random.randn(2)
df.insert(1, 'random', rand)
print(df)
```

The output from the preceding code block is here:

```
   one    random  two  three  four   flag
A    1 -1.703111    2      4     2  False
B    4  1.139189    5     25    20   True
```

Columns can be deleted, as shown in following code snippet that deletes the "two" column:

```
del df['two']
print(df)
```

The output from the preceding code block is here:

```
one    random  three  four   flag

A    1 -0.460401      4     2  False
B    4  1.211468     25    20   True
```

Columns can be deleted via the `pop()` method, as shown in following code snippet that deletes the "three" column:

```
three = df.pop('three')
print(df)

   one    random  four   flag
A    1 -0.544829     2  False
B    4  0.581476    20   True
```

When inserting a scalar value, it will naturally be propagated to fill the column:

```
df['foo'] = 'bar'
print(df)
```

The output from the preceding code snippet is here:

```
   one    random  four   flag  foo
A    1 -0.187331     2  False  bar
B    4 -0.169672    20   True  bar
```

HANDLING OUTLIERS IN PANDAS

If you are unfamiliar with outliers and anomalies, please read online articles that discuss these two concepts because this section uses `Pandas` to find outliers in a dataset. The key idea involves finding the "z score" of the values in the dataset, which involves calculating the mean `sigma` and standard deviation `std`, and then mapping each value x in the dataset to the value `(x-sigma)/std`.

Next, you specify a value of `z` (such as 3) and find the rows whose z score is greater than 3. These are the rows that contain values that are considered outliers. *Note that a suitable value for the z score is your decision (not some other external factor).*

```
pandas-outliers1.py
pandas-outliers2.py
pandas-outliers3.py
```

Listing 3.24 displays the contents of `outliers_zscores.py` that illustrates how to find rows of a dataset whose z-score greater than (or less than) a specified value.

LISTING 3.24: outliers_zscores.py

```
import numpy as np
import pandas as pd
from scipy import stats
from sklearn import datasets

df = datasets.load_iris()
columns = df.feature_names
iris_df = pd.Data frame(df.data)
iris_df.columns = columns
```

```
print("=> iris_df.shape:",iris_df.shape)
print(iris_df.head())
print()

z = np.abs(stats.zscore(iris_df))
print("z scores for iris:")
print("z.shape:",z.shape)

upper = 2.5
lower = 0.01
print("=> upper outliers:")
print(z[np.where(z > upper)])
print()

outliers = iris_df[z < lower]
print("=> lower outliers:")
print(outliers)
print()
```

Listing 3.24 initializes the variable `df` with the contents of the built-in `Iris` dataset. Next, the variable columns is initialized with the column names, and the data frame `iris_df` is initialized from the contents of `df.data` that contains the actual data for the `Iris` dataset. In addition, `iris_df.columns` is initialized with the contents of the variable `columns`.

The next portion of Listing 3.24 displays the shape of the data frame `iris_df`, followed by the `zscore` of the `iris_df` data frame, which is computed by subtracting the mean and then dividing by the standard deviation (performed for each row).

The last two portions of Listing 3.24 display the outliers (if any) whose `zscore` is outside the interval [0.01, 2.5]. Launch the code in Listing 3.24 and you will see the following output:

```
=> iris_df.shape: (150, 4)
   sepal length (cm)  sepal width (cm)  petal length (cm)
petal width (cm)
0                5.1               3.5                1.4
0.2
1                4.9               3.0                1.4
0.2
2                4.7               3.2                1.3
0.2
3                4.6               3.1                1.5
0.2
4                5.0               3.6                1.4
0.2

z scores for iris:
z.shape: (150, 4)

=> upper outliers:
[3.09077525 2.63038172]

=> lower outliers:
    sepal length (cm)  sepal width (cm)  petal length (cm)
petal width (cm)
73                6.1               2.8                4.7
1.2
82                5.8               2.7                3.9
1.2
90                5.5               2.6                4.4
1.2
92                5.8               2.6                4.0
1.2
95                5.7               3.0                4.2
1.2
```

PANDAS DATAFRAMES AND SIMPLE STATISTICS

Listing 3.25 displays the contents of `housing_stats.py` that illustrates how to gather basic statistics from data in a `Pandas DataFrame`.

LISTING 3.25: housing_stats.py

```
import pandas as pd

df = pd.read_csv("Housing.csv")

minimum_bdrms = df["bedrooms"].min()
median_bdrms  = df["bedrooms"].median()
maximum_bdrms = df["bedrooms"].max()

print("minimum # of bedrooms:",minimum_bdrms)
print("median  # of bedrooms:",median_bdrms)
print("maximum # of bedrooms:",maximum_bdrms)
print("")

print("median values:",df.median().values)
print("")

prices = df["price"]
print("first 5 prices:")
print(prices.head())
print("")

median_price = df["price"].median()
print("median price:",median_price)
print("")

corr_matrix = df.corr()
print("correlation matrix:")
print(corr_matrix["price"].sort_values(ascending=False))
```

Listing 3.25 initializes the `Pandas DataFrame df` with the contents of the `CSV` file `housing.csv`. The next three variables are initialized with the minimum, median, and maximum number of bedrooms, respectively, and then these values are displayed.

The next portion of Listing 3.25 initializes the variable `prices` with the contents of the `prices` column of the `Pandas DataFrame df`. Next, the first five rows are printed via the `prices.head()` statement, followed by the median value of the prices.

The final portion of Listing 3.25 initializes the variable `corr_matrix` with the contents of the correlation matrix for the `Pandas DataFrame df`, and then displays its contents. The output from Listing 3.25 is here:

```
Apples
10
```

FINDING DUPLICATE ROWS IN PANDAS

Listing 3.26 displays the contents of `duplicates.csv` and Listing 3.27 displays the contents of `duplicates.py` that illustrates how to find duplicate rows in a `Pandas DataFrame`.

LISTING 3.26: duplicates.csv

```
fname,lname,level,dept,state
Jane,Smith,Senior,Sales,California
Dave,Smith,Senior,Devel,California
Jane,Jones,Year1,Mrktg,Illinois
Jane,Jones,Year1,Mrktg,Illinois
Jane,Stone,Senior,Mrktg,Arizona
Dave,Stone,Year2,Devel,Arizona
Mark,Aster,Year3,BizDev,Florida
Jane,Jones,Year1,Mrktg,Illinois
```

LISTING 3.27: duplicates.py

```
import pandas as pd

df = pd.read_csv("duplicates.csv")
print("Contents of data frame:")
print(df)
print()

print("Duplicate rows:")
#df2 = df.duplicated(subset=None)
df2 = df.duplicated(subset=None, keep='first')
print(df2)
print()

print("Duplicate first names:")
df3 = df[df.duplicated(['fname'])]
print(df3)
print()

print("Duplicate first name and level:")
df3 = df[df.duplicated(['fname','level'])]
print(df3)
print()
```

Listing 3.27 initializes the data frame `df` with the contents of the CSV file `duplicates.csv`, and then displays the contents of the data frame. The next portion of Listing 3.27 displays the duplicate rows by invoking the `duplicated()` method, whereas the next portion of Listing 3.27 displays only the first name `fname` of the duplicate rows. The final portion of Listing 3.27 displays the first name `fname` as well as the level of the duplicate rows. Launch the code in Listing 3.27 and you will see the following output:

```
Contents of data frame:
  fname  lname   level    dept       state
0  Jane  Smith  Senior   Sales  California
1  Dave  Smith  Senior   Devel  California
2  Jane  Jones   Year1   Mrktg    Illinois
3  Jane  Jones   Year1   Mrktg    Illinois
4  Jane  Stone  Senior   Mrktg     Arizona
5  Dave  Stone   Year2   Devel     Arizona
6  Mark  Aster   Year3  BizDev     Florida
7  Jane  Jones   Year1   Mrktg    Illinois

Duplicate rows:
0    False
1    False
2    False
3     True
4    False
5    False
6    False
7     True
dtype: bool

Duplicate first names:
  fname  lname   level   dept     state
2  Jane  Jones   Year1  Mrktg  Illinois
3  Jane  Jones   Year1  Mrktg  Illinois
4  Jane  Stone  Senior  Mrktg   Arizona
5  Dave  Stone   Year2  Devel   Arizona
7  Jane  Jones   Year1  Mrktg  Illinois

Duplicate first name and level:
  fname  lname   level   dept     state
3  Jane  Jones   Year1  Mrktg  Illinois
4  Jane  Stone  Senior  Mrktg   Arizona
7  Jane  Jones   Year1  Mrktg  Illinois
```

Listing 3.28 displays the contents of `drop_duplicates.py` that illustrates how to drop duplicate rows in a `Pandas DataFrame`.

LISTING 3.28: drop_duplicates.py

```
import pandas as pd

df = pd.read_csv("duplicates.csv")
print("Contents of data frame:")
print(df)
print()

fname_filtered = df.drop_duplicates(['fname'])
print("Drop duplicate first names:")
print(fname_filtered)
print()

fname_lname_filtered = df.drop_duplicates(['fname','lname'])
print("Drop duplicate first and last names:")
print(fname_lname_filtered)
print()
```

Listing 3.28 initializes the data frame `df` with the contents of the `CSV` file `duplicates.csv`, and then displays the contents of the data frame. The next portion of Listing 3.28 deletes the rows that have duplicate `fname` values, followed by a code block that drops rows with duplicate `fname` and `lname` values. Launch the code in Listing 3.28 and you will see the following output:

```
Contents of data frame:
  fname  lname   level    dept       state
0  Jane  Smith  Senior   Sales  California
1  Dave  Smith  Senior   Devel  California
2  Jane  Jones   Year1   Mrktg    Illinois
3  Jane  Jones   Year1   Mrktg    Illinois
4  Jane  Stone  Senior   Mrktg     Arizona
5  Dave  Stone   Year2   Devel     Arizona
6  Mark  Aster   Year3  BizDev     Florida
7  Jane  Jones   Year1   Mrktg    Illinois

Drop duplicate first names:
  fname  lname   level    dept       state
0  Jane  Smith  Senior   Sales  California
1  Dave  Smith  Senior   Devel  California
6  Mark  Aster   Year3  BizDev     Florida

Drop duplicate first and last names:
  fname  lname   level    dept       state
0  Jane  Smith  Senior   Sales  California
1  Dave  Smith  Senior   Devel  California
2  Jane  Jones   Year1   Mrktg    Illinois
4  Jane  Stone  Senior   Mrktg     Arizona
5  Dave  Stone   Year2   Devel     Arizona
6  Mark  Aster   Year3  BizDev     Florida
```

FINDING MISSING VALUES IN PANDAS

Listing 3.29 displays the contents of `employees2.csv` and Listing 3.30 displays the contents of `missing_values.py` that illustrates how to display rows of a data frame that have missing values in a `Pandas DataFrame`.

LISTING 3.29: employees2.csv

```
name,year,month
Jane-Smith,2015,Aug
Jane-Smith,2015,Aug
Dave-Smith,2020,
Dave-Stone,,Apr
Jane-Jones,2018,Dec
Jane-Stone,2017,Feb
Jane-Stone,2017,Feb
Mark-Aster,,Oct
Jane-Jones,NaN,Jun
```

LISTING 3.30: missing_values.py

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv("employees2.csv")

print("=> contents of CSV file:")
print(df)
print()

#NA:  Not Available (Pandas)
#NaN: Not a Number (Pandas)
#NB:  NumPy uses np.nan() to check for NaN values

df = pd.read_csv("employees2.csv")

print("=> contents of CSV file:")
print(df)
print()

print("=> any NULL values per column?")
print(df.isnull().any())
print()

print("=> count of NAN/MISSING values in each column:")
print(df.isnull().sum())
print()

print("=> count of NAN/MISSING values in each column:")
print(pd.isna(df).sum())
print()

print("=> count of NAN/MISSING values in each column (sorted):")
print(df.isnull().sum().sort_values(ascending=False))
print()

nan_null = df.isnull().sum().sum()
miss_values = df.isnull().any().sum()
```

```
print("=> count of NaN/MISSING values:",nan_null)
print("=> count of MISSING values:",miss_values)
print("=> count of NaN values:",nan_null-miss_values)
```

Listing 3.30 initializes the data frame `df` with the contents of the `CSV` file `employees2.csv`, and then displays the contents of the data frame. The next portion of Listing 3.30 displays the number of null values that appear in any row or column. The next portion of Listing 3.30 displays the fields and the names of the fields that have null values.

The next portion of Listing 3.30 displays the number of duplicate rows, followed by the row numbers that are duplicates. Launch the code in Listing 3.30 and you will see the following output:

```
=> contents of CSV file:
         name    year month
0  Jane-Smith  2015.0   Aug
1  Jane-Smith  2015.0   Aug
2  Dave-Smith  2020.0   NaN
3  Dave-Stone     NaN   Apr
4  Jane-Jones  2018.0   Dec
5  Jane-Stone  2017.0   Feb
6  Jane-Stone  2017.0   Feb
7  Mark-Aster     NaN   Oct
8  Jane-Jones     NaN   Jun

=> any NULL values per column?
name     False
year      True
month     True
dtype: bool

=> count of NAN/MISSING values in each column:
name     0
year     3
month    1
dtype: int64

=> count of NAN/MISSING values in each column:
name     0
year     3
month    1
dtype: int64

=> count of NAN/MISSING values in each column (sorted):
year     3
month    1
name     0
dtype: int64

=> count of NaN/MISSING values: 4
=> count of MISSING values: 2
=> count of NaN values: 2
```

MISSING VALUES IN IRIS-BASED DATASET

This section shows you how to replace missing values in the dataset `nan_iris.csv` that was created as follows:

- Copy the header and the first 50 data rows of the `Iris` dataset.
- Substitute `NaN` in randomly selected rows and columns.

For your convenience, the `iris.csv` dataset and the `nan_iris.csv` dataset are included in the companion files.

Listing 3.32 displays an initial portion of the contents of `nan_iris.csv`, whereas Listing 3.33 displays the contents of `missingdatairis.py` that illustrates how to replace the `NaN` values with meaningful values.

LISTING 3.32: nan_iris.csv

```
SepalLength,SepalWidth,PetalLength,PetalWidth,Name
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3,1.4,0.2,Iris-setosa
NaN,3.2,NaN,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5,3.6,1.4,0.2,Iris-setosa
NaN,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
NaN,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,NaN,0.1,NaN
// details omitted for brevity
4.5,2.3,1.3,0.3,Iris-setosa
4.4,NaN,NaN,NaN,NaN
5,3.5,NaN,NaN,Iris-setosa
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5,3.3,1.4,0.2,Iris-setosa
```

LISTING 3.33: missingdatairis.py

```
import numpy as np
import pandas as pd

# Step 1:
data = pd.read_csv('nan_iris.csv')

# Step 2:
print("=> Details of dataset columns:")
print(data.info())
print()
```

```
# Step 3:
print("=> Missing values per column:")
print(data.isnull().sum())
print()

# Step 4:
print("=> First range from 40 to 45:")
print(data[40:45])
print()

# Step 5:
print("=> List of Mean Values:")
print(data.mean())
print()

# list of column labels:
# SepalLength SepalWidth PetalLength PetalWidth Name

# Step 6:
# fill numeric columns with the mean (per column):
data.fillna(data.mean(), inplace=True)

# Step 7:
print("=> Second range from 40 to 45:")
print(data[40:45])
print()

# Step 8:
# create a new category for categorical data:
data['Name'] = data['Name'].fillna('UNKNOWN')

# Step 9:
print("=> Third range from 40 to 45:")
print(data[40:45])
```

Listing 3.33 contains various blocks of code with self-explanatory comments that explain the purpose of the code, starting with the first step that reads the contents of `nan_iris.csv` into the data frame `data`, followed by the block of code that displays the details of the dataset.

Next, the third block of code displays the number of missing values in each column of the dataset, followed by a block of code that display the contents of rows 40 through 45.

The fifth block of code displays the mean values for each column in the dataset, followed by a block of code that replaces the missing numeric values with the mean value, on a column-by-column basis, via the following code snippet:

```
data.fillna(data.mean(), inplace=True)
```

The seventh block of code displays the updated contents of the dataset, followed by a block of code that replaces the `NaN` values with `UNKNOWN` in the `Name` column.

The final block of code displays the data in rows 40 through 45, and at this point all the `NaN` values in the data frame have been replaced with a numeric value or the string `UNKNOWN`. Now launch the code in Listing 3.33 and you will see the following output:

```
=> Details of dataset columns:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   SepalLength  39 non-null     float64
 1   SepalWidth   49 non-null     float64
 2   PetalLength  46 non-null     float64
 3   PetalWidth   48 non-null     float64
 4   Name         46 non-null     object
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
None

=> Missing values per column:
SepalLength    11
SepalWidth      1
PetalLength     4
PetalWidth      2
Name            4
dtype: int64

=> First range from 40 to 45:
    SepalLength  SepalWidth  PetalLength  PetalWidth         Name
40          NaN         3.5          1.3         0.3  Iris-setosa
41          4.5         2.3          1.3         0.3  Iris-setosa
42          4.4         NaN          NaN         NaN          NaN
43          5.0         3.5          NaN         NaN  Iris-setosa
44          5.1         3.8          1.9         0.4  Iris-setosa

=> List of Mean Values:
SepalLength    5.002564
SepalWidth     3.422449
PetalLength    1.467391
PetalWidth     0.237500
dtype: float64

=> Second range from 40 to 45:
    SepalLength  SepalWidth  PetalLength  PetalWidth         Name
40     5.002564    3.500000     1.300000      0.3000  Iris-setosa
41     4.500000    2.300000     1.300000      0.3000  Iris-setosa
42     4.400000    3.422449     1.467391      0.2375          NaN
43     5.000000    3.500000     1.467391      0.2375  Iris-setosa
44     5.100000    3.800000     1.900000      0.4000  Iris-setosa

=> Third range from 40 to 45:
    SepalLength  SepalWidth  PetalLength  PetalWidth         Name
40     5.002564    3.500000     1.300000      0.3000  Iris-setosa
```

```
41     4.500000    2.300000     1.300000      0.3000  Iris-setosa
42     4.400000    3.422449     1.467391      0.2375      UNKNOWN
43     5.000000    3.500000     1.467391      0.2375  Iris-setosa
44     5.100000    3.800000     1.900000      0.4000  Iris-setosa
```

SORTING DATA FRAMES IN PANDAS

Listing 3.34 displays the contents of `sort_df.py` that illustrates how to sort the rows in a `Pandas DataFrame`.

LISTING 3.34: sort_df.py

```
import pandas as pd

df = pd.read_csv("duplicates.csv")
print("Contents of data frame:")
print(df)
print()

df.sort_values(by=['fname'], inplace=True)
print("Sorted (ascending) by first name:")
print(df)
print()

df.sort_values(by=['fname'], inplace=True,ascending=False)
print("Sorted (descending) by first name:")
print(df)
print()

df.sort_values(by=['fname','lname'], inplace=True)
print("Sorted (ascending) by first name and last name:")
print(df)
print()
```

Listing 3.34 initializes the data frame `df` with the contents of the `CSV` file `duplicates.csv`, and then displays the contents of the data frame. The next portion of Listing 3.34 displays the rows in *ascending* order based on the first name, and the next code block displays the rows in *descending* order based on the first name. The final code block in Listing 3.34 displays the rows in ascending order based on the first name as well as the last name. Launch the code in Listing 3.34 and you will see the following output:

```
Contents of data frame:
  fname  lname   level    dept       state
0  Jane  Smith  Senior   Sales  California
1  Dave  Smith  Senior   Devel  California
2  Jane  Jones   Year1   Mrktg    Illinois
3  Jane  Jones   Year1   Mrktg    Illinois
4  Jane  Stone  Senior   Mrktg     Arizona
5  Dave  Stone   Year2   Devel     Arizona
6  Mark  Aster   Year3  BizDev     Florida
7  Jane  Jones   Year1   Mrktg    Illinois
```

```
Sorted (ascending) by first name:
  fname  lname   level    dept       state
1  Dave  Smith  Senior   Devel  California
5  Dave  Stone   Year2   Devel     Arizona
0  Jane  Smith  Senior   Sales  California
2  Jane  Jones   Year1   Mrktg    Illinois
3  Jane  Jones   Year1   Mrktg    Illinois
4  Jane  Stone  Senior   Mrktg     Arizona
7  Jane  Jones   Year1   Mrktg    Illinois
6  Mark  Aster   Year3  BizDev     Florida

Sorted (descending) by first name:
  fname  lname   level    dept       state
6  Mark  Aster   Year3  BizDev     Florida
0  Jane  Smith  Senior   Sales  California
2  Jane  Jones   Year1   Mrktg    Illinois
3  Jane  Jones   Year1   Mrktg    Illinois
4  Jane  Stone  Senior   Mrktg     Arizona
7  Jane  Jones   Year1   Mrktg    Illinois
1  Dave  Smith  Senior   Devel  California
5  Dave  Stone   Year2   Devel     Arizona

Sorted (ascending) by first name and last name:
  fname  lname   level    dept       state
1  Dave  Smith  Senior   Devel  California
5  Dave  Stone   Year2   Devel     Arizona
2  Jane  Jones   Year1   Mrktg    Illinois
3  Jane  Jones   Year1   Mrktg    Illinois
7  Jane  Jones   Year1   Mrktg    Illinois
0  Jane  Smith  Senior   Sales  California
4  Jane  Stone  Senior   Mrktg     Arizona
6  Mark  Aster   Year3  BizDev     Florida
```

WORKING WITH GROUPBY() IN PANDAS

Listing 3.35 displays the contents of `groupby1.py` that illustrates how to invoke the Pandas `groupby()` method in order to compute subtotals of feature values.

LISTING 3.35: groupby1.py

```
import pandas as pd

# colors and weights of balls:
data = {'color':['red','blue','blue','red','blue'],
        'weight':[40,50,20,30,90]}
df1 = pd.Data frame(data)
print("df1:")
print(df1)
print()
print(df1.groupby('color').mean())
print()
```

```
red_filter = df1['color']=='red'
print(df1[red_filter])
print()
blue_filter = df1['color']=='blue'
print(df1[blue_filter])
print()

red_avg = df1[red_filter]['weight'].mean()
blue_avg = df1[blue_filter]['weight'].mean()
print("red_avg,blue_avg:")
print(red_avg,blue_avg)
print()

df2 = pd.Data frame({'color':['blue','red'],'weight':[red_
avg,blue_avg]})
print("df2:")
print(df2)
print()
```

Listing 3.35 defines the variable `data` containing `color` and `weight` values, and then initializes the data frame `df` with the contents of the variable `data`. The next two code blocks define `red_filter` and `blue_filter` that match the rows whose colors are red and blue, respectively, and then prints the matching rows.

The next portion of Listing 3.35 defines the two filters `red_avg` and `blue_avg` that calculate the average weight of the red value and the blue values, respectively. The last code block in Listing 3.35 defines the data frame `df2` with a `color` column and a `weight` column, where the latter contains the average weight of the red values and the blue values. Launch the code in Listing 3.35 and you will see the following output:

```
initial data frame:
df1:
  color  weight
0   red      40
1  blue      50
2  blue      20
3   red      30
4  blue      90

          weight
color
blue   53.333333
red    35.000000

  color  weight
0   red      40
3   red      30

  color  weight
1  blue      50
2  blue      20
4  blue      90
```

```
red_avg,blue_avg:
35.0 53.333333333333336

df2:
  color     weight
0  blue  35.000000
1   red  53.333333
```

AGGREGATE OPERATIONS WITH THE TITANIC.CSV DATASET

Listing 3.36 displays the contents of `aggregate2.py` that illustrates how to perform aggregate operations with columns in the `CSV` file `titanic.csv`.

LISTING 3.36: aggregate2.py

```
import pandas as pd

#Loading titanic.csv in Seaborn:
#df = sns.load_dataset('titanic')
df = pd.read_csv("titanic.csv")

# convert floating point values to integers:
df['survived'] = df['survived'].astype(int)

# specify column and aggregate functions:
aggregates1 = {'embark_town': ['count', 'nunique', 'size']}

# group by 'deck' value and apply aggregate functions:
result = df.groupby(['deck']).agg(aggregates1)
print("=> Grouped by deck:")
print(result)
print()

# some details regarding count() and nunique():
# count() excludes NaN values whereas size() includes them
# nunique() excludes NaN values in the unique counts

# group by 'age' value and apply aggregate functions:
result2 = df.groupby(['age']).agg(aggregates1)
print("=> Grouped by age (before):")
print(result2)
print()

# some "age" values are missing (so drop them):
df = df.dropna()

# convert floating point values to integers:
df['age'] = df['age'].astype(int)

# group by 'age' value and apply aggregate functions:
result3 = df.groupby(['age']).agg(aggregates1)
```

```
print("=> Grouped by age (after):")
print(result3)
print()
```

Listing 3.36 initializes the data frame `df` with the contents of the `CSV` file `titanic.csv`. The next code snippet converts floating point values to integer, followed by defining the variable `aggregates1` that specifies the functions `count()`, `nunique()`, and `size()` that will be invoked on the `embark_town` field.

The next code snippet initializes the variable `result` after invoking the `groupby()` method on the `deck` field, followed by invoking the `agg()` method.

The next code block performs the same computation to initialize the variable result2, except that the `groupby()` function is invoked on the `age` field instead of the `embark_town` field. Notice the comment section regarding the `count()` and `nunique()` functions: we will drop the rows with missing values via `df.dropna()` and investigate how that affects the calculations.

After dropping the rows with missing values, the final code block initializes the variable `result3` in exactly the same way that `result2` was initialized. Now launch the code in Listing 3.36 and the output is shown here:

```
=> Grouped by deck:

     embark_town
           count nunique size
deck
A             15       2   15
B             45       2   47
C             59       3   59
D             33       2   33
E             32       3   32
F             13       3   13
G              4       1    4

=> Grouped by age (before):
        age
      count nunique size
age
0.42      1       1    1
0.67      1       1    1
0.75      2       1    2
0.83      2       1    2
0.92      1       1    1
...     ...     ...  ...
70.00     2       1    2
70.50     1       1    1
71.00     2       1    2
74.00     1       1    1
80.00     1       1    1

[88 rows x 3 columns]
```

```
=> Grouped by age (after):
      age
    count nunique size
age
0       1       1    1
1       1       1    1
2       3       1    3
3       1       1    1
4       3       1    3
6       1       1    1
11      1       1    1
14      1       1    1
15      1       1    1
// details omitted for brevity
60      2       1    2
61      2       1    2
62      1       1    1
63      1       1    1
64      1       1    1
65      2       1    2
70      1       1    1
71      1       1    1
80      1       1    1
```

WORKING WITH APPLY() AND MAPAPPLY() IN PANDAS

Earlier in this chapter you saw an example of the `Pandas` `apply()` method for modifying the categorical values of a feature in the `CSV` file `shirts.csv`. This section contains more examples of the `apply()` method, along with examples of the `mapappy()` method.

Listing 3.37 displays the contents of `apply1.py` that illustrates how to invoke the `Pandas` `apply()` method in order to compute the sum of a set of values.

LISTING 3.37: apply1.py

```
import pandas as pd

df = pd.Data frame({'X1': [1,2,3], 'X2': [10,20,30]})

def cube(x):
  return x * x * x

df1 = df.apply(cube)
# same result:
# df1 = df.apply(lambda x: x * x * x)

print("initial data frame:")
print(df)
print("cubed values:")
print(df1)
```

Listing 3.37 initializes the data frame `df` with columns `X1` and `X2`, where the values for `X2` are 10 times the corresponding values in `X1`. Next, the `Python` function `cube()` returns the cube of its argument. Listing 3.36 then defines the variable `df1` by invoking the `apply()` function, which specifies the user-defined `Python` function `cube()`, and then prints the values of `df` as well as `df1`. Launch the code in Listing 3.37 and you will see the following output:

```
initial data frame:
   X1  X2
0   1  10
1   2  20
2   3  30
cubed values:
   X1     X2
0   1   1000
1   8   8000
2  27  27000
```

Listing 3.38 displays the contents of `apply2.py` that illustrates how to invoke the `Pandas apply()` method in order to compute the sum of a set of values.

LISTING 3.38: apply2.py

```
import pandas as pd
import numpy as np

df = pd.Data frame({'X1': [10,20,30], 'X2': [50,60,70]})

df1 = df.apply(np.sum, axis=0)
df2 = df.apply(np.sum, axis=1)

print("initial data frame:")
print(df)
print("add values (axis=0):")
print(df1)
print("add values (axis=1):")
print(df2)
```

Listing 3.38 is a variation of Listing 3.37: the variables `df1` and `df2` contain the column-wise sum and the row-wise sum, respectively, of the data frame `df`. Launch the code in Listing 3.38 and you will see the following output:

```
   X1  X2
0  10  50
1  20  60
2  30  70
add values (axis=0):
X1     60
X2    180
dtype: int64
add values (axis=1):
0     60
```

```
1     80
2    100
dtype: int64
```

Listing 3.39 displays the contents of `mapapply1.py` that illustrates how to invoke the `Pandas mapapply()` method in order to compute the sum of a set of values.

LISTING 3.39: mapapply1.py

```
import pandas as pd
import math

df = pd.Data frame({'X1': [1,2,3], 'X2': [10,20,30]})
df1 = df.applymap(math.sqrt)

print("initial data frame:")
print(df)
print("square root values:")
print(df1)
```

Listing 3.39 is yet another variant of Listing 3.37: in this case, the variable `df1` is defined by invoking the `applymap()` function on the variable `df`, which in turn references (but does not execute) the `math.sqrt()` function. Next, a print statement displays the contents of `df`, followed by a `print()` statement that displays the contents of `df1`: it is at this point that the built-in `math.sqrt()` function is invoked in order to calculate the square root of the values in `df`. Launch the code in Listing 3.39 and you will see the following output:

```
initial data frame:
   X1  X2
0   1  10
1   2  20
2   3  30

square root values:
         X1        X2
0  1.000000  3.162278
1  1.414214  4.472136
2  1.732051  5.477226
```

Listing 3.40 displays the contents of `mapapply2.py` that illustrates how to invoke the `Pandas mapapply()` method in order to convert strings to lowercase and uppercase.

LISTING 3.40: mapapply2.py

```
import pandas as pd

df = pd.Data frame({'fname': ['Jane'], 'lname': ['Smith']},
                  {'fname': ['Dave'], 'lname': ['Jones']})
```

```
df1 = df.applymap(str.lower)
df2 = df.applymap(str.upper)

print("initial data frame:")
print(df)
print()
print("lowercase:")
print(df1)
print()
print("uppercase:")
print(df2)
print()
```

Listing 3.40 initializes the variable `df` with two first and last name pairs, and then defines the variables `df1` and `df2` by invoking the `applymap()` method to the variable `df`. The variable `df1` converts its input values to lowercase, whereas the variable `df2` converts its input values to uppercase. Launch the code in Listing 3.40 and you will see the following output:

```
initial data frame:
      fname  lname
fname  Jane  Smith
lname  Jane  Smith

lowercase:
      fname  lname
fname  jane  smith
lname  jane  smith

uppercase:
      fname  lname
fname  JANE  SMITH
lname  JANE  SMITH
```

USEFUL ONE-LINE COMMANDS IN PANDAS

This section contains an eclectic mix of one-line commands in `Pandas` (some of which you have already seen in this chapter) that are useful to know:

List the column names of a `Data frame`:

```
df.columns
```

Drop missing data from a `Data frame`:

```
df.dropna(axis=0, how='any')
```

Remove an unnecessary column:

```
my_dataset = my_dataset.drop(["url"],axis=1)
```

Remove columns with a single value, or columns that are missing more than 50% of their values:

```
dataset = dataset.dropna(thresh=half_count,axis=1)
```

Replace missing data in a `Data frame`:

```
df.replace(to_replace=None, value=None)
```

Check for NANs in a `Data frame`:

```
pd.isnull(object)
```

Drop a feature in a `Data frame`:

```
df.drop('feature_variable_name', axis=1)
```

Convert object type to float in a `Data frame`:

```
pd.to_numeric(df["feature_name"], errors='coerce')
```

Convert data in a `Data frame` to `NumPy` array:

```
df.as_matrix()
```

Display the first n rows of a data frame:

```
df.head(n)
```

Get data by feature name in a `Data frame`:

```
df.loc[feature_name]
```

Apply a function to a data frame, such as multiplying all values in the "height" column of the data frame by 3:

```
df["height"].apply(lambda height: 3 * height)
```

OR:

```
def multiply(x):
    return x * 3
df["height"].apply(multiply)
```

Rename the fourth column of the data frame as "height":

```
df.rename(columns = {df.columns[3]:'height'}, inplace=True)
```

Get the unique entries of the column "first" in a data frame:

```
df["first"].unique()
```

Create a data frame with columns `first` and `last` from an existing `Data frame`:

```
new_df = df[["first", "last"]]
```

Sort the data in a Data frame:

```
df.sort_values(ascending = False)
```

Filter the data column named "size" to display only values equal to 7:

```
df[df["size"] == 7]
```

Select the first row of the "height" column in a Data frame:

```
df.loc([0], ['height'])
```

WORKING WITH JSON-BASED DATA

A JSON object consists of data represented as colon-separated name/value pairs and data objects are separated by commas. An object is specified inside curly braces {}, and an array of objects is indicated by square brackets []. Note that character-valued data elements are inside a pair of double quotes "" (but no quotes are used for numeric data).

Here is a simple example of a JSON object:

```
{ "fname":"Jane", "lname":"Smith", "age":33, "city":"SF" }
```

Here is a simple example of an array of JSON objects (note the outer enclosing square brackets):

```
[
{ "fname":"Jane", "lname":"Smith", "age":33, "city":"SF" },
{ "fname":"John", "lname":"Jones", "age":34, "city":"LA" },
{ "fname":"Dave", "lname":"Stone", "age":35, "city":"NY" },
]
```

Python Dictionary and JSON

The Python json library enables you to work with JSON-based data in Python.

Listing 3.41 displays the contents of dict2json.py that illustrates how to convert a Python dictionary to a JSON string.

LISTING 3.41: dict2json.py

```
import json

dict1 = {}
dict1["fname"] = "Jane"
dict1["lname"] = "Smith"
dict1["age"]   = 33
dict1["city"]  = "SF"

print("Python dictionary to JSON data:")
print("dict1:",dict1)
json1 = json.dumps(dict1, ensure_ascii=False)
```

```
print("json1:",json1)
print("")

# convert JSON string to Python dictionary:
json2 = '{"fname":"Dave", "lname":"Stone", "age":35, "city":"NY"}'
dict2 = json.loads(json2)
print("JSON data to Python dictionary:")
print("json2:",json2)
print("dict2:",dict2)
```

Listing 3.41 invokes the `json.dumps()` function to perform the conversion from a `Python` dictionary to a `JSON` string. Launch the code in Listing 3.41 and you will see the following output:

```
Python dictionary to JSON data:
dict1: {'fname': 'Jane', 'lname': 'Smith', 'age': 33, 'city': 'SF'}
json1: {"fname": "Jane", "lname": "Smith", "age": 33, "city": "SF"}

JSON data to Python dictionary:
json2: {"fname":"Dave", "lname":"Stone", "age":35, "city":"NY"}
dict2: {'fname': 'Dave', 'lname': 'Stone', 'age': 35, 'city': 'NY'}
```

Python, Pandas, and JSON

Listing 3.42 displays the contents of `pd_python_json.py` that illustrates how to convert a `Python` dictionary to a `Pandas DataFrame` and then convert the data frame to a `JSON` string.

LISTING 3.42: pd_python_json.py

```
import json
import pandas as pd

dict1 = {}
dict1["fname"] = "Jane"
dict1["lname"] = "Smith"
dict1["age"]   = 33
dict1["city"]  = "SF"

df1 = pd.Data frame.from_dict(dict1, orient='index')
print("Pandas df1:")
print(df1)
print()

json1 = json.dumps(dict1, ensure_ascii=False)
print("Serialized to JSON1:")
print(json1)
print()

print("Data frame to JSON2:")
json2 = df1.to_json(orient='split')
print(json2)
```

Listing 3.42 initializes a `Python` dictionary `dict1` with multiple attributes for a user (first name, last name, and so forth). Next, the data frame `df1` is created from the `Python` dictionary `dict1`, and its contents are displayed.

The next portion of Listing 3.42 initializes the variable `json1` by serializing the contents of `dict1`, and its contents are displayed. The last code block in Listing 3.42 initializes the variable `json2` to the result of converting the data frame `df1` to a `JSON` string. Launch the code in Listing 3.42 and you will see the following output:

```
dict1: {'fname': 'Jane', 'lname': 'Smith', 'age': 33, 'city': 'SF'}
Pandas df1:
           0
fname   Jane
lname  Smith
age       33
city      SF

Serialized to JSON1:
{"fname": "Jane", "lname": "Smith", "age": 33, "city": "SF"}

Data frame to JSON2:
{"columns":[0],"index":["fname","lname","age","city"],"data":[["Jane"],
["Smith"],[33],["SF"]]}
json1: {"fname": "Jane", "lname": "Smith", "age": 33, "city": "SF"}
```

SUMMARY

This chapter introduced you to `Pandas` for creating labeled `Data frames` and displaying metadata of `Pandas DataFrames`. Then you learned how to create `Pandas DataFrames` from various sources of data, such as random numbers and hard-coded data values.

You also learned how to read `Excel` spreadsheets and perform numeric calculations on that data, such as the minimum, mean, and maximum values in numeric columns. Then you saw how to create `Pandas DataFrames` from data stored in `CSV` files. In addition, you learned how to generate a scatterplot from data in a `Pandas DataFrame`.

Finally, you got a brief introduction to `JSON`, along with an example of converting a `Python` dictionary to `JSON`-based data (and vice versa).

CHAPTER 4

INTRODUCTION TO RDBMS AND SQL

This chapter introduces you to RDBMSes, various SQL concepts, and a quick introduction to MySQL. In case you are wondering, MySQL is a robust RDBMS and it is available as a free download from an ORACLE website. Moreover, virtually everything that you learn about MySQL in this chapter transfer to other RDBMSes, such as PostgreSQL and ORACLE.

This chapter describes a hypothetical website that enables users to register themselves for the purpose of purchasing various tools (hammers, wrenches, and so forth). Although there is no code in this section, you will learn about the tables that are required, their relationships, and the structure of those tables.

WHAT IS AN RDBMS?

RDBMS is an acronym for relational database management system. RDBMSes store data in tables that contain labeled *attributes* (informally sometimes called *columns*) that have a specific data type. Examples of an RDBMS include MySQL, ORACLE, and IBM DB2. Although relational databases usually provide a decent solution for storing data, speed and scalability might be an issue in some cases, NoSQL databases (such as MongoDB) might be more suitable for scalability.

What Relationships Do Tables Have in an RDBMS?

While an RDBMS can consist of a single table, it often comprises multiple tables that can have various types of associations with each other. For example, when you buy various items at a food store, your receipt consists of one purchase order that contains one or more "line items," where each line item indicates the details of a particular item that you purchased. This is called a *one-to-many* relationship between a purchase order (which is stored in a purchase_orders table) and the line items (stored in a line_items table) for each item that you purchased.

Another example involves students and courses: each student is enrolled in one or more courses, which is a one-to-many relationship from students to courses. Moreover, each course contains one or more students, so there is a one-to-many relationship from courses to students. Therefore, the students and courses tables have a many-to-many relationship.

A third example is an employees table, where each row contains information about one employee. If each row includes the `id` of the manager of the given employee, then the employees table is a *self-referential* table because finding the manager of the employee involves searching the `employees` table with the manager's `id` that is stored in the given employee record. However, if the rows in an `employees` table do *not* contain information about an employee's manager, then the table is not self-referential.

In addition to table definitions, a database frequently contains indexes, primary keys, and foreign keys that facilitate searching for data in tables and also "connecting" a row in a given table with its logically related row (or rows) in another table. For example, if we have the id value for a particular purchase order in a `purchase_orders` table, we can find all the line items (i.e., the items that were purchased) in a `line_items` table that contain the same purchase order `id`.

Features of an RDBMS

An `RDBMS` provides a convenient way to store data, often associated with some type of application. For example, later you will see the details of a four-table `RDBMS` that keeps track of tools that are purchased via a Web-based application. From a high-level perspective, an `RDBMS` provides the following characteristics:

- a database contains one or more tables
- data is stored in tables
- data records have the same structure
- well-suited for vertical scaling
- support for ACID (explained in the following section)

Another useful concept is a *logical schema* that consists of the collection of tables and their relationships (along with indexes, views, triggers, and so forth) in an `RDBMS`. The schema is used for generating a *physical schema*, which consists of all the `SQL` statements that are required in order to create the specified tables and their relationships.

What Is ACID?

`ACID` is an acronym for atomicity, consistency, isolation, and durability, which refers to properties of RDBMS transactions.

Atomicity means that each transaction is all-or-nothing, so if a transaction fails, the system is rolled back to its previous state.

Consistency means that successful transactions always result in a valid database state.

Isolation means that executing transactions concurrently or serially will result in the state.

Durability means that a committed transaction will remain in the same state.

Keep in mind that RDBMSes support ACID, whereas NoSQL databases generally do not support ACID.

WHEN DO WE NEED AN RDBMS?

The short answer is that an RDBMS is useful when we need to store one or more records of events that have occurred, which can be involve simple item purchases as well as complex multi-table financial transactions.

An RDBMS allows you to define a collection of tables that contain rows of data, where a row contains one or more attributes (informally called "fields"). A row of data is a record of an event that occurred at a specific point in time, which can involve more than one table, and can also involve some type of "transaction."

For example, consider a database that contains a single table called an events table in which a single row contains information about a single event that you created by some process (such as a Web page registration form). Although this is conceptually simple, notice that the following attributes are relevant for each row in the events table: event_id, event_time, event_title, event_duration, and event_location, and possibly additional attributes.

Now consider a money transfer scenario between two bank accounts: you need to transfer USD 100 from a savings account to a checking account. The process involves two steps:

1. debiting (subtracting) the savings account by USD 100, and
2. crediting (adding) the savings account with USD 100.

However, if there is a system failure after step 1 and before step 2 can be completed, you have lost USD 100. Obviously steps 1 and 2 must be treated as an *atomic transaction*, which means that the transaction is successful only when both steps have completed successfully. If the transaction is unsuccessful, the transaction is "rolled back" so the system is returned to the state prior to transferring money between the two accounts.

As you learned earlier in this chapter, RDBMSes support ACID, which ensures that the previous transaction (i.e., transferring money between accounts) is treated as an atomic transaction.

Although atomic transactions are fundamental to financial systems, they might not be as critical for other systems. For example, the previous example involved inserting a new row in an events table whenever a new event is created. If this process fails, the solution might involve registering the event again when the system is online again (perhaps the database crashed).

As another example, displaying a set of pictures in a Web page might not display the pictures in the correct order (e.g., based on their creation time). However, a failure in the event creation is not as critical as a failure in a financial system, and displaying images in the wrong sequence will probably be rectified when the Web page is refreshed.

THE IMPORTANCE OF NORMALIZATION

This section contains a gentle introduction to the concept of *normalization*, ideally providing you with the intuition that underlies normal forms. A complete explanation is beyond the scope of this book, but you can find detailed information in online articles.

As a starting point, consider an RDBMS that stores records for the temperature of a room during a time interval (such as a day, a week, or some other time interval). We just need one `device_temperature` table where each row contains the temperature of a room at a specific time. In the case of IoT (internet of things), the temperature is recorded during regular time intervals (such as minute-by-minute or hourly).

If you need to track only one room, the device_temperature table is probably sufficient. However, if you need to track *multiple* devices in a room, then it is convenient to create a second table called device_details that contains attributes for each device, such as `device_id`, `device_name`, `device_year`, `device_price`, `device_warranty`, and so forth.

However, we need to connect information from a row in the table device_temperature to its associated row in the `device_details` table. The two-table connection is simple: each row in the `device_details` table contains a `device_id` that uniquely identifies the given row. Moreover, the same `device_id` appears in any row of the `device_temperature` table that refers to the given device.

The preceding two-table structure is a minimalistic example of something called database *normalization*, which reduces data redundancy in database tables, sometimes at the expense of slower performance during the execution of some types of `SQL` statements (e.g., those that contain a JOIN keyword).

If you are new to the concept of database normalization, you might be thinking that normalization increases complexity and reduces performance without providing tangible benefits. While this is a valid question, the trade-off is worthwhile.

In order to convince you of the value of normalization, suppose that every record in the `purchase_orders` table contains all the details of the customer who made the associated purchase. As a result, we can eliminate the `customers` table. However, if we ever need to update the address of a particular customer, we need to update all the rows in the `purchase_orders` table that contain that customer. By contrast, if we maintain a `customers` table then updating a customer's address involves changing a *single* row in the `customers` table.

Normalization enables us to avoid data duplication so that there is a single "source of truth" in the event that information (such as a customer's address) must be updated. From another perspective, data duplication means that the same data appears in two (or possibly more) locations, and if an update is not applied to all those locations, the database data is in an inconsistent state. Depending on the nature of the application, the consequences of inconsistent data can range from minor to catastrophic.

Always remember the following point: whenever you need to update the same data that resides in two different locations, you increase the risk of data inconsistency that adversely affects data integrity.

As another example, suppose that a website sells widgets online: at a minimum, the associated database needs the following four tables:

- `customer_details`
- `purchase_orders`
- `po_line_items`
- `item_desc`

The preceding scenario is explored in greater detail in the next section that specifies the attributes of each of the preceding tables.

A FOUR-TABLE RDBMS

As an introductory example, suppose that *www.mytools.com* sells tools for home use or construction (the details of which are not important). For simplicity, we will pretend that an actual Web page is available at the preceding `URL` and the Web page contains the following sections:

- new user register registration
- existing user log in
- input fields for selecting items for purchase (and the quantities)

For example, the registered user `John Smith` wants to purchase one hammer, two screwdrivers, and three wrenches. The Web page needs to provide users with the ability to search for products by their type (e.g., hammer, screwdriver, wrench, and so forth) and then display a list of matching products. Each product in that list would also contain an SKU, which is an industry-standard labeling mechanism for products (just like ISBNs for identifying books).

The preceding functionality is necessary in order to develop a Web page that enables users to purchase products. However, the purpose of this section is to describe a set of tables (and their relationships to each other) in an `RDBMS`, so we will assume that the necessary Web-based features are available at our `URL`.

We will describe a so-called "use case" that contains the sequence of steps that will be performed on behalf of an existing customer `John Smith` (whose customer id is 1000), who wants to purchase 1 hammer, 2 screwdrivers, and 3 wrenches:

Step 1: Customer John Smith (with cust_id 1000) initiates a new purchase.
Step 2: A new purchase order is created with the value 12500 for po_id.
Step 3: John Smith selects 1 hammer, 2 screwdrivers, and 3 wrenches.
Step 4: The associated US price of \$20.00, \$16.00, and \$30.00 is displayed on the screen.
Step 5: The subtotal is displayed, which is \$66.00.
Step 6: The tax of \$6.60 is displayed (a tax rate of 10%).
Step 7: The total cost of \$72.60 is displayed.

Step 8 would allow John Smith to remove an item, increase/decrease the quantity for each selected item, delete items, or cancel the purchase order. Step 9 would enable `John Smith` to make a payment. Once again, for the sake of simplicity, we will assume that step 8 and step 9 are available.

Note that step 8 involves updating several of our tables with the details of the purchase order. Step 9 creates a time stamp for the date when the purchase order was created, as well as the status of the purchase order ("paid" versus "pending"). The status of a purchase order is used to generate reports to display the customers whose payment is overdue (and perhaps also send them friendly reminders). Sometimes companies have a reward-based system whereby customers who have paid on time can collect credits that can be applied to other purchases (in other words, it's essentially a discount mechanism).

DETAILED TABLE DESCRIPTIONS

If you visualize the use case described in the previous section, you can probably see that we need a table for storing customer-specific information, another table to store purchase orders (which is somehow linked to the associated customer), a table that contains the details of the items and quantity that are purchased (which are commonly called "line items"), and a table that contains information about each tool (which includes the name, the description, and the price of the tool). Therefore, the RDBMS for our website requires the following tables:

```
customers
purchase_orders
line_items
item_desc
```

The following subsections describe the contents of the preceding tables, along with the relationships among these tables.

The customers Table

Although there are different ways to specify the attributes of the `customers` table, you need enough information to uniquely identify each customer in the table. By analogy, the following information (except for `cust_id`) is required in order to mail an envelope to a person:

```
cust_id
first_name
last_name
home_address
city
state
zip_code
```

We will create the customers table with the attributes in the preceding list. Note that the `cust_id` attribute is called a *key* because it uniquely identifies every customer. Although we'll defer the discussion of keys to a later chapter, it is obvious that we need a mechanism for uniquely identifying every customer.

Whenever we need to refer to the details of a particular customer, we will use the associated value of `cust_id` to retrieve those details from the row in the customers table that has the associated `cust_id`.

The preceding paragraph describes the essence of linking related tables `T1` and `T2` in an RDBMS: the key in `T1` is stored as an attribute value in `T2`. If we need to access related information in table `T3`, then we store the key in `T2` as an attribute value in `T3`.

Note that a `customers` table in a production system would contain other attributes, such as the following:

```
title (Mr, Mrs, Ms, and so forth)
shipping_address
cell_phone
```

For the sake of simplicity, we'll use the initial set of attributes to define the `customers` table: later on you can add the new attributes to the table schema to make the system more like a real life system.

In order to make this table more concrete, suppose that the following information pertains to customer `John Smith`, who has been assigned a `cust_id` of 1000:

```
cust_id: 1000
first_name: John
last_name: Smith
home_address: 1000 Appian Way
city: Sunnyvale
state: California
zip_code:95959
```

Whenever `John Smith` makes a new purchase, we will use the `cust_id` value of 1000 to create a new row for this customer in the `purchase_order` table.

The purchase_orders Table

When customers visit the website, we need to create a purchase order that will be inserted as a new row in the `purchase_orders` table. While you might be tempted to place all the customers' details in the new row, we will identify the customer by the associated `cust_id` and use this value instead.

Note that we create a new row in the `customers` table whenever new users register at the website, whereas repeat customers are identified by an existing `cust_id` that must be determined by searching the customers table with the information that the customer types into the input fields of the main Web page.

We saw that the `customers` table contains a key attribute; similarly, the `purchase_orders` table requires an attribute that we will call `po_id` (you are free to use a different string) in order to identify a purchase order for a given customer.

Keep in mind the following detail: the row with a given `po_id` requires a `cust_id` attribute in order to also identify the customer (in the `customers` table) who is making the current purchase.

Although there are multiple ways to define a set of suitable attributes, we will use the following set of attributes for the `purchase_orders` table:

```
po_id
cust_id
purchase_date
```

For example, suppose that customer `John Smith`, whose `cust_id` is 1000, purchases some tools on December 15, 2021. There are dozens of different date formats that are supported in RDBMSes: for simplicity, we will use the MM-DD-YYYY format (which you can change to suit your particular needs).

Then the new row for John Smith in the `purchase_orders` would look something like the following:

```
po_id: 12500
cust_id: 1000
purchase_date: 12-01-2021
```

The line_items Table

As a concrete example, suppose that customer `John  Smith` requested 1 hammer, 2 screwdrivers, and 3 wrenches in his most recent purchase order. Each of these purchased items requires a row in the `line_items` table that:

- is identified by a line_id value
- specifies the quantity of each purchased item
- contains the value for the associated `po_id` in the purchase_orders table
- contains the value for the associated item_id in the item_desc table

For simplicity, we will assign the values 5001, 5002, and 5003 to the line_id attribute for the three new rows in the `line_items` table that represent the hammer, screwdriver, and wrench items in the current purchase order. A `line_item` row might look something like this:

```
po_id: 12500
line_id: 5001
item_id: 100 <= we'll discuss this soon
```

```
item_count: 1
item_price: 20.00
item_tax:  2.00
item_subtotal: 22.00
```

Notice there is no `cust_id` in the preceding line_item: that's because of the top-down approach for retrieving data. Specifically, we start with a particular `cust_id` that we use to find a list of purchase orders in the `purchase_orders` table that belong to the given `cust_id`, and for each purchase order in the `purchase_orders` table, we perform a search for the associated line items in the `line_items` table. Moreover, we can repeat the preceding sequence of steps for each customer in a list of `cust_id` values.

Returning to the earlier line_item details: we need to reference each purchased item by its associated identifier in the `item_desc` table. Once again, we will arbitrarily assign `item_id` values of 100, 200, and 300, respectively, for the hammer, screwdriver, and wrench items. The actual values will undoubtedly be different in your application, so there is no special significance to the numbers 100, 200, and 300.

The three rows in the `line_items` table (that belong to the same purchase order) would look like this (we'll look at the corresponding `SQL` statements later):

```
po_id: 12500
line_id: 5001
item_id: 100
item_count: 1
item_price: 20.00
item_tax:  2.00
item_subtotal: 22.00

po_id: 12500
line_id: 5002
item_id: 200
item_count: 2
item_price: 8.00
item_tax:   1.60
item_subtotal: 17.60

po_id: 12500
line_id: 5003
item_id: 300
item_count: 3
item_price: 10.00
item_tax:    3.00
item_subtotal: 33.00
```

The item_desc Table

Recall that the `customers` table contains information about each customer, and a new row is created each time that a new customer creates an account for our Web application. In a somewhat analogous fashion, the `item_desc` table contains information about each item (aka product) that can be purchased from our website. If our website becomes popular, the contents of the item_desc table contents are updated more frequently than the customers table, typically in the following situations:

- A new tool (aka product) is available for purchase.
- An existing tool is no longer available for purchase.

Thus, the item_desc table contains all the details for every tool that is available for sale, and it is the "source of truth" for the tools that customers can purchase from the website. At a minimum, this table contains three fields, as shown here (later we'll discuss the SQL statement for creating and populating this table):

```
SELECT *
FROM item_desc;
+---------+-------------+------------+
| item_id | item_desc   | item_price |
+---------+-------------+------------+
|     100 | hammer      |      20.00 |
|     200 | screwdriver |       8.00 |
|     300 | wrench      |      10.00 |
+---------+-------------+------------+
3 rows in set (0.001 sec)
```

There is one more important detail to discuss: if an item is no longer for sale, can we simply drop its row from the item_desc table? The answer is "no" because we need this row in order to generate reports that contain information about the items that customers purchased.

Therefore, it would be a good idea to add another attribute called AVAILABLE (or something similar) that contains either 1 or 0 to indicate whether or not the product is available for purchase. As a result, some of the SQL queries that involve this table will also need to take into account this new attribute. Implementation of this functionality is not central to the purpose of this book, and therefore it is left as an enhancement to the reader.

WHAT IS SQL?

SQL is an acronym for structured query language, which is used for managing data in tables in a relational database (RDBMS). In fact, SQL is a standard language for retrieving and manipulating structured databases.

In high-level terms, a SQL statement to retrieve data generally involves the following:

- choosing desired data (SELECT)
- the table(s) where the data resides (FROM)
- constraints (if any) on the data (WHERE)

For example, suppose that a friends table contains the attributes (database parlance for "fields") lname and fname for the last name and first name, respectively, of a set of friends, and each row in this table contains details about one friend.

In Chapter 2, we'll learn how to create database tables and how to populate those tables with data, but for now we will just pretend that those tasks have already been performed. Then the SQL statement for retrieving the first and last names of the people in the friends table looks like this:

```
SELECT lname, fname
FROM friends;
```

Suppose that the friends table also contains a height attribute, which is a number (in centimeters) for each person in the friends table. We can extend the preceding SQL statement to specify that we want the people (rows) whose height attribute is less than 180 as follows:

```
SELECT lname, fname
FROM friends
WHERE height < 180;
```

As you will see, SQL provides a plethora of keywords that enable you to specify sophisticated queries for retrieving data from multiple tables. Both of the preceding SQL statements are called DML statements, which is one of the four main categories of SQL statements:

- DCL (data control language
- DDL (data definition language)
- DQL (data query language)
- DML (data manipulation language)

The following subsections provide additional information for each item in the preceding list.

DCL, DDL, DQL, DML, and TCL

DCL is an acronym for data control language, which refers to any SQL statement that contains the keywords GRANT or REVOKE. Both of the keywords affect the permissions that are either granted or revoked for a particular user.

DDL is an acronym for data definition language, which refers to any SQL statements that specify: CREATE, ALTER, DROP, RENAME, TRUNCATE, or COMMENT. These SQL keywords are used in conjunction with database tables and in many cases with database views (discussed later).

DQL is an acronym for data query language, which refers to any SQL statement that contains the keyword SELECT.

DML is an acronym for data manipulation language, which refers to SQL statements that execute queries against one or more tables in a database.

Specifically, the SQL statements can contain any of the keywords INSERT, UPDATE, DELETE, MERGE, CALL, EXPLAIN PLAN, or LOCK TABLE. In most cases these keywords modify the existing values of data in one or more tables.

TCL is an acronym for transaction control language, which refers to any of the keywords COMMIT, ROLLBACK, SAVEPOINT, or SET TRANSACTION.

SQL Privileges

There are two types of privileges available in SQL, both of which are described briefly in this section. These privileges refer to database objects such as database tables and indexes that are discussed in greater detail in subsequent chapters.

System privileges involve an object of a particular type and specifies the right to perform one or more actions on the object. Such actions include the administrator giving users permission to perform tasks such as ALTER INDEX, ALTER CACHE GROUP, CREATE/ALTER/DELETE TABLE, or CREATE/ALTER/DELETE VIEW.

Object privileges allow users to perform actions on an object or object of another user, such as tables, views, indexes, and so forth. Additional object privileges are EXECUTE, INSERT, UPDATE, DELETE, SELECT, FLUSH, LOAD, INDEX, and REFERENCES.

PROPERTIES OF SQL STATEMENTS

SQL statements and SQL functions are not case sensitive, but quoted text *is* case sensitive. Here are some examples:

```
select VERSION();
+-----------+
| VERSION() |
+-----------+
| 8.0.21    |
+-----------+
1 row in set (0.000 sec)

MySQL [mytools]> SeLeCt Version();
+-----------+
| Version() |
+-----------+
| 8.0.21    |
+-----------+
1 row in set (0.000 sec)
```

Also keep in mind the following useful details regarding SQL statements:

- SQL statements are not case sensitive.
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indentation is for enhancing readability.

The CREATE Keyword

In general, you will use the `CREATE` keyword sometimes to create a database and more often to create tables, views, and indexes. However, the following list contains all the objects that you can create via the `CREATE` statement:

- `DATABASE`
- `EVENT`
- `FUNCTION`
- `INDEX`
- `PROCEDURE`
- `TABLE`
- `TRIGGER`
- `USER`
- `VIEW`

Some of the keywords in the preceding list are discussed in this chapter as well as the next chapter of this book.

WHAT IS MYSQL?

`MySQL` is open source database that is portable and provides many features that are available in commercial databases. Oracle is the steward of the `MySQL` database, which you can download here:

https://www.mysql.com/downloads/

If you prefer, `MySQL` also provides a GUI interface for performing database-related operations. `MySQL` 8 provides the following new features:

- a transactional data dictionary
- Improved support for BLOB, TEXT, GEOMETRY, and JSON data types

As you will see in Chapter 7, `MySQL` supports pluggable storage engines, such as `InnoDB` (the most commonly used `MySQL` storage engine). In addition, Facebook developed an open source storage engine called `MyRocks` that has better compression and performance, so it might be worthwhile to explore the advantage of `MyRocks` over the other storage engines for `MySQL`.

What About MariaDB?

`MySQL` began as an open source project, and retained its name after the Oracle acquisition. Shortly thereafter, the MariaDB database was created, which is a "fork" of the `MySQL` database. Although `MariaDB` supports all the features of `MySQL`, there are important differences between `MySQL` and `MariaDB` that you can read about here:

https://mariadb.com/kb/en/mariadb-vs-mysql-compatibility/

Installing MySQL

Download the `MySQL` distribution for your machine and perform the installation procedure.

https://towardsdatascience.com/pandas-and-sql-together-a-premier-league-and-player-scouting-example-b41713a5dd3e

You can log into MySQL as root with the following command, which will prompt you for the root password:

```
$ mysql -u root -p
```

If you installed `MySQL` via a `DMG` file, then the root password is the same as the password for your machine.

DATA TYPES IN MYSQL

This section start with a lengthy list of data types that `MySQL` supports, followed by some comments about several of the data types, all of which you can use in table definitions:

- The BIT datatype is for storing bit values in MySQL.
- The BOOLEAN datatype stores True/False values.
- The CHAR data type is for storing fixed length strings.
- The DATE datatype is for storing date values.
- The DATETIME datatype is for storing combined date and time values.
- The DECIMAL datatype is for storing exact values in decimal format.
- The ENUM datatype is a compact way to store string values.
- The INT datatype is for storing an integer data type.
- The JSON data type is for storing JSON documents.
- The TEXT datatype is for storing text values.
- The TIME datatype is for storing time values.
- The TIMESTAMP datatype is for a wider range of date and time values.
- The TO_SECONDS datatype is for converting time to seconds.
- The VARCHAR datatype is for variable length strings.
- The XML data type provides support for XML documents.

The CHAR and VARCHAR Data Types

The `CHAR` type has a fixed column length, declared while creating tables, whose length can range from 1 to 255. `CHAR` values are right padded with spaces to the specified length, and trailing spaces are removed when `CHAR` values are retrieved.

By contrast, the `VARCHAR` type indicates variable length `CHAR` values whose length can be between 1 and 2000, and it occupies the space for `NULL` values.

By contrast, the `VARCHAR2` type indicates variable length `CHAR` values whose length can be between 1 and 4000, but can not occupy the space for `NULL` values. Therefore, `VARCHAR2` has better performance that `VARCHAR`.

String-Based Data Types

The previous bullet list contains various string types that have been extracted and placed in a separate list below for your convenience:

- BLOB
- CHAR
- ENUM
- SET
- TEXT
- VARCHAR

The ENUM datatype is string object that specifies a set of predefined values, which can be used during table creation, as shown here:

```
CREATE TABLE PIZZA(name ENUM('Small', 'Medium','Large'));
Query OK, 0 rows affected (0.021 sec)

DESC pizza;
+-------+--------------------------------+------+-----+---------+-------+
| Field | Type                           | Null | Key | Default | Extra |
+-------+--------------------------------+------+-----+---------+-------+
| name  | enum('Small','Medium','Large') | YES  |     | NULL    |       |
+-------+--------------------------------+------+-----+---------+-------+
1 row in set (0.004 sec)
```

FLOAT and DOUBLE Data Types

Numbers in the FLOAT format are stored in four bytes and have eight decimal places of accuracy. Numbers in the DOUBLE format are stored in eight bytes and have eighteen decimal places of accuracy.

BLOB and TEXT Data Types

A BLOB is an acronym for binary large object that can hold a variable amount of data. There are four BLOB types whose only difference is their maximum length:

- TINYBLOB
- BLOB
- MEDIUMBLOB
- LONGBLOB

A TEXT data type is a case-insensitive BLOB, and there are four TEXT types whose difference pertains to their maximum length (all of which are nonstandard data types):

- TINYTEXT
- TEXT
- MEDIUMTEXT
- LONGTEXT

Keep in mind the following difference between BLOB types and TEXT types: BLOB types involve case-sensitive sorting and comparisons, whereas these operations are case-insensitive for TEXT types.

MYSQL DATABASE OPERATIONS

There are several operations that you can perform with a MySQL database, as shown here:

- Create a database.
- Export a database.
- Drop a database.
- Rename a database.

You will see examples of how to perform each of the preceding bullet items in the following subsections.

Creating a Database

Log into MySQL and invoke the following command to create the mytools database:

```
MySQL [mysql]> create database mytools;
Query OK, 1 row affected (0.004 sec)
```

Now select the mytools database with the following command:

```
MySQL [(none)]> use mytools;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
```

Display a List of Databases

Display the existing databases by invoking the following SQL statement:

```
mysql> SHOW DATABASES;
```

The preceding command displays the following output (the output will be different for your machine):

```
+--------------------+
| Database           |
+--------------------+
| beans              |
| information_schema |
| minimal            |
| mysql              |
| mytools            |
| performance_schema |
| sys                |
+--------------------+
9 rows in set (0.002 sec)
```

Display a List of Database Users

Display the existing users by invoking the following SQL statement:

```
mysql> select user from mysql.user;
The preceding command displays the following output:
+------------------+
| user             |
+------------------+
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
+------------------+
4 rows in set (0.001 sec)
```

Dropping a Database

Log into MySQL and invoke the following command to create, select, and then drop the pizza database:

```
MySQL [(none)]> create database pizza;
Query OK, 1 row affected (0.004 sec)

MySQL [(none)]> use pizza;
Database changed
MySQL [pizza]> drop database pizza;
Query OK, 0 rows affected (0.007 sec)
```

Although performing this task with a database that does not contain any data might seem pointless, it is very straightforward and you will already know how to perform this task if it becomes necessary to do so in the future.

EXPORTING A DATABASE

Although you currently have an empty database, it is still good to know the steps for exporting a database, which is handy as a backup and also provides a simple way to create a copy of an existing database on another machine.

By way of illustration, we will first create the database called minimal in MySQL, as shown here:

```
MySQL [mytools]> create database minimal;
Query OK, 1 row affected (0.006 sec)
```

Next, invoke the mysqldump command to export the minimal database, as shown here:

```
mysqldump -u username -p"password" -R minimal > minimal.sql
```

Notice the following details at the preceding command. First, there are no intervening spaces between the -p flag and the password in order to bypass a command line prompt to enter the password. Second, make sure that you omit the quote marks. Third, the `-R` flag instructs mysqldump to copy stored procedures and functions in addition to the database data.

At this point you can create tables in the `minimal` database and periodically export its contents. If you are curious, Listing 4.1 displays the contents of `minimal.sql`, which is the complete description of the `minimal` database.

LISTING 4.1: minimal.sql

```
-- MariaDB dump 10.18  Distrib 10.5.8-MariaDB, for osx10.15
(x86_64)
--
-- Host: localhost    Database: minimal
-- ------------------------------------------------------
-- Server version      8.0.21

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_
CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_
RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_
CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_
CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_
VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Dumping routines for database 'minimal'
--
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_
RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION
*/;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2022-02-03 22:44:54
```

RENAMING A DATABASE

Although you currently have an empty database, it is still good to know how to rename a database (and besides, it is faster to do so with an empty database).

Older versions of MySQL provided the RENAME DATABASE command to rename a database; however, newer versions of MySQL have removed this functionality in order to avoid security risks.

Fortunately, you can perform a three-step process involving several MySQL command line utilities to rename a MySQL database OLD_DB (which you need to replace with the name of the database that you want to rename) to a new database NEW_DB (replaced with the actual new database name):

```
Step 1. Create an exported copy of database OLD_DB
Step 2. Create a new database called NEW_DB
Step 3. Import data from OLD_DB into NEW_DB
```

Perform step 1) by invoking the following command (see previous section):

```
mysqldump -u username -p"password" -R OLD_DB > OLD_DB.sql
```

Perform step 2) by invoking the following command:

```
mysqladmin -u username -p"password" create NEW_DB
```

Perform step 3) by invoking the following command:

```
mysql -u username -p"password" newDbName < OLD_DB.sql
```

Verify that everything worked correctly by logging into MySQL and selecting the new database:

```
MySQL [mysql]> use NEW_DB;
Database changed
```

THE INFORMATION_SCHEMA TABLE

The INFORMATION_SCHEMA table enables you to retrieve information about the columns in a given table, and it has the following attributes:

```
TABLE_SCHEMA
TABLE_NAME
COLUMN_NAME
ORDINAL_POSITION
COLUMN_DEFAULT
IS_NULLABLE
DATA_TYPE
CHARACTER_MAXIMUM_LENGTH
NUMERIC_PRECISION
NUMERIC_SCALE
DATETIME_PRECISION
```

For example, we will look at the structure of the weather table that is available in the companion files:

```
MySQL [mytools]> desc weather;
+----------+----------+------+-----+---------+-------+
| Field    | Type     | Null | Key | Default | Extra |
+----------+----------+------+-----+---------+-------+
| day      | date     | YES  |     | NULL    |       |
| temper   | int      | YES  |     | NULL    |       |
| wind     | int      | YES  |     | NULL    |       |
| forecast | char(20) | YES  |     | NULL    |       |
| city     | char(20) | YES  |     | NULL    |       |
| state    | char(20) | YES  |     | NULL    |       |
+----------+----------+------+-----+---------+-------+
6 rows in set (0.001 sec)
```

We can obtain additional information about the columns in the weather table with the following SQL query:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'weather'
AND table_schema = 'mytools';
The preceding SQL query generates the following output:
+-------------+-----------+-------------+----------------+
| COLUMN_NAME | DATA_TYPE | IS_NULLABLE | COLUMN_DEFAULT |
+-------------+-----------+-------------+----------------+
| city        | char      | YES         | NULL           |
| day         | date      | YES         | NULL           |
| forecast    | char      | YES         | NULL           |
| state       | char      | YES         | NULL           |
| temper      | int       | YES         | NULL           |
| wind        | int       | YES         | NULL           |
+-------------+-----------+-------------+----------------+
6 rows in set (0.001 sec)
```

THE PROCESSLIST TABLE

The PROCESSLIST table contains information about the status of SQL statements. This information is useful when you want to see the status of table-level or row-level locks on a table (discussed in Chapter 2). The following SQL statement shows you an example of the contents of this table:

```
MySQL [mytools]> show processlist;
+----+-----------------+-----------+---------+---------+---
-----+------------------------+------------------+
| Id | User            | Host      | db      | Command |
Time   | State                  | Info             |
+----+-----------------+-----------+---------+---------+---
-----+------------------------+------------------+
|  5 | event_scheduler | localhost | NULL    | Daemon  |
138765 | Waiting on empty queue | NULL             |
|  9 | root            | localhost | mytools | Query   |
0 | starting                | show processlist |
+----+-----------------+-----------+---------+---------+---
-----+------------------------+------------------+
2 rows in set (0.000 sec)
```

SQL FORMATTING TOOLS

As you might expect, there are various formatting styles for SQL statements, and you can peruse them to determine which style is most appealing to you. The following link is for an online SQL formatter:

https://codebeautify.org/sqlformatter

The following link contains 18 SQL formatters, some of which are commercial and some are free:

https://www.sqlshack.com/sql-formatter-tools/

The following link contains a list of SQL formatting conventions (i.e., it is not about formatting tools):

https://opendatascience.com/best-practices-sql-formatting

If you work in an environment where the SQL formatting rules have already been established, it might be interesting to compare that style with those of the SQL formatting tools in the preceding links.

If you are a SQL beginner working on your own, it is also worth exploring these links as you learn more about SQL statements throughout this book. As you gain more knowledge about writing SQL statements, you will encounter various styles in blog posts, which means you will also notice which conventions those blog posts adopt for formatting SQL statements.

SUMMARY

This chapter started with an introduction to the concept of an RDBMS, and the rationale for using an RDBMS. In particular, you saw an example of an RDBMS with a single table, two tables, and four tables (and much larger RDBMSes abound).

Then you got a brief introduction to the notion of database normalization, and how doing so will help you maintain data integrity ("single source of truth") in an RDBMS.

Next, you learned about the structure of the tables in a four-table database that keeps track of customer purchases of tools through a Web page. You also saw which tables have a one-to-many relationship so that you can find all the line items that belong to a given purchase order.

In addition, you got a brief introduction to SQL and some basic examples of SQL queries (more details are in Chapter 2). You also learned about various types of SQL statements that can be classified as DCL (data control language, DDL (data definition language), DQL (data query language), or DML (data manipulation language).

Finally, you learned about MySQL, some simple SQL statements, data types in MySQL, and various operations that you can perform with MySQL databases.

CHAPTER 5

WORKING WITH SQL AND MYSQL

The previous chapter provided a fast-paced introduction to RDBMSes and SQL concepts, whereas this chapter focuses on MySQL and the SQL statements that are necessary to manage database tables and the data in those tables. MySQL is used for the SQL in this book because it is an RDBMS that is available as a free download from an Oracle website. Moreover, virtually everything that you learn about MySQL in this chapter will transfer to other RDBMSes, such as PostgreSQL and ORACLE.

The first part of this chapter presents various ways to create MySQL tables, which can be done manually, from SQL scripts, or from the command line. You will also see how to create a MySQL table that contains Japanese text that contains a mixture of Kanji and Hiragana. This section also shows you how to drop and alter MySQL tables, and how to populate MySQL tables with seed data.

The second part of this chapter contains an assortment of SQL statements that use the SELECT keyword. You will see SQL statements that find the distinct rows in a MySQL table as well as the unique rows, along with using the EXISTS and LIMIT keywords. This section also explains the differences among the DELETE, TRUNCATE, and DROP keywords in SQL.

The third part of this chapter shows you how to create indexes on MySQL tables, and some criteria for defining indexes, followed by how to select columns for an index. Although the four tables in Chapter 1 are small enough that they do not require any indexes, it is important to understand the purpose of indexes and how to create them.

The final part of this chapter shows you how to export the result set of a SQL query, and also how to export a database as well as the entire contents of a database.

CREATE DATABASE TABLES

Log into MySQL and select the mytools database as shown below:

```
MySQL [(none)]> use mytools;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
```

There are three ways to create database tables in MySQL as well as other RDBMSes. One technique is manual (shown first); another technique (shown second) invokes a SQL file that contains suitable SQL commands; a third technique involves redirecting a SQL file to the MySQL executable from the command line.

The next section shows you how to create the four tables (described in Chapter 4) for the mini application.

Manually Creating Tables for mytools.com

This section shows you how to manually create the four tables for the mytools database in MySQL. Specifically, you will see how to create the following four tables:

- customers
- purchase_orders
- line_items
- item_desc

Log into MySQL, and after selecting the mytools database, type the following commands to create the required tables:

```
MySQL [mytools]> CREATE TABLE customers (cust_id INTEGER, first_name
VARCHAR(20), last_name VARCHAR(20), home_address VARCHAR(20), city
VARCHAR(20), state VARCHAR(20), zip_code VARCHAR(10));

MySQL [mytools]> CREATE TABLE purchase_orders ( cust_id INTEGER,
po_id INTEGER, purchase_date date);

MySQL [mytools]> CREATE TABLE line_items (po_id INTEGER, line_
id INTEGER, item_id INTEGER, item_count INTEGER, item_price
DECIMAL(8,2), item_tax DECIMAL(8,2), item_subtotal DECIMAL(8,2));

MySQL [mytools]> CREATE TABLE item_desc (item_id INTEGER, item_
desc VARCHAR(80), item_price DECIMAL(8,2));
```

Describe the structure of the customers table with the following command:

```
MySQL [mytools]> desc customers;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| cust_id      | int         | YES  |     | NULL    |       |
| first_name   | varchar(20) | YES  |     | NULL    |       |
| last_name    | varchar(20) | YES  |     | NULL    |       |
| home_address | varchar(20) | YES  |     | NULL    |       |
| city         | varchar(20) | YES  |     | NULL    |       |
| state        | varchar(20) | YES  |     | NULL    |       |
| zip_code     | varchar(10) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
7 rows in set (0.003 sec)
```

Describe the structure of the `purchase_orders` table with the following command:

```
MySQL [mytools]> desc purchase_orders;
+---------------+------+------+-----+---------+-------+
| Field         | Type | Null | Key | Default | Extra |
+---------------+------+------+-----+---------+-------+
| cust_id       | int  | YES  |     | NULL    |       |
| po_id         | int  | YES  |     | NULL    |       |
| purchase_date | date | YES  |     | NULL    |       |
+---------------+------+------+-----+---------+-------+
3 rows in set (0.004 sec)
```

Describe the structure of the `line_items` table with the following command:

```
MySQL [mytools]> desc line_items;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Yes  | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| po_id         | int          | YES  |     | NULL    |       |
| line_id       | int          | YES  |     | NULL    |       |
| item_id       | int          | YES  |     | NULL    |       |
| item_count    | int          | YES  |     | NULL    |       |
| item_price    | decimal(8,2) | YES  |     | NULL    |       |
| item_tax      | decimal(8,2) | YES  |     | NULL    |       |
| item_subtotal | decimal(8,2) | YES  |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
7 rows in set (0.002 sec)
```

Describe the structure of the `item_desc` table with the following command:

```
MySQL [mytools]> desc item_desc;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| item_id    | int          | YES  |     | NULL    |       |
| item_desc  | varchar(80)  | YES  |     | NULL    |       |
| item_price | decimal(8,2) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
3 rows in set (0.006 sec)
```

Creating Tables via an SQL Script for mytools.com

The previous section shows you a manual technique for creating database tables, and this section shows you how to create the required tables by launching the `SQL` file `mytools_create_tables.sql` whose contents are displayed in Listing 5.1.

LISTING 5.1: mytools_create_tables.sql

```
USE mytools;

-- drop tables if they already exist:
DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS purchase_orders;
DROP TABLE IF EXISTS line_items;
DROP TABLE IF EXISTS item_desc;

—- these SQL statements are the same as the previous section:
CREATE TABLE customers (cust_id INTEGER, first_name
VARCHAR(20), last_name VARCHAR(20), home_address
VARCHAR(20), city VARCHAR(20), state VARCHAR(20), zip_code
VARCHAR(10));

CREATE TABLE purchase_orders ( cust_id INTEGER, po_id
INTEGER, purchase_date date);

CREATE TABLE line_items (po_id INTEGER, line_id
INTEGER, item_id INTEGER, item_count INTEGER, item_
price DECIMAL(8,2), item_tax DECIMAL(8,2), item_subtotal
DECIMAL(8,2));

CREATE TABLE item_desc (item_id INTEGER, item_desc
VARCHAR(80), item_price DECIMAL(8,2));
```

Listing 5.1 contains three sections. The first section selects the `mytools` database, and the second section drops any of the four required tables if they already exist. The third section contains the `SQL` commands to create the four required tables.

Creating Tables With Japanese Text

Although this section is not required for any of the code samples in this book, it is nonetheless interesting to see how easily you can create a `MySQL` table with Japanese text. In case you are wondering, the Japanese text was inserted from a MacBook after adding a Hiragana keyboard and a Katakana keyboard. Perform an online search for instructions that show you how to add these keyboards to your laptop.

Listing 5.2 displays the contents of `japanese1.sql` that illustrates how to create a `MySQL` table that is populated with Japanese text.

LISTING 5.2: japanese1.sql

```
use mytools;
DROP TABLE IF EXISTS japn1;

CREATE TABLE japn1
(
   emp_id INT NOT NULL AUTO_INCREMENT,
   fname VARCHAR(100) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
   lname VARCHAR(100) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
   title VARCHAR(100) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
   PRIMARY KEY (emp_id)
);

INSERT INTO japn1 SET fname="ひでき", lname="日浦",title="しちお";
INSERT INTO japn1 SET fname="ももたろ", lname="つよい",title="かちょ";
INSERT INTO japn1 SET fname="オズワルド", lname="カmポ",title="悪ガキ";
INSERT INTO japn1 SET fname="東京", lname="日本",title="すごい！";

\! echo '=> All rows in table japn1:';
SELECT * FROM japn1;

\! echo '=> Rows whose lname contains カ:';
SELECT * FROM japn1
WHERE lname LIKE '%カ%';
```

Listing 5.2 starts with the definition of the table `japn1` that defines the `fname`, `lname`, and `title` attributes as `VARCHAR(100)` and also specifies `utf8` as the character set and `utf8_general_ci` as the collating sequence. These extra keywords enable us to store Hiragana and Kanji characters in these three attributes. Launch the code in Listing 5.2 from the `MySQL` prompt and you will see the following output:

```
Database changed
Query OK, 0 rows affected (0.005 sec)
Query OK, 0 rows affected, 6 warnings (0.005 sec)
Query OK, 1 row affected (0.001 sec)
Query OK, 1 row affected (0.001 sec)
Query OK, 1 row affected (0.001 sec)
Query OK, 1 row affected (0.001 sec)

=> All rows in table japn1:
+--------+-----------------+-----------+--------------+
| emp_id | fname           | lname     | title        |
+--------+-----------------+-----------+--------------+
|      1 | ひでき          | 日浦      | しちお       |
|      2 | ももたろ        | つよい    | かちょ       |
|      3 | オズワルド      | カmポ     | 悪ガキ       |
|      4 | 東京            | 日本      | すごい！     |
+--------+-----------------+-----------+--------------+
4 rows in set (0.000 sec)

=> Rows whose lname matches カ:
+--------+-----------------+---------+-----------+
| emp_id | fname           | lname   | title     |
+--------+-----------------+---------+-----------+
|      3 | オズワルド      | カmポ   | 悪ガキ    |
+--------+-----------------+---------+-----------+
1 row in set (0.000 sec)
```

The preceding example is a rudimentary example of working with Japanese text in a `MySQL` table. Chapter 4 shows you how to perform a join on the table `japn1` with the table `japn2`, where the text in `japn2` contains the English counterpart to the text in `japn1`. You can also search online for other `SQL`-based operations that you can perform with this data, as well as examples of creating `MySQL` tables for other Asian languages.

Creating Tables From the Command Line

The third technique for invoking a `SQL` file is from the command line. First make sure that the specified database already exists (such as `mytools`). Next, invoke the following command from the command line to execute the contents of `employees.sql` in `MySQL`:

```
mysql --password=<your-password> --user=root mytools < user.sql
```

Listing 5.3 displays the contents of `user.sql` that illustrates how to create a database table and populate that table with data.

LISTING 5.3: user.sql

```
USE mytools;

DROP TABLE IF EXISTS user;
CREATE TABLE user (user_id INTEGER(8), user_title VARCHAR(20));

INSERT INTO user VALUES (1000, 'Developer');
INSERT INTO user VALUES (2000, 'Project Lead');
INSERT INTO user VALUES (3000, 'Dev Manager');
INSERT INTO user VALUES (4000, 'Senior Dev Manager');
```

Now log into `MySQL` with the following command from the command line:

```
mysql --password=<your-password> —user=root
```

Now enter the following two commands (shown in bold):

```
MySQL [(none)]> use mytools;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [mytools]> desc user;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| user_id    | int         | YES  |     | NULL    |       |
| user_class | int         | YES  |     | NULL    |       |
| user_title | varchar(20) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.002 sec)
```

DROP DATABASE TABLES

There are several ways of dropping database tables in `MySQL` that are described in the following subsections.

Dropping Tables via a SQL Script for mytools.com

Sometimes you might want to simply drop database tables without recreating them. Listing 5.4 displays the contents of `mytools_drop_tables.sql` that illustrates how to drop database tables without recreating them.

LISTING 5.4: mytools_drop_tables.sql

```
USE mytools;

-- drop tables if they already exist:
DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS purchase_orders;
DROP TABLE IF EXISTS line_items;
DROP TABLE IF EXISTS item_desc;
```

Listing 5.4 is straightforward: it consists of section 1 and section 2 of the SQL file displayed in Listing 5.1.

ALTERING DATABASE TABLES WITH THE ALTER KEYWORD

If you want to modify the columns in a table, you can use the ALTER command to add new columns, drop existing columns, or modify the data type of an existing column. Whenever a new column is added to a database table, that column will contain NULL values. However, you can invoke SQL statements to populate the new column with values, as shown in the next section.

Add a Column to a Database Table

As a simple example, we will create the table user2 from table user, as shown here:

```
CREATE TABLE user2 AS (SELECT * FROM user);
```

Add the character columns fname and lname to table user2 by executing the following SQL commands:

```
MySQL [mytools]>
ALTER TABLE user2
ADD COLUMN fname VARCHAR(20);
Query OK, 0 rows affected (0.011 sec)
Records: 0  Duplicates: 0  Warnings: 0

MySQL [mytools]>
ALTER TABLE user2
ADD COLUMN lname VARCHAR(20);
Query OK, 0 rows affected (0.012 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Now look at the structure of table user2, which contains two new columns with NULL values:

```
MySQL [mytools]> desc user2;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| user_id    | int         | YES  |     | NULL    |       |
| user_title | varchar(20) | YES  |     | NULL    |       |
| fname       | varchar(20) | YES  |     | NULL    |       |
| lname       | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.002 sec)
```

Now look at the rows in table `user2` by issuing the following `SQL` query:

```
select * from user2;
+---------+--------------------+-------+-------+
| user_id | user_title         | fname | lname |
+---------+--------------------+-------+-------+
|    1000 | Developer          | NULL  | NULL  |
|    2000 | Project Lead       | NULL  | NULL  |
|    3000 | Dev Manager        | NULL  | NULL  |
|    4000 | Senior Dev Manager | NULL  | NULL  |
+---------+--------------------+-------+-------+
4 rows in set (0.001 sec)
```

How do we insert appropriate values for the new `fname` and `lname` attributes for each existing row? One way to update these attributes is to issue a `SQL` query for each row that updates these attributes based on the `user_id`:

```
UPDATE user2
SET fname = 'John', lname = 'Smith'
WHERE user_id = 1000;

UPDATE user2
SET fname = 'Jane', lname = 'Stone'
WHERE user_id = 2000;

UPDATE user2
SET fname = 'Dave', lname = 'Dodds'
WHERE user_id = 3000;

UPDATE user2
SET fname = 'Jack', lname = 'Jones'
WHERE user_id = 4000;
```

We can confirm that the `user2` table has been updated correctly with the following `SQL` query:

```
select * from user2;
+---------+--------------------+-------+-------+
| user_id | user_title         | fname | lname |
+---------+--------------------+-------+-------+
|    1000 | Developer          | John  | Smith |
|    2000 | Project Lead       | Jane  | Stone |
|    3000 | Dev Manager        | Dave  | Dodds |
|    4000 | Senior Dev Manager | Jack  | Jones |
+---------+--------------------+-------+-------+
4 rows in set (0.000 sec)
```

Unfortunately, the preceding solution is not scalable if you need to update hundreds or thousands of rows with values for the new attributes. There are several options available, depending on the location of the values for the new attributes: one option involves importing data and another involves programmatically generating SQL statements.

If you have a CSV file that contains the complete data for the table rows, including values for the fname and lname attributes, the solution is straightforward: delete the rows from the user2 table and then import the data from the CSV file into the user2 table.

However, if the existing data is located in one CSV file and the data for the two new attributes is located in a separate CSV file, you need to merge the two CSV files into a single CSV file, after which you can import the CSV file directly into the user2 table. An example of performing this task is discussed after the following section that describes referential constraints.

Drop a Column From a Database Table

The following SQL statement illustrates how to drop the column str_date from the table mytable:

```
ALTER TABLE mytable
DROP COLUMN str_date;
```

Just to be safe, it is a good idea to make a backup of a table before you drop any of its columns. If you have enough disk space and/or the table is medium-sized or smaller, you can create an online backup with this SQL statement:

```
CREATE TABLE mytable_backup AS (SELECT * FROM mytable);
```

Change the Data Type of a Column

Listing 5.5 displays the contents of people_ages.sql that illustrates how to change the data type of a column in a MySQL table.

LISTING 5.5: people_ages.sql

```
USE mytools;
DROP TABLE IF EXISTS people_ages;
CREATE TABLE people_ages (float_ages DECIMAL(4,2), floor_ages INT);

INSERT INTO people_ages VALUES (12.3,0);
INSERT INTO people_ages VALUES (45.6,0);
INSERT INTO people_ages VALUES (78.9,0);
INSERT INTO people_ages VALUES (-3.4,0);
DESC people_ages;
SELECT * FROM people_ages;

-- populate floor_ages with FLOOR (=INT) value:
UPDATE people_ages
SET floor_ages = FLOOR(float_ages);
SELECT * FROM people_ages;

-- change float_ages to INT data type:
ALTER TABLE people_ages CHANGE float_ages int_ages INT;
DESC people_ages;
SELECT * FROM people_ages;
```

```
-- rows whose minimum age is less than min_value:
SELECT @min_value := 2;
SELECT * FROM people_ages WHERE floor_ages < @min_value;
```

Listing 5.5 creates and populates the people_ages table with data. The other code in Listing 5.5 contains three SQL statements, each of which starts with a comment statement that explains its purpose.

The first SQL statement populates the integer-valued column floor_ages with the floor of the float_ages column via the built-in FLOOR() function.

The second SQL statement alters the decimal-valued column float_ages to a column of type INT.

The third SQL statement displays the rows in the people_ages table whose floor_ages value is less than min_value.

Now launch the code in Listing 5.5 and you will see the following output:

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| float_ages | decimal(4,2) | YES  |     | NULL    |       |
| floor_ages | int          | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
2 rows in set (0.001 sec)

+------------+------------+
| float_ages | floor_ages |
+------------+------------+
|      12.30 |          0 |
|      45.60 |          0 |
|      78.90 |          0 |
|      -3.40 |          0 |
+------------+------------+
4 rows in set (0.000 sec)

Query OK, 4 rows affected (0.001 sec)
Rows matched: 4  Changed: 4  Warnings: 0

+------------+------------+
| float_ages | floor_ages |
+------------+------------+
|      12.30 |         12 |
|      45.60 |         45 |
|      78.90 |         78 |
|      -3.40 |         -4 |
+------------+------------+
4 rows in set (0.000 sec)

Query OK, 4 rows affected (0.014 sec)
Records: 4  Duplicates: 0  Warnings: 0

+------------+------+------+-----+---------+-------+
| Field      | Type | Null | Key | Default | Extra |
+------------+------+------+-----+---------+-------+
| int_ages   | int  | YES  |     | NULL    |       |
| floor_ages | int  | YES  |     | NULL    |       |
+------------+------+------+-----+---------+-------+
2 rows in set (0.001 sec)
```

```
+----------+------------+
| int_ages | floor_ages |
+----------+------------+
|       12 |         12 |
|       46 |         45 |
|       79 |         78 |
|       -3 |         -4 |
+----------+------------+
4 rows in set (0.000 sec)

+-----------------+
| @min_value := 2 |
+-----------------+
|               2 |
+-----------------+
1 row in set, 1 warning (0.000 sec)

+----------+------------+
| int_ages | floor_ages |
+----------+------------+
|       -3 |         -4 |
+----------+------------+
1 row in set (0.000 sec)
```

What Are Referential Constraints?

Referential constraints (also called constraints) prevent the insertion of invalid data into database tables. In general, constraints on a table are specified during the creation of the table. Here is a list of constraints that SQL implementations support:

- CHECK
- DEFAULT
- FOREIGN KEY
- PRIMARY KEY
- NOT NULL
- UNIQUE

In case you do not already know, an *orphan row* in a database table is a row without its associated parent row that's typically stored in a separate table. An example would be a customer in the (parent) customers table and the associated (child) rows in the purchase_orders table. Note that a similar relationship exists between the (parent) purchase_orders table and the associated (child) rows in the line_items table.

COMBINING DATA FOR A TABLE UPDATE (OPTIONAL)

This section shows you how to perform the task described in the previous section: how to merge two CSV files and load the result into a database table. This section is optional because the solution involves Pandas, which has not been discussed yet. You can skip this section with no loss of continuity, and perhaps return to this section when you need to perform this task. Also keep in mind that there are other ways to perform the tasks in this section.

The first subsection shows you how to *merge* the columns of a CSV file into the columns of another CSV file, and then save the updated CSV file to the file system. The second subsection shows you how to *append* the contents of a CSV file to the contents of another CSV file, and then save the updated CSV file to the file system.

Merging Data for a Table Update

Suppose that we have a CSV files called user.csv with a set of columns and that we want to merge the columns of user.csv with columns of the CSV file user2.csv. For simplicity, assume that there are no missing values in either CSV file.

Listing 5.6 displays the contents of user.csv that contains the original data for the user table, and Listing 5.7 displays the contents of user2.csv that contains the data for the fname and lname attributes.

LISTING 5.6: user.csv

```
fname,lname
id,title
1000,Developer
2000,Project Lead
3000,Dev Manager
4000,Senior Dev Manager
```

LISTING 5.7: user2.csv

```
fname,lname
1000,John,Smith
2000,Jane,Stone
3000,Dave,Dodds
4000,Jack,Jones
```

Listing 5.8 displays the contents of user_merged.py that illustrates how to use Pandas data frames to merge two CSV file and generate a CSV file with the merged data.

LISTING 5.8: user_merged.py

```
import pandas as pd

df_user = pd.read_csv("user.csv")
df_user2 = pd.read_csv("user2.csv")
df_user['fname'] = df_user2['fname'].values
df_user['lname'] = df_user2['lname'].values
df_user.to_csv('user_merged.csv', index=False)
```

Listing 5.8 contains an import statement followed by assigning the contents of user.csv and user2.csv to the Pandas data frames df_user and df_user2, respectively. The next pair of code snippets create the columns fname and lname in the df_users data frame and initialize their values from the corresponding columns in the df_user2 data frame.

The last code snippet in Listing 5.8 saves the updated data frame to the CSV file user_merged.csv, which is located in the same directory as the CSV files user.csv and user2.csv. Now launch the code in Listing 5.8 in order to generate the CSV file user_merged.csv, whose contents are displayed in Listing 5.9.

LISTING 5.9: user_merged.csv

```
id,title,fname,lname
5000,Developer,Sara,Edwards
6000,Project Lead,Beth,Woodward
7000,Dev Manager,Donald,Jackson
8000,Senior Dev Manager,Steve,Edwards
```

If need be, the code in Listing 5.7 can be modified to insert the fname values and the lname value in the first two columns.

Appending Data to a Table From a CSV File

Suppose that we have two CSV files called user_merged.csv and user_merged2.csv that contain the same columns. For simplicity, we will also assume that there are no missing values in either CSV file.

Listing 5.10 displays the contents of user_merged.csv and Listing 5.11 displays the contents of user_merged2.csv.

LISTING 5.10: user_merged.csv

```
id,title,fname,lname
5000,Developer,Sara,Edwards
6000,Project Lead,Beth,Woodward
7000,Dev Manager,Donald,Jackson
8000,Senior Dev Manager,Steve,Edwards
```

LISTING 5.11: user_merged2.csv

```
id,title,fname,lname
5000,Developer,Sara,Edwards
6000,Project Lead,Beth,Woodward
7000,Dev Manager,Donald,Jackson
8000,Senior Dev Manager,Steve,Edwards
```

Listing 5.12 displays the contents of merge_all_data.py that illustrates how to use Pandas data frames to concatenate the contents of two or more CSV files in the same directory and generate a CSV file with the merged data. This code sample generalizes the code in Listing 5.8 that concatenates only two CSV files.

LISTING 5.12: merge_all_data.py

```
import glob
import os
import pandas as pd

# merge the data-related files as one data frame:
df = pd.concat(map(pd.read_csv, glob.glob(os.path.join('',
"data*.csv"))))
# save data frame to a CSV file:
df.to_csv('all_data.csv')
```

Listing 5.12 contains an `import` statement followed by assigning the contents of `user.csv` and `user2.csv` to the `Pandas` data frames `df_user` and `df_user2`, respectively. The next pair of code snippets create the columns `fname` and `lname` in the `df_users` data frame and initialize their values from the corresponding columns in the `df_user2` data frame.

The last code snippet in Listing 5.12 saves the updated data frame to the `CSV` file `all_data.csv`, which is located in the same directory as the `CSV` files `user_merged.csv` and `user_merged2.csv`. Now launch the code in Listing 5.12 to generate the `CSV` file `user_merged3.csv`, whose contents are displayed in Listing 5.13.

LISTING 5.13: all_data.csv

```
id,title
1000,Developer
2000,Project Lead
3000,Dev Manager
4000,Senior Dev Manager
1000,Developer
2000,Project Lead
3000,Dev Manager
4000,Senior Dev Manager
1000,Developer
2000,Project Lead
3000,Dev Manager
4000,Senior Dev Manager
1000,Developer
2000,Project Lead
3000,Dev Manager
4000,Senior Dev Manager
```

Appending Table Data from CSV Files via SQL

Suppose that the data in the `CSV` file `user_merged.csv` has already been inserted into table `user3`. We can use the following `SQL` statement to insert the contents of the `CSV` file `user_merged2.csv` into the table `user3` as follows:

```
LOAD DATA INFILE 'user_merged.csv'
    INTO TABLE user3
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '/n';
```

Depending on the manner in which the MySQL server was launched, you might encounter the following error message:

```
ERROR 1290 (HY000): The MySQL server is running with
the --secure-file-priv option so it cannot execute this
statement
```

The preceding error occurs due to either of the following reasons:

- The SQL statement specified an incorrect path to the file
- No directory is specified under secure_file_priv variable

```
Select @@global.secure_file_priv;
+---------------------------+
| @@global.secure_file_priv |
+---------------------------+
| NULL                      |
+---------------------------+
1 row in set (0.001 sec)
```

Another similar query is shown below:

```
SHOW VARIABLES LIKE "secure_file_priv";
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| secure_file_priv | NULL  |
+------------------+-------+
1 row in set (0.022 sec)
```

If you have verified that the path to the file is correct and you still see the same error message, then launch the following command (requires root access):

```
sudo /usr/local/mysql/support-files/mysql.server restart
--secure_file_priv=/tmp
```

Keep in mind that you might need to replace preceding command with a command that is specific to your system, which depends on a combination of:

- the operating system (Windows/Mac/Linux)
- the version of MySQL on your system
- the utility that installed MySQL (brew, .dmg file, and so forth)

Perform an online search to find a solution that is specific to your MySQLinstallation on your machine. Keep in mind that some solutions specify modifying the file `/etc/my.ini` or `/etc/my.cnf`, neither of which exists on Mac Catalina with MySQL 8.

Another possibility is the following SQL statement that specifies LOCAL:

```
LOAD DATA **LOCAL** INFILE "user_merged.csv" INTO TABLE user3;
```

Unfortunately, the preceding SQL statement does not work with MySQL 8: you will see the following error message:

```
ERROR 3948 (42000): Loading local data is disabled; this
must be enabled on both the client and server sides
```

Fortunately, there is a solution: MySQL Workbench that enables you to export tables and databases via a GUI interface, and also how to import databases and CSV files into tables. Perform an online search for MySQL Workbench to download the distribution and perform an installation. If you launch a SQL query from inside SQL Workbench, simply click on the "Export" icon, and then follow the subsequent prompts to save the result set to a file with the desired format.

INSERTING DATA INTO TABLES

This section shows you several SQL statements for inserting data into database tables in the mytools database. The following SQL statements insert data into the customers, purchase_orders, and line_items tables, respectively:

```
use mytools;

-- create a new customer:
INSERT INTO customers
VALUES (1000,'John','Smith','123 Main St','Fremont','CA','94123');

-- create a new purchase order:
INSERT INTO purchase_orders VALUES (1000,12500, '2021-12-01');

-- line item => one hammer:
INSERT INTO line_items VALUES (12500,5001,100,1,20.00,2.00,22.00);

-- line item => two screwdrivers:
INSERT INTO line_items VALUES (12500,5002,200,2,8.00,1.60,17.60);

-- line item => three wrenches:
INSERT INTO line_items VALUES (12500,5003,300,3,10.00,3.00,33.20);
```

You can also create a SQL file that consists of multiple SQL INSERT statements that populate one or more tables with data. In addition, you can upload data from CSV files into database tables, which is discussed in the next section.

POPULATING TABLES FROM TEXT FILES

Log into MySQL, select the mytools database, and invoke the following command to create the people table:

```
MySQL [mytools]> CREATE TABLE people (fname VARCHAR(20),
lname VARCHAR(20), age VARCHAR(20), gender CHAR(1), country
VARCHAR(20));
```

Describe the structure of the `people` table with the following command:

```
MySQL [mytools]> desc people;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| fname   | varchar(20) | YES  |     | NULL    |       |
| lname   | varchar(20) | YES  |     | NULL    |       |
| age     | varchar(20) | YES  |     | NULL    |       |
| gender  | char(1)     | YES  |     | NULL    |       |
| country | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
5 rows in set (0.002 sec)
```

Listing 5.14 displays the contents of the `CSV` file `people.csv` that contains information about several people that will be inserted into the `people` table.

LISTING 5.14: people.csv

```
fname,lname,age,gender,country
john,smith,30,m,usa
jane,smith,31,f,france
jack,jones,32,m,france
dave,stone,33,m,italy
sara,stein,34,f,germany
```

Listing 5.15 displays the contents of `people.sql` that contains several `SQL` commands for inserting the data in Listing 5.19 into the `people` table.

LISTING 5.15: people.sql

```
INSERT INTO people VALUES ('john','smith','30','m','usa');
INSERT INTO people VALUES ('jane','smith','31','f','france');
INSERT INTO people VALUES ('jack','jones','32','m','france');
INSERT INTO people VALUES ('dave','stone','33','m','italy');
INSERT INTO people VALUES ('sara','stein','34','f','germany');
INSERT INTO people VALUES ('eddy','bower','35','m','spain');
```

As you can see, the `INSERT` statements in Listing 5.15 contain data that is located in `people.csv`. Now log into `MySQL`, select the `mytools` database, and invoke the following command to populate the `people` table:

```
MySQL [mysql]> source people.sql
Query OK, 1 row affected (0.004 sec)
Query OK, 1 row affected (0.001 sec)
Query OK, 1 row affected (0.001 sec)
Query OK, 1 row affected (0.001 sec)
Query OK, 1 row affected (0.001 sec)
Query OK, 1 row affected (0.001 sec)
```

Execute the following SQL statement to display the contents of the people table:

```
MySQL [mysql]> select * from people;
+-------+-------+------+--------+---------+
| fname | lname | age  | gender | country |
+-------+-------+------+--------+---------+
| john  | smith | 30   | m      | usa     |
| jane  | smith | 31   | f      | france  |
| jack  | jones | 32   | m      | france  |
| dave  | stone | 33   | m      | italy   |
| sara  | stein | 34   | f      | germany |
| eddy  | bower | 35   | m      | spain   |
+-------+-------+------+--------+---------+
6 rows in set (0.000 sec)
```

The second option involves manually executing each SQL statement in Listing 5.20, which is obviously inefficient for a large number of rows. The third option involves loading data from a CSV file into a table, as shown below:

```
MySQL [mysql]> LOAD DATA LOCAL INFILE 'people.csv' INTO TABLE people;
```

However, you might encounter the following error (which depends on the configuration of MySQL on your machine):

```
ERROR 3948 (42000): Loading local data is disabled; this
must be enabled on both the client and server sides
```

In general, a SQL script is preferred because it is easy to execute multiple times, and you can schedule SQL scripts to run as "cron" jobs.

WORKING WITH SIMPLE SELECT STATEMENTS

Earlier in this chapter you saw examples of the SELECT keyword in SQL statements, and this section contains additional SQL statements to show you additional ways to select subsets of data from a table. In its simplest form, a SQL statement with the SELECT keyword looks like this:

```
SELECT [one-or-more-attributes]
FROM [one-table]
```

Specify an asterisk ("*") after the SELECT statement if you want to select all the attributes of a table. For example the following SQL statement illustrates how to select all rows from the people table:

```
MySQL [mytools]> select * from people;
+-------+-------+------+--------+---------+
| fname | lname | age  | gender | country |
+-------+-------+------+--------+---------+
| john  | smith | 30   | m      | usa     |
| jane  | smith | 31   | f      | france  |
| jack  | jones | 32   | m      | france  |
| dave  | stone | 33  | m      | italy   |
| sara  | stein | 34  | f      | germany |
| eddy  | bower | 35  | m      | spain   |
+-------+-------+-----+--------+---------+
6 rows in set (0.000 sec)
```

Issue the following SQL statement that contains the LIMIT keyword if you want only the first row from the people table:

```
select * from people limit 1;
+-------+-------+-----+--------+---------+
| fname | lname | age  | gender | country |
+-------+-------+-----+--------+---------+
| john  | smith | 30   | m      | usa     |
+-------+-------+-----+--------+---------+
1 row in set (0.000 sec)
```

Replace the number 1 in the previous SQL query with any other positive integer in order to display the number of rows that you need. Incidentally, if you replace the number 1 with the number 0 you will see 0 rows returned.

Include the WHERE keyword to specify a condition on the rows, which will return a (possibly empty) subset of rows:

```
SELECT [one-or-more-attributes]
FROM [one-or-more-tables]
WHERE [some condition]
```

For example, the following SQL statement illustrates how to display all the attributes of the rows in the people table where the first name is john:

```
MySQL [mytools]> select * from people where fname = 'john';
+-------+-------+-----+--------+---------+
| fname | lname | age  | gender | country |
+-------+-------+-----+--------+---------+
| john  | smith | 30   | m      | usa     |
+-------+-------+-----+--------+---------+
1 row in set (0.000 sec)
```

Include the ORDER BY to specify the order in which you want to display the rows:

```
SELECT *
FROM weather
ORDER BY city;
+------------+--------+------+----------+------+-------+
| day        | temper | wind | forecast | city | state |
+------------+--------+------+----------+------+-------+
| 2021-07-01 |     42 |   16 | Rain     |      | ca    |
| 2021-08-04 |     50 |   12 | Snow     |      | mn    |
| 2021-09-03 |     15 |   12 | Snow     | chi  | il    |
| 2021-04-03 |     78 |  -12 | NULL     | se   | wa    |
| 2021-04-01 |     42 |   16 | Rain     | sf   | ca    |
| 2021-04-02 |     45 |    3 | Sunny    | sf   | ca    |
| 2021-07-02 |     45 |   -3 | Sunny    | sf   | ca    |
| 2021-07-03 |     78 |   12 | NULL     | sf   | mn    |
| 2021-08-06 |     51 |   32 |          | sf   | ca    |
| 2021-09-01 |     42 |   16 | Rain     | sf   | ca    |
| 2021-09-02 |     45 |   99 |          | sf   | ca    |
+------------+--------+------+----------+------+-------+
11 rows in set (0.003 sec)
```

Although we will not cover the JOIN keyword, it is very important for retrieving related data from two or more tables.

Duplicate versus Distinct Rows

Unless it is explicitly stated, the default action for a SQL SELECT statement is to select all rows (which includes duplicates), as shown here:

```
SELECT department_id
FROM employees;
```

However, you can retrieve only distinct rows by specifying the keyword DISTINCT, an example of which is here:

```
SELECT DISTINCT department_id
FROM employees;
```

Later you will learn how to use the GROUP BY clause and the HAVING clause in SQL statements.

Unique Rows

The DISTINCT keyword selects a single row (i.e., it ignores duplicates), whereas the UNIQUE keyword selects a row only if that row does not have any duplicates. A query that contains the UNIQUE keyword returns the same result set as a query that contains the DISTINCT keyword if and only if there are no duplicate rows.

As a preview, the following SQL query contains a SQL subquery, which is a topic that outside the scope of this book. However, the SQL query is included in this section of the chapter so that you can compare the functionality of DISTINCT versus UNIQUE. With the preceding in mind, here is the SQL statement to find unique rows in a database table:

```
select city, state
from weather
where unique (select state from weather);
```

The EXISTS Keyword

The EXISTS keyword selects a row based on the existence of a value.

```
select city, state
from weather where exists
(select city from weather where city = 'abc');
Empty set (0.001 sec)
```

The preceding is somewhat contrived because it can be replaced with this simpler and intuitive query:

```
select city, state
from weather
where city = 'abc';
```

The LIMIT Keyword

The `LIMIT` keyword limits the number of rows that are in a result set. For example, the `weather` table contains 11 rows, as shown here:

```
SELECT COUNT(*) FROM weather;
+----------+
| count(*) |
+----------+
|       11 |
+----------+
1 row in set (0.001 sec)
```

If we want to see only three rows instead of all the rows in the weather table, issue the following `SQL` query:

```
SELECT city,state
FROM weather ORDER
BY state, city
LIMIT 3;
+------+-------+
| city | state |
+------+-------+
|      | ca    |
| sf   | ca    |
| sf   | ca    |
+------+-------+
3 rows in set (0.000 sec)
```

DELETE, TRUNCATE, AND DROP IN SQL

`SQL` enables you to delete *all* the data from a table in several ways. One way is to invoke the following `SQL` statement:

```
DELETE from customers;
```

However, if a database table has a large number of rows, a faster technique is the `TRUNCATE` statement, as shown here:

```
TRUNCATE customers;
```

Both of the preceding `SQL` commands involve removing rows from a table without dropping the table. If you want to drop the rows in a table then also drop the table, use the `DROP` statement as shown here:

```
DROP TABLE IF EXISTS customers;
```

More Options for the DELETE Statement in SQL

The preceding section showed you how to delete all the rows in a table, and this section shows you how to delete a subset of the rows in a table, which involves specifying a condition for the rows that you want to drop from a table.

The following SQL statement deletes the rows in the customers table where the first name is John:

```
DELETE
FROM customers
Where FNAME = 'John';
```

The next SQL statement deletes the rows in the customers table where the first name is John and the rows in the purchase_orders table that are associated with John:

```
DELETE
FROM customers
Where FNAME = 'John'
CASCADE;
```

The preceding SQL statement is called a "cascading delete" and is very useful when the rows in a table have external dependencies. For example, the customers table has a one-to-many relationship with the purchase_orders table; therefore, if you remove a "parent" row from the customers table, you want to remove the "child" rows from the purchase_orders table.

You can also specify the LIMIT keyword with DELETE, an example of which is shown here:

```
DELETE
FROM customers
Where FNAME = 'JOHN'
LIMIT 1;
```

The preceding SQL statement will delete one row, but there is an exception: the preceding SQL query will delete all rows whose name equals JOHN if you specify ON DELETE CASCADE in the table definition of the customers table.

CREATING TABLES FROM EXISTING TABLES IN SQL

SQL provides two ways to create new tables without specifying their attributes. One technique involves a SQL statement that contains the TEMPORARY keyword, and the second technique does not specify the TEMPORARY keyword.

A temporary table is useful when it is impractical to query data that requires a single SELECT statement with JOIN clauses. Instead, use a temporary table to store an immediate result and then process that data with other SQL queries. However, keep in mind that the query optimizer cannot optimizer SQL queries containing a temporary table.

Working With Temporary Tables in SQL

Before we create a temporary table, we will drop the `temp_cust` table if it already exists:

```
MySQL [mytools]> DROP TEMPORARY TABLE IF EXISTS temp_cust;
Query OK, 0 rows affected, 1 warning (0.000 sec)
```

The following SQL statement illustrates how to create the temporary table `temp_cust` from the `customers` table:

```
MySQL [mytools]> CREATE TEMPORARY TABLE IF NOT EXISTS temp_cust
                AS (SELECT * FROM customers);
Query OK, 1 row affected (0.019 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

The following SQL statement displays the structure of `temp_cust`:

```
MySQL [mytools]> DESC temp_cust;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| cust_id      | int         | YES  |     | NULL    | NULL  |
| first_name   | varchar(20) | YES  |     | NULL    | NULL  |
| last_name    | varchar(20) | YES  |     | NULL    | NULL  |
| home_address | varchar(20) | YES  |     | NULL    | NULL  |
| city         | varchar(20) | YES  |     | NULL    | NULL  |
| state        | varchar(20) | YES  |     | NULL    | NULL  |
| zip_code     | varchar(10) | YES  |     | NULL    | NULL  |
+--------------+-------------+------+-----+---------+-------+
7 rows in set (0.005 sec)
```

The `temp_cust` table contains the same data as the `customers` table, as shown here:

```
MySQL [mytools]> SELECT * FROM temp_cust;
+---------+------------+-----------+--------------+---------+-------+----------+
| cust_id | first_name | last_name | home_address | city    | state | zip_code |
+---------+------------+-----------+--------------+---------+-------+----------+
|    1000 | John       | Smith     | 123 Main St  | Fremont | CA    | 94123    |
+---------+------------+-----------+--------------+---------+-------+----------+
1 row in set (0.001 sec)
```

In addition, you can specify an index for a temporary table, as shown here:

```
CREATE TEMPORARY TABLE IF NOT EXISTS
  temp_cust3 ( INDEX(last_name) )
ENGINE=MyISAM
AS (
  SELECT first_name, last_name
  FROM customers
);
```

In fact, you can also create a temporary table with a primary key, as shown here:

```
MySQL [mytools]> CREATE TEMPORARY TABLE temp_cust4 ENGINE=MEMORY
    -> as (select * from customers);
Query OK, 1 row affected (0.003 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

However, keep in mind the following point: ENGINE=MEMORY is not supported when a table contains BLOB/TEXT columns. Now that you understand how to create tables with the TEMPORARY keyword, we will look at the preceding SQL statements when we omit the TEMPORARY keyword.

Creating Copies of Existing Tables in SQL

Another technique to create a copy of an existing table is to execute the previous SQL statements without the TEMPORARY keyword, as shown here:

```
MySQL [mytools]> DROP TABLE IF EXISTS temp_cust2;
Query OK, 0 rows affected, 1 warning (0.008 sec)

MySQL [mytools]> CREATE TABLE IF NOT EXISTS temp_cust2
                AS (SELECT * FROM customers);
Query OK, 1 row affected (0.028 sec)
Records: 1  Duplicates: 0  Warnings: 0

MySQL [mytools]> SELECT COUNT(*) FROM temp_cust2;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.009 sec)
```

WHAT IS AN SQL INDEX?

An *index* is a construct that enables faster retrieval of records from database tables and therefore improve performance. An index contains an entry that corresponds to each row in a table, and the index itself is stored in a tree-like structure. SQL enables you to define one or more indexes for a table, and some guidelines are provided in a subsequent section.

By way of analogy, the index of a book enables you to search for a word or a term, locate the associated page number(s), and then you can navigate to one of those pages. Clearly the use of the book index is much faster than looking sequentially through every page in a book.

Types of Indexes

A *unique index* prevents duplicate values in a column, provided that the column is also uniquely indexed, which can be performed automatically if a table has a primary key.

A *clustered index* actually changes the order of the rows in a table, and then performs a search that is based in the key values. A table can have only one clustered index. A clustered index is useful for optimizing DML statements for tables that use the InnoDB engine.

MySQL 8 introduced *invisible indexes* that are unavailable for the query optimizer. MySQL ensures that those indexes are kept current when data in the referenced column are modified. You can make indexes invisible by explicitly declare their visibility during table creation or via the ALTER TABLE command, as you will see in a later section.

Creating an Index

An index on a MySQL table can be defined in two convenient ways:

- as part of the table definition during table creation
- after table has been created

Here is an example of creating an index on the full_name attribute *during* the creation of the table friend_table:

```
DROP TABLE IF EXISTS friend_table;

CREATE TABLE friend_table  (
  friend_id int(8) NOT NULL AUTO_INCREMENT,
  full_name varchar(40) NOT NULL,
  fname varchar(20) NOT NULL,
  lname varchar(20) NOT NULL,
  PRIMARY KEY (friend_id),INDEX(full_name)
);
```

Here is an example of creating index friend_lname_idx on the lname attribute *after* the creation of the table friend_table:

```
CREATE INDEX friend_lname_idx ON friend_table(lname);
Query OK, 0 rows affected (0.035 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can create an index on multiple columns, an example of which is shown here:

```
CREATE INDEX friend_lname_fname_idx ON friend_table(lname,fname);
```

Keep in mind that an index on a MySQL table can specify a maximum of 16 indexed columns, and a table can contain a maximum of 64 secondary indexes.

Disabling and Enabling an Index

As you will learn shortly, sometimes it is useful to disable indexes, perform some intensive operation, and then re-enable the indexes. The syntax for disabling an index is here:

```
alter table friend_table disable keys;
Query OK, 0 rows affected, 1 warning (0.004 sec)
```

The corresponding syntax for re-enabling an index is here:

```
alter table friend_table enable keys;
Query OK, 0 rows affected, 1 warning (0.002 sec)
```

View and Drop Indexes

As you probably guessed, you can drop specific indexes as well as display the indexes associated with a given table and also drop specific indexes. The following SQL statement drops the specified index on the friend_table table:

```
DROP INDEX friend_lname_fname_idx ON friend_table;
Query OK, 0 rows affected (0.011 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Invoke the preceding SQL statement again, and the following error message confirms that the index was dropped:

```
ERROR 1091 (42000): Can't DROP 'friend_lname_fname_idx';
check that column/key exists
```

You can also issue the following SQL statement to display the indexes that exist on the table friend_table:

```
SHOW INDEXES FROM friend_table;
+--------------+------------+------------------+-------------
+-------------+-----------+-------------+----------+--------+--
----+------------+---------+---------------+---------+--------
----+
| Table        | Non_unique | Key_name         | Seq_in_index
| Column_name | Collation | Cardinality | Sub_part | Packed
| Null | Index_type | Comment | Index_comment | Visible |
Expression |
+--------------+------------+------------------+-------------
+-------------+-----------+-------------+----------+--------+--
----+------------+---------+---------------+---------+--------
----+
| friend_table |          0 | PRIMARY          |            1
| friend_id   | A         |           0 |     NULL |   NULL |
| BTREE      |         |               | YES     | NULL       |
| friend_table |          1 | full_name        |            1
| full_name   | A         |           0 |     NULL |   NULL |
| BTREE      |         |               | YES     | NULL       |
| friend_table |          1 | friend_lname_idx |            1
| lname       | A         |           0 |     NULL |   NULL |
| BTREE      |         |               | YES     | NULL       |
+--------------+------------+------------------+-------------
+-------------+-----------+-------------+----------+--------+--
----+------------+---------+---------------+---------+--------
----+
3 rows in set (0.005 sec)
```

When you define a MySQL table, you can specify that an index is invisible with the following code snippet:

```
INDEX phone(phone) INVISIBLE
```

The following SQL statement displays the invisible indexes in MySQL, which is a new feature in version 8:

```
SHOW INDEXES FROM friend_table
WHERE VISIBLE = 'NO';
Empty set (0.003 sec)
```

Overhead of Indexes

An index occupies some memory on secondary storage. In general, if you issue a SQL statement that involves an index, that index is first loaded into memory and then it is utilized to access the appropriate record(s). A SQL query that involves simply accessing (reading) data via an index is almost always more efficient than accessing data without an index.

However, if a SQL statement *updates* records in one or more tables, then *all* the affected indexes must be updated. As a result, there can be a performance impact when multiple indexes are updated as a result of updating table data. Hence, it is important to determine a suitable number of indexes, and the columns in each of those indexes, which can be done either by experimentation (not recommended for beginners) or open source tools that provide statistics regarding the performance of SQL statements when indexes are involved.

Considerations for Defining Indexes

As you might already know, a full table scan for large tables will likely be computationally expensive. Therefore, make sure that you define an index on columns that appear in the WHERE clause in your SQL statements. As a simple example, consider the following SQL statement:

```
SELECT *
FROM customers
WHERE lname = 'Smith';
```

If you do not have an index that includes the lname attribute of the customers table, then a full table scan is executed.

Consider defining an index on attributes that appear in query statements that involve SELECT, GROUP BY, ORDER BY, or JOIN. As mentioned earlier, updates to table data necessitate updates to indexes, which in turn can result in lower performance.

A suggestion before inserting a large volume of data into a table (or tables):

1. Disable the indexes.
2. Insert the data.
3. Reactivate the indexes.

Although the preceding approach involves rebuilding the indexes, which is performed after step 3, you might see a performance improvement compared to directly inserting the table data. Of course, you could also try both approaches and calculate the time required to complete the data insertion.

As yet another option, it is possible to perform a multirow insert in MySQL, which enables you to insert several rows with a single SQL statement, thereby reducing the number of times the indexes must be updated. The maximum number of rows that can be inserted via a multirow insert depends on the value of max_allowed_packet (whose default value is 4M), as described here:

https://dev.mysql.com/doc/refman/5.7/en/packet-too-large.html

Another suggestion: check the order of the columns in multicolumn indexes and compare that order with the order of the columns in each index. MySQL will only use an index if the left-leading column is referenced.

Selecting Columns for an Index

An index of a database table is used if the attribute in the WHERE clause is included in the index. For example, the following SQL query specifies the lname attribute of the users table in the WHERE clause:

```
SELECT *
FROM users
WHERE lname = 'SMITH'
```

In the previous section, you learned that the users table does not have an index containing the lname attribute then a full table scan is executed and the contents of the lname attribute in every row is compared with SMITH.

Keep in mind that the *average* number of comparisons in a full table scan is n/2, where n is the number of rows in the given table. Thus, a table containing 1,024 rows (which is a very modest size) would require an average of 512 comparisons, whereas a suitably defined index would reduce the average number of comparisons to 10 (and sometimes even fewer comparisons).

Based on the preceding paragraph, indexes can be useful for improving the performance of read operations. In general, the candidates for inclusion in the definition of an index are the attributes that appear in frequently invoked SQL statements that select, join, group, or order data. However, keep in mind that the space requirement for indexes is related to the number of rows in tables.

Hence, keep in mind that too many indexes involve more memory, and they must be updated after a write operation, which can incur a performance penalty. An experienced DBA can provide you with helpful advice. Experiment with the number and type of indexes, and profile your system in order to determine the optimal combination for your system. In addition, you can download various SQL monitoring tools to determine which SQL operations are candidates for optimization.

Finding Columns Included in Indexes

Please keep in mind that this section contains SQL statements that are specific to MySQL: for information about other databases (such as Oracle) perform an online search to find the correct syntax. MySQL enables you to find columns that are in indexes with this SQL statement:

```
SHOW INDEX FROM people;
```

You can also query the STATISTICS table in the INFORMATION_SCHEMA to show indexes for all tables in a schema, an example of which is shown here:

```
SELECT DISTINCT TABLE_NAME, INDEX_NAME
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = 'mytools';
```

EXPORT DATA FROM MYSQL

In Chapter 4, you learned how to import data from CSV files into MySQL tables, and in this section you will learn how to export data from MySQL to files of various formats. After you create a SQL query that produces the desired data set, you can launch the SQL query from the command line or from inside a tool such as SQL Workbench that is available as a free download. Moreover, there are two main options for exporting data, as discussed in the following subsections.

Export the Result Set of a SQL Query

Any SQL query that generates a result set can be exported. For example, a SQL query that select all rows (or some rows) from a single table or a SQL query that is based on the join of two tables can be exported. The format of a file that contains a result set includes CSV, TSV, XML, HTML, JSON, and Excel. As you learned earlier in this chapter, you can save the output of a SQL statement from inside SQL Workbench by clicking on the "Export" icon, and then follow the subsequent prompts to save the result set to a file with the desired format.

Export a Database or Its Contents

This option for exporting a database involves three options:

1. Export all the data in all the database tables and save that data in a file.
2. Export only the structure of the database tables and indexes.
3. Export the data and the database structure.

Option #1 serves as a database backup, which you can use to restore the entire database. Keep in mind that you can lose the most recent data in some tables, which depends on when the most recent backup was performed. The backup enables you to restore the database to a state in which most of the earlier data is available.

Option #2 is useful when you want to re-create the structure of a database, or some of its tables, in another database that can be on a different machine or even for a different person.

Option #3 has combines the benefits of the first two options. Moreover, you can either create one file for everything, or create one file for each table. Keep in mind that if you want to restore only one table from a very large backup file, this option can be much more time consuming than restoring a table from a SQL statement that contains everything pertaining to that table.

For example, the following command exports everything in the mytools database:

```
mysqldump -u root -p mytools > mytools.sql
```

Again, SQL Workbench provides a GUI interface for exporting and importing databases, tables, and CSV files, without the need to remember the syntax for the associated commands.

USING LOAD DATA IN MYSQL

MySQL enables you to load data from a CSV file into a database table using the following syntax:

```
LOAD DATA INFILE 'path of the file/file_name.txt'
INTO TABLE employees
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\r\n'
(id,name,sex,designation,dob)
```

The preceding command specifies a semi-colon as the delimiter separating the values in each row.

LOAD DATA INFILE is relevant when you need to load a large amount of data into a database table. The first step involves creating a table (if it does not already exist) that corresponds to the columns in a CSV file, and then we can invoke a suitable SQL statement to populate that table from a CSV file.

DATA CLEANING IN SQL

This section contains several subsections that perform data cleaning tasks in SQL. Note that it is not mandatory to perform these tasks in SQL: another option is to read the contents of a database table into a Pandas data frame and then use Pandas methods to achieve the same result.

However, this section illustrates how to perform the following data cleaning tasks that affect an attribute of a database table:

- Replace NULL with 0.
- Replace NULL with the average value.
- Replace multiple values into a single value.

- Handle data type mismatch.
- Convert a string date to a date format.

Replace NULL With 0

This task is very straightforward, which you can perform with either of the following SQL statements:

```
SELECT ISNULL(column_name, 0 ) FROM table_name
OR
SELECT COALESCE(column_name, 0 ) FROM table_name
```

Replace NULL Values With Average Value

This task involves two steps: first find the average of the non-NULL values of a column in a database table, and then update the NULL values in that column with the value that you found in the first step.

Listing 5.16 displays the contents of replace_null_values.sql that performs this pair of steps.

LISTING 5.16: replace_null_values.sql

```
USE mytools;
DROP TABLE IF EXISTS temperatures;
CREATE TABLE temperatures (temper INT, city CHAR(20));

INSERT INTO temperatures VALUES(78,'sf');
INSERT INTO temperatures VALUES(NULL,'sf');
INSERT INTO temperatures VALUES(42,NULL);
INSERT INTO temperatures VALUES(NULL,'ny');
SELECT * FROM temperatures;

SELECT @avg1 := AVG(temper) FROM temperatures;
update temperatures
set temper = @avg1
where ISNULL(temper);
SELECT * FROM temperatures;

-- initialize city1 with the most frequent city value:
SELECT @city1 := (SELECT city FROM temperatures GROUP BY
city ORDER BY COUNT(*) DESC LIMIT 1);

-- update NULL city values with the value of city1:
update temperatures
set city = @city1
where ISNULL(city);
SELECT * FROM temperatures;
```

Listing 5.16 creates and populates the table temperatures with several rows and then initializes the variable avg1 with the average temperature in the temper attribute of the temperatures table. Now launch the code in Listing 5.16 and you will see the following output:

```
+--------+------+
| temper | city |
+--------+------+
|     78 | sf   |
|   NULL | sf   |
|     42 | NULL |
|   NULL | ny   |
+--------+------+
4 rows in set (0.000 sec)

+---------------------+
| @avg1 := AVG(temper) |
+---------------------+
|         60.000000000 |
+---------------------+
1 row in set, 1 warning (0.000 sec)

Query OK, 2 rows affected (0.001 sec)
Rows matched: 2  Changed: 2  Warnings: 0

+--------+------+
| temper | city |
+--------+------+
|     78 | sf   |
|     60 | sf   |
|     42 | NULL |
|     60 | ny   |
+--------+------+
4 rows in set (0.000 sec)

+-----------------------------------------------------------
----------+
| @city1 := (SELECT city FROM temperatures GROUP BY city
ORDER BY COUNT(*) DESC LIMIT 1) |
+-----------------------------------------------------------
----------+
| sf
|
+-----------------------------------------------------------
----------+
1 row in set, 1 warning (0.000 sec)

Query OK, 1 row affected (0.000 sec)
Rows matched: 1  Changed: 1  Warnings: 0

+--------+------+
| temper | city |
+--------+------+
|     78 | sf   |
|     60 | sf   |
|     42 | sf   |
|     60 | ny   |
+--------+------+
4 rows in set (0.000 sec)
```

Replace Multiple Values With a Single Value

An example of coalescing multiple values in an attribute involves replacing multiple strings for the state of New York (such as `new_york`, `NewYork`, `NY`, `New_York`, and so forth) with `NY`. Listing 5.17 displays the contents of `reduce_values.sql` that performs this pair of steps.

LISTING 5.17: reduce_values.sql

```
use mytools;

DROP TABLE IF EXISTS mytable;
CREATE TABLE mytable (str_date CHAR(15), state CHAR(20),
reply CHAR(10));

INSERT INTO mytable VALUES('20210915','New York','Yes');
INSERT INTO mytable VALUES('20211016','New York','no');
INSERT INTO mytable VALUES('20220117','Illinois','yes');
INSERT INTO mytable VALUES('20220218','New York','No');
SELECT * FROM mytable;

-- replace yes, Yes, y, Ys with Y:
update mytable
set reply = 'Y'
where upper(substr(reply,1,1)) = 'Y';
SELECT * FROM mytable;

-- replace all other values with
update mytable
set reply = 'N' where substr(reply,1,1) != 'Y';
SELECT * FROM mytable;
```

Listing 5.17 creates and populates the table `mytable`, and then replaces the variants of the word "yes" with the letter `Y` in the reply attribute. The final portion of Listing 5.17 replaces any string that does *not* start with the letter `Y` with the letter `N`. Launch the code in Listing 5.17 and you will see the following output:

```
+----------+----------+-------+
| str_date | state    | reply |
+----------+----------+-------+
| 20210915 | New York | Yes   |
| 20211016 | New York | no    |
| 20220117 | Illinois | yes   |
| 20220218 | New York | No    |
+----------+----------+-------+
4 rows in set (0.000 sec)

Query OK, 2 rows affected (0.001 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

```
+----------+---------+-------+
| str_date | state    | reply |
+----------+---------+-------+
| 20210915 | New York | Y     |
| 20211016 | New York | no    |
| 20220117 | Illinois | Y     |
| 20220218 | New York | No    |
+----------+---------+-------+
4 rows in set (0.000 sec)

Query OK, 2 rows affected (0.001 sec)
Rows matched: 2  Changed: 2  Warnings: 0

+----------+---------+-------+
| str_date | state    | reply |
+----------+---------+-------+
| 20210915 | New York | Y     |
| 20211016 | New York | N     |
| 20220117 | Illinois | Y     |
| 20220218 | New York | N     |
+----------+---------+-------+
4 rows in set (0.001 sec)
```

Handle Mismatched Attribute Values

This task involves two steps: first find the average of the non-NULL values of a column in a database table, and then update the NULL values in that column with the value that you found in the first step.

Listing 5.18 displays the contents of type_mismatch.sql that performs this pair of steps.

LISTING 5.18: type_mismatch.sql

```
USE mytools;
DROP TABLE IF EXISTS emp_details;
CREATE TABLE emp_details (emp_id CHAR(15), city CHAR(20), state
CHAR(20));

INSERT INTO emp_details VALUES('1000','Chicago','Illinois');
INSERT INTO emp_details VALUES('2000','Seattle','Washington');
INSERT INTO emp_details VALUES('3000','Santa Cruz','California');
INSERT INTO emp_details VALUES('4000','Boston','Massachusetts');
SELECT * FROM emp_details;

select emp.emp_id, emp.title, det.city, det.state
from employees emp join emp_details det
WHERE emp.emp_id = det.emp_id;

--required for earlier versions of MySQL:
--WHERE emp.emp_id = cast(det.emp_id as INT);
```

Listing 5.18 creates and populates the table emp_details, followed by a SQL JOIN statement involving the tables emp and emp_details. Although the emp_id attribute is defined as an INT type and a CHAR type, respectively, in the tables `emp` and `emp_details`, the code works as desired. However, in earlier versions of `MySQL`, you need to use the built-in `CAST()` function in order to convert a `CHAR` value to an `INT` value (or vice versa), as shown in the commented out code snippet:

```
--WHERE emp.emp_id = cast(det.emp_id as INT);
```

Now launch the code in Listing 5.18 and you will see the following output:

```
+--------+------------+---------------+
| emp_id | city       | state         |
+--------+------------+---------------+
| 1000   | Chicago    | Illinois      |
| 2000   | Seattle    | Washington    |
| 3000   | Santa Cruz | California    |
| 4000   | Boston     | Massachusetts |
+--------+------------+---------------+
4 rows in set (0.000 sec)
+--------+--------------------+------------+---------------+
| emp_id | title              | city       | state         |
+--------+--------------------+------------+---------------+
|   1000 | Developer          | Chicago    | Illinois      |
|   2000 | Project Lead       | Seattle    | Washington    |
|   3000 | Dev Manager        | Santa Cruz | California    |
|   4000 | Senior Dev Manager | Boston     | Massachusetts |
+--------+--------------------+------------+---------------+
4 rows in set (0.002 sec)
```

Convert Strings to Date Values

Listing 5.19 displays the contents of `str_to_date.sql` that illustrates how to populate a date attribute with date values that are determined from another string-based attribute that contains strings for dates.

LISTING 5.19: str_to_date.sql

```
use mytools;

DROP TABLE IF EXISTS mytable;
CREATE TABLE mytable (str_date CHAR(15), state CHAR(20),
reply CHAR(10));

INSERT INTO mytable VALUES('20210915','New York','Yes');
INSERT INTO mytable VALUES('20211016','New York','no');); 
INSERT INTO mytable VALUES('20220117','Illinois','yes');); 
INSERT INTO mytable VALUES('20220218','New York','No');); 

SELECT * FROM mytable;

-- 1) insert date-based feature:
ALTER TABLE mytable
ADD COLUMN (real_date DATE);
SELECT * FROM mytable;
```

```
-- 2) populate real_date from str_date:
UPDATE mytable t1
        INNER JOIN mytable t2
             ON t1.str_date = t2.str_date
SET t1.real_date = DATE(t2.str_date);
SELECT * FROM mytable;

-- 3) Remove unwanted features:
ALTER TABLE mytable
DROP COLUMN str_date;
SELECT * FROM mytable;
```

Listing 5.19 creates and populates the table `mytable` and displays the contents of this table. The remainder of Listing 5.19 consists of three `SQL` statements, each of which starts with a comment statement that explains its purpose.

The first `SQL` statement inserts a new column `real_date` of type `DATE`. The second `SQL` statement populates the `real_date` column with the values in the `str_date` column that have been converted to a date value via the `DATE()` function. The third `SQL` statement is optional: it drops the `str_date` column if you wish to do so. Now launch the code in Listing 5.19 and you will see the following output:

```
+----------+----------+-------+
| str_date | state    | reply |
+----------+----------+-------+
| 20210915 | New York | Yes   |
| 20211016 | New York | no    |
| 20220117 | Illinois | yes   |
| 20220218 | New York | No    |
+----------+----------+-------+
4 rows in set (0.000 sec)

Query OK, 0 rows affected (0.007 sec)
Records: 0  Duplicates: 0  Warnings: 0

+----------+----------+-------+-----------+
| str_date | state    | reply | real_date |
+----------+----------+-------+-----------+
| 20210915 | New York | Yes   | NULL      |
| 20211016 | New York | no    | NULL      |
| 20220117 | Illinois | yes   | NULL      |
| 20220218 | New York | No    | NULL      |
+----------+----------+-------+-----------+
4 rows in set (0.002 sec)

Query OK, 4 rows affected (0.002 sec)
Rows matched: 4  Changed: 4  Warnings: 0

+----------+----------+-------+------------+
| str_date | state    | reply | real_date  |
+----------+----------+-------+------------+
| 20210915 | New York | Yes   | 2021-09-15 |
| 20211016 | New York | no    | 2021-10-16 |
| 20220117 | Illinois | yes   | 2022-01-17 |
| 20220218 | New York | No    | 2022-02-18 |
+----------+----------+-------+------------+
4 rows in set (0.000 sec)

Query OK, 0 rows affected (0.018 sec)
Records: 0  Duplicates: 0  Warnings: 0

+----------+-------+------------+
| state    | reply | real_date  |
+----------+-------+------------+
| New York | Yes   | 2021-09-15 |
| New York | no    | 2021-10-16 |
| Illinois | yes   | 2022-01-17 |
| New York | No    | 2022-02-18 |
+----------+-------+------------+
4 rows in set (0.000 sec)
```

DATA CLEANING FROM THE COMMAND LINE (OPTIONAL)

This section is marked "optional" because the solutions to tasks involve an understanding of some Unix-based utilities. Although this book does not contain details about those utilities, you can find online tutorials with examples regarding these utilities.

This section contains several subsections that perform data cleaning tasks that involve the "sed" and "awk" utilities:

- Replace multiple delimiters with a single delimiter (sed).
- Restructure a dataset so all rows have the same column count (awk).

Keep in mind the following point about these examples: they must be performed from the command line before they can be processed in a `Pandas` data frame.

Working With the sed Utility

This section contains an example of how to use the `sed` command line utility to replace different delimiters with a single delimiter for the fields in a text file. You can use the same code for other file formats, such as `CSV` files and `TSV` files.

Keep in mind that this section does not provide any details about `sed` beyond the code sample in this section. However, after you read the code—it is a one-liner—you will understand how to adapt that code snippet to your own requirements (i.e., how to specify different delimiters).

Listing 5.20 displays the contents of `delimiter1.txt` and Listing 5.21 displays the contents of `delimiter1.sh` that replaces all delimiters with a comma (",").

LISTING 5.20: delimiter1.txt

```
1000|Jane:Edwards^Sales
2000|Tom:Smith^Development
3000|Dave:Del Ray^Marketing
```

LISTING 5.21: delimiter1.sh

```
cat delimiter1.txt | sed -e 's/:/,/' -e 's/|/,/' -e 's/\^/,/'
```

Listing 5.26 starts with the `cat` command line utility, which sends the contents of the file `delimiter1.txt` "standard output," which is the screen (by default). However, in this example the output of this command becomes the input to the sed command because of the pipe ("|") symbol.

The `sed` command consists of three parts, all of which are connected by the "-e" switch. You can think of "-e" as indicating "there is more processing to be done" by the `sed` command. In this example, there are three occurrences of "-e," which means that the `sed` command will be invoked three times.

The first code snippet is `'s/:/,/'`, which translates into "replace each semi-colon with a comma." The result of this operation is passed to the next code snippet, which is `'s/|/,/'`. This code snippet translates into "replace each pipe symbol with a comma." The result of this operation is passed to the next code snippet, which is `'s/\^/,/'`. This code snippet translates into "replace each caret symbol ("^") with a comma." The result of this operation is sent to standard output, which can be redirected to another text file. Launch the code in Listing 5.27 and you will see the following output:

```
1000,Jane,Edwards,Sales
2000,Tom,Smith,Development
3000,Dave,Del Ray,Marketing
```

There are three things to keep in mind. First, the snippet contains a backslash because the caret symbol ("^") is a meta character, so we need to "escape" this character. The same is true for other meta characters, such as "$" and ".".

Second, you can easily extend the `sed` command for each new delimiter that you encounter as a field separator in a text file: simply follow the pattern that you see in Listing 5.21.

Third, launch the following command to redirect the output of `delimiter1.sh` to the text file `delimiter2.txt`:

```
./delimiter1.sh > delimiter2.txt
```

If an error occurs in the preceding code snippet, make sure that `delimiter1.sh` is executable by invoking the following command:

```
chmod 755 delimiter1.sh
```

This concludes the example involving the `sed` command line utility, which is a very powerful utility for processing text files. Check online for articles and blog posts if you want to learn more about the `sed` utility.

Working With the awk Utility

The awk command line utility is a self-contained programming language, with a truly impressive capability for processing text files. However, this section does not provide details about awk beyond the code sample. If you are interested, there are plenty of online articles that provide in-depth explanations regarding the awk utility.

Listing 5.22 displays the contents FixedFieldCount1.sh that illustrates how to use the awk utility in order to split a string into rows that contain three strings.

LISTING 5.22: FixedFieldCount1.sh

```
echo "=> pairs of letters:"
echo "aa bb cc dd ee ff gg hh"
echo

echo "=> split on multiple lines:"
echo "aa bb cc dd ee ff gg hh"| awk '
BEGIN { colCount = 3 }
{
  for(i=1; i<=NF; i++) {
     printf("%s ", $i)
     if(i % colCount == 0) { print "" }
  }
  print ""
}
'
```

Listing 5.22 displays the contents of a string, and then provides this string as input to the awk command. The main body of Listing 5.22 is a loop that iterates from 1 to NF, where NF is the number of fields in the input line, which in this example equals 8. The value of each field is represented by $i: $1 is the first field, $2 is the second field, and so forth. Note that $0 is the contents of the *entire* input line (which is used in the next code sample).

Next, if the value of i (which is the field *position*, not the contents of the field) is a multiple of 3, then the code prints a linefeed. Launch the code in Listing 5.22 and you will see the following output:

```
=> pairs of letters:
aa bb cc dd ee ff gg hh

=> split on multiple lines:
aa bb cc
dd ee ff
gg hh
```

Listing 5.23 displays the contents of employees.txt and Listing 5.24 displays the contents of FixedFieldCount2.sh that illustrates how to use the awk utility in order to ensure that all the rows have the same number of columns.

LISTING 5.23: employees.txt

```
jane:jones:SF:
john:smith:LA:
dave:smith:NY:
sara:white:CHI:
>>>none:none:none<<<:
jane:jones:SF:john:
smith:LA:
dave:smith:NY:sara:white:
CHI:
```

LISTING 5.24: FixedFieldCount2.sh

```
cat employees.txt | awk -F":" '{printf("%s", $0)}' | awk -F':' '
BEGIN { colCount = 3 }
{
  for(i=1; i<=NF; i++) {
     printf("%s#", $i)
     if(i % colCount == 0) { print "" }
  }
}
'
```

Notice that the code in Listing 5.24 is almost identical to the code in Listing 5.28: the new code snippet that is shown in bold removes the "\n" character from its input that consists of the contents of `employees.txt`. In case you need to be convinced, launch the following code snippet from the command line:

```
cat employees.txt | awk -F":" '{printf("%s", $0)}'
```

The output of the preceding code snippet is shown here:

```
jane:jones:SF:john:smith:LA:dave:smith:NY:sara:white:CHI:>>
>none:none:none<<<:jane:jones:SF:john:smith:LA:dave:smith:N
Y:sara:white:CHI:
```

The reason that the "\n" has been removed in the preceding output is because of this code snippet:

```
printf("%s", $0)
```

If you want to retain the "\n" character after each input line, then replace the preceding code snippet with this snippet:

```
printf("%s\n", $0)
```

We have now reduced the task in Listing 5.24 to the same task as Listing 5.22, which is why we have the same awk-based inner code block.

Now launch the code in Listing 5.24 and you will see the following output:

```
jane#jones#SF#
john#smith#LA#
dave#smith#NY#
sara#white#CHI#
>>>none#none#none<<<#
jane#jones#SF#
john#smith#LA#
dave#smith#NY#
sara#white#CHI#
```

SUMMARY

This chapter introduced you to `SQL` and how to invoke various types of `SQL` statements. You saw how to create tables manually from the `SQL` prompt and also by launching a `SQL` script that contains `SQL` statements for creating tables.

You learned how to drop tables, along with the effect of the `DELETE`, `TRUNCATE`, and `DROP` keywords in `SQL` statements. Next, you learned how to invoke a `SQL` statement to dynamically create a new table based on the structure of an existing table.

Then you saw an assortment of `SQL` statements that use the `SELECT` keyword. Examples of such `SQL` statements include finding the distinct rows in a `MySQL` table, finding unique rows, along with using the `EXISTS` and `LIMIT` keywords. Moreover, you learned about the differences among the `DELETE`, `TRUNCATE`, and `DROP` keywords in `SQL`.

Next, you saw how to create indexes on `MySQL` tables, and some criteria for defining indexes, followed by how to select columns for an index.

Finally, you learned how to use the `sed` command line utility to replace multiple field delimiters with the same delimiter, followed by the `awk` command utility to generate a text file in which every row has the same number of columns.

CHAPTER 6

NLP AND DATA CLEANING

This chapter contains `Python` code samples for some `NLP`-related concepts, along with an assortment of code samples involving regular expressions, the `Python` library `BeautifulSoup`, `Scrapy`, and various `NLP`-related Python code samples that use the `SpaCy` library.

The first part of this chapter describes some common NLP tasks and NLP techniques, such as stemming and lemmatization. The second section discusses POS (parts of speech) in text as well as POS tagging techniques.

The third section contains examples of cleaning text via regular expression (REs), and also how to handle contractions of words. In addition, you will learn about `BeautifulSoup`, which is a `Python` module for scraping HTML Web pages. This section contains some `Python` code samples that retrieve and then manipulate the contents of an HTML Web page from the GitHub code repository. This section also contains a brief introduction to `Scrapy`, which is a `Python`-based library that provides Web scraping functionality and various other APIs.

NLP TASKS IN ML

Since there are many types of `NLP` tasks, there are also many `NLP` techniques that have been developed, some of which are listed here:

text embeddings
text summarization
text classification
sentence segmentation
POS (part-of-speech tagging)
NER (named entity recognition)
word sense disambiguation

text categorization
topic modeling
text similarity
syntax and parsing
language modeling
dialogs
probabilistic parsing
clustering

In this chapter, we'll focus on `NLP` preprocessing steps for training a language model, some of which are discussed in the following subsections.

NLP Steps for Training a Model

Although the specific set of text-related tasks depends on the specific task that you are trying to complete, the following set of steps is common:

[1] convert words to lowercase
[1] noise removal
[2] normalization
[3] text enrichment
[3] stop word removal
[3] stemming
[3] lemmatization

The number in brackets in the preceding bullet list indicates the type of task. Specifically, the values [1], [2], and [3] indicate "must do," "should do," and "task dependent," respectively.

TEXT NORMALIZATION AND TOKENIZATION

Text normalization involves several tasks, such as the removal of unwanted hash tags, emojis, URLs, special characters such as "&," "!," "$," and so forth. However, you might need to make decisions regarding some punctuation marks.

First, what about the period (".") punctuation mark? If you retain every period (".") in a dataset, consider whether or not to treat this character as a token during the tokenization step. However, if you remove every period (".") from a dataset, this will also remove every ellipsis (three consecutive periods), and also the period from the strings "Mr.," "U.S.A.," "P.O.," and so forth. If the dataset is small, perform a visual inspection of the dataset. If the dataset is very large, try inspecting several smaller and randomly selected subsets of the original dataset.

Second, although you might think it is a good idea to remove question marks ("?"), the opposite is true: in general, question marks enable you to identify questions (as opposed to statements) in a corpus.

Third, you also need to determine whether or not to remove numbers, which can convey quantity when they are separate tokens ("1,000 barrels of oil") or they can be data entry errors when they are embedded in alphabetic strings. For example, it is probably okay to remove the 99 from the string "large99 oranges," but what about the 99 in "99large oranges"?

Another standard normalization task involves converting all words to lowercase ("case folding"). Chinese characters do not have uppercase text, so converting text to lowercase is unnecessary. Keep in mind that text normalization is entirely unrelated to normalizing database tables in an RDBMS, or normalizing (scaling) numeric data in machine learning tasks. The task of converting categorical (character) data into a numeric counterpart.

Although "case folding" is a straightforward task, this step can be problematic. For instance, accents are optional for uppercase French words, and after case folding some words do require an accent. A simple example is the French word *peche*, which means *fish* or *peach* with one accent mark, and *sin* with a different accent mark. The Italian counterparts are *pesce*, *pesca*, and *peccato*, respectively, and there is no issue regarding accents marks. Incidentally, the plural of *pesce* is *pesci* (so *Joe Pesci* is *Joe Fish or Joe Fishes*, depending on whether you are referring to one type of fish or multiple types of fish). To a lesser extent, converting English words from uppercase to lowercase can cause issues: is the word "stone" from the noun "stone" or from the surname "Stone"?

After normalizing a dataset, tokenization involves "splitting" a sentence, paragraph, or document into its individual words (tokens). The complexity of this task can vary significantly between languages, depending on the nature of the alphabet of a specific language. In particular, tokenization is straightforward for Indo-European languages because those languages use a space character to separate words.

However, although tokenization can be straightforward when working with regular text, the process can be more challenging when working with biomedical data that contains acronyms and a higher frequency use of punctuation. One `NLP` technique for handling acronyms is named entity recognition (NER), which is discussed later in this chapter.

Word Tokenization in Japanese

Unlike most languages, the use of a space character in Japanese text is optional. Another complicating factor is the existence of multiple alphabets in Japanese, and sentences often contain a mixture of these alphabets. Specifically, Japanese supports Romaji (essentially the English alphabet), Hiragana, Katakana (used exclusively for words imported to Japanese from other languages), and Kanji characters.

As a simple example, navigate to Google translate in your browser and enter the following sentence, which in English means "I gave a book to my friend":

```
watashiwatomodachinihonoagemashita
```

The translation (which is almost correct) is the following text in Hiragana:

わたしはこれだけのほげあげました

Now enter the same sentence, but with spaces between each word, as shown here:

```
watashi wa tomodachi ni hon o agemashita
```

Now Google translate produces the following correct translation in Hiragana:

私はともだちに本をあげました

The preceding sentence starts with the Kanji character 私 that is the correct translation for "watashi."

Mandarin and Cantonese are two more languages that involves complicated tokenization. Both of these languages are tonal, and they use pictographs instead of an alphabets. Mandarin provides Pinyin, which is the romanization of the sounds in Mandarin, along with 4 digits to indicate the specific tone for each syllable (the neutral sound does not have a tone). Mandarin has 6 tones, of which 4 are commonly used, whereas Cantonese has 9 tones (but does not have a counterpart to Pinyin).

As a simple example, the following sentences are in Mandarin and in Pinyin, respectively, and their translation into English is "How many children do you have":

- 你有几个孩子
- `Nǐ yǒu jǐ gè háizi`
- `Ni3 you3 ji3ge4 hai2zi (digits instead of tone marks)`

The second and third sentences in the preceding group are both Pinyin. The third sentence contains the numbers 2, 3, and 4 that correspond to the second, third, and fourth tones, respectively, in Mandarin. The third sentence is used in situations where the tonal characters are not supported (such as older browsers). Navigate to Google Translate and type the following words for the source language:

```
ni you jige haizi
```

Select Mandarin for the target language and you will see the following translation:

```
how many kids do you have
```

The preceding translation is quite impressive, when you consider that the tones were omitted, which can significantly change the meaning of words. If you are skeptical, look at the translation of the string "ma" when it is written with the first tone, then the second tone, and again with the third tone and the fourth tone: the meanings of these our words are entirely unrelated.

Tokenization can be performed via regular expressions (which are discussed in one of the appendices) and rule-based tokenization. However, rule-based tokenizers are not well-equipped to handle rare words or compound words that are very common in German.

Text Tokenization With Unix Commands

Text tokenization can be performed not only in `Python` but also from the Unix command line. For example, consider the text file `words.txt` whose contents are shown here:

```
lemmatization: removing word endings edit distance: measure
the distance between two words based on the number of
changes needed based on the inner product of 2 vectors a
metric for determining word similarity
```

The following command illustrates how to tokenize the preceding paragraph using several Unix commands that are connected via the Unix pipe ("|") symbol:

```
tr -sc 'A-Za-z' '\n' < words.txt | sort | uniq
```

The output from the preceding command is shown here:

```
1a
2 based
1 between
1 changes
1 determining
2 distance
1 edit
1 endings
1for
1 inner
1 lemmatization
// text omitted for brevity
```

As you can see, the preceding output is an alphabetical listing of the tokens of the contents of the text file `words.txt`, along with the frequency of each token.

HANDLING STOP WORDS

Stop words are words that are considered unimportant in a sentence. Although the omission of such words would result in grammatically incorrect sentences, the meaning of such sentences would most likely still be recognizable.

In English, stop words include the words "a," "an," and "the," along with common words and prepositions ("inside," "outside," and so forth). Stop words are usually filtered from search queries because they would return a vast amount of unnecessary information. As you will see later, `Python` libraries such as `NLTK` provide a list of built-in stop words, and you can supplement that list of words with your own list.

Removing stop words works fine with `BoW` and `tf-idf`, both of which are discussed in Chapter 7, but they can adversely models that use word context to detect semantic meaning. A more detailed explanation (and an example) is here:

https://towardsdatascience.com/why-you-should-avoid-removing-stop-words-aa7a353d2a52

Keep in mind that a universal list of stop words does not exist, and different toolkits (`NLTK`, gensim, and so forth) have different sets of stop words. The `Sklearn` library provides a list of stop words that consists of basic words ("and," "the," "her," and so forth). However, a list of stop words for the text in a marketing-related website is probably different from such a list for a technical website. Fortunately, `Sklearn` enables you to specify your own list of stop words via the hyperparameter `stop_words`.

Finally, the following link contains a list of stop words for an impressive number of languages:

https://github.com/Alir3z4/stop-words

WHAT IS STEMMING?

This concept refers to reducing words to their root or base unit. Keep in mind that a stemmer operates on individual words without any context for those words. Stemming will "chop off" the ends of words, which means that "fast" is the stem for the words `fast`, `faster`, and `fastest`. Stemming algorithms are typically rule-based and involve conditional logic. In general, stemming is simpler than lemmatization (discussed later), and it is a special case of normalization.

Singular versus Plural Word Endings

The manner in which the plural of a word is formed varies among languages. In many cases, the letter "s" or the letters "es" represent the plural form of words in English. In some cases, English words have a singular form that ends in s/us/x (basis, abacus, and box), and a plural form with the letter "I" or "ces": cactus/cacti, appendix/appendices, and so forth.

However, German can form the plural of a noun with "er" and "en," such as buch/bucher, frau/frauen, and so on.

Common Stemmers

The following list contains several commonly used stemmers in NLP:

Porter stemmer (English)
Lancaster stemmer
SnowballStemmers (more than 10 languages)
ISRIStemmer (Arabic)
RSLPSStemmer (Portuguese)

The `Porter` stemmer was developed in the 1980s, and while it is good in a research environment, it is not recommended for production. The `Snowball` stemmer is based on the `Porter2` stemming algorithm, and it is an improved version of `Porter` (about 5% better).

The `Lancaster` stemmer is a good stemming algorithm, and you can even add custom rules to the `Lancaster` stemmer in `NLTK` (but results can be odd). The other three stemmers support non-English languages.

As a simple example, the following code snippet illustrates how to define two stemmers using the `NLTK` library:

```
import nltk
from nltk.stem import PorterStemmer, SnowballStemmer

porter = PorterStemmer()
porter.stem("Running")

snowball = SnowballStemmer("spanish", ignore_stopwords=True)
snowball.stem("Corriendo")
```

Notice that the second stemmer defined in the preceding code block also ignores stop words.

Stemmers and Word Prefixes

Word prefixes can pose interesting challenges. For example, the prefix "un" often means "not" (such as the word unknown) but not in the case of "university." One approach for handling this type of situation involves creating a word list and after removing a prefix, check if the remaining word is in the list: if not, then the prefix in the original word is not a negative. Among the few (only?) stemmers that provides prefix stemming in `NLTK` are Arabic stemmers:

https://github.com/nltk/nltk/blob/develop/nltk/stem/arlstem.py#L115

https://github.com/nltk/nltk/blob/develop/nltk/stem/snowball.py#L372

However, it is possible to write custom `Python` code to remove prefixes. First, navigate to this URL to see a list of prefixes in the English language is here:

https://dictionary.cambridge.org/grammar/british-grammar/word-formation/prefixes

https://stackoverflow.com/questions/62035756/how-to-find-the-prefix-of-a-word-for-nlp

Next, a `Python` code sample that implements a basic prefix finder is here:

https://stackoverflow.com/questions/52140526/python-nltk-stemmers-never-remove-prefixes

Over Stemming and Under Stemming

Over stemming occurs when too much of a word is truncated, which can result in unrelated words having the same stem. For example, consider the following sequence of words:

```
university, universities, universal, universe
```

The stem for the four preceding words is *universe*, even though these words have different meanings.

Under stemming is the opposite of over stemming: this happens when a word is insufficiently "trimmed." For example, the words `data` and `datu` both have the stem `dat`, but what about the word `date`? This simple example illustrates that it is difficult to create good stemming algorithms.

WHAT IS LEMMATIZATION?

Lemmatization determines whether or not words have the same root, which involves the removal of inflectional endings of words. Lemmatization involves the `WordNet` database during the process of finding the root word of each word in a corpus.

Lemmatization finds the base form of a word, such as the base word *good* for the three words good, better, and best. Lemmatization determines the dictionary form of words and therefore requires knowledge of parts of speech. In general, creating a lemmatizer is more difficult than a heuristic stemmer. The `NLTK` lemmatizer is based on the `WordNet` database.

Lemmatization is also relevant for verb tenses. For instance, the words run, runs, running, and ran are variants of the verb "run." Another example of lemmatization involves irregular verbs, such as "to be" and "to have" in romance languages. Thus, the collection of verbs is, was, were, and be are all variants of the verb "be." Keep in mind that there is a trade-off: lemmatization can produce better results than stemming at the cost of being more computationally expensive.

Stemming/Lemmatization Caveats

Both techniques are designed for "recall" whereas precision tends to suffer. Result can also differ significantly in non-English languages, even those that seem related to English, because the implementation details of some concepts are quite different.

Although both techniques generate the root form of inflected words, the stem might not be an actual word, whereas the lemma *is* an actual language word. In general, use stemming if you are primarily interested in higher speed, and use lemmatization if you are primarily interested in higher accuracy.

Limitations of Stemming and Lemmatization

Although stemming and lemmatization are suitable for Indo-European languages, these techniques are not as well-suited for Chinese because a Chinese character can be a combination of two other characters, all three of which can have different meanings.

For example, the character for mother is the combination of the radical for female and the radical for horse. Hence, separating the two radicals for mother via stemming and lemmatization change the meaning of the word from "mother" to "female." More detailed information regarding Chinese natural language processing is available here:

https://towardsdatascience.com/chinese-natural-language-pre-processing-an-introduction-995d16c2705f

WORKING WITH TEXT: POS

The acronym `POS` refers to parts of speech, which involves identifying the parts of speech for words in a sentence. The following subsections provide more details regarding `POS`, some `POS` techniques, and also `NER` (named entity recognition).

POS Tagging

Parts of Speech (POS) are the grammatical function of the words in a sentence. Consider the following simple English sentence:

```
The sun gives warmth to the Earth.
```

In the preceding example, "sun" is the subject, "gives" is the verb, "warmth" is the direct object, and Earth is the indirect object. In addition, the subject, direct object, and direct object are also nouns.

When the meaning of a word "overloaded," its function depends on the context. Here are three examples of using the word "bank" in three different contexts:

He went to the bank.
He sat on the river bank.
He cannot bank on that outcome.

`POS` tagging refers to assigning a grammatical tag to the words in a corpus, and useful for developing lemmatizers. `POS` tags are used during the creation of parse trees and to define `NER`s (discussed in the next section).

POS Tagging Techniques

Several major `POS` tagging techniques exist, each of which uses different criteria for assigning a POS tag to a given word.

Deep learning architectures, such as RNNs and LSTMs, can be used for assigning tags to words.

Lexical-based methods assign a tag to a word by determining the tags that is the most frequently associated with a given word.

A *probabilistic method* assigns a tag to a given word based on the frequency of a word or the probability of the occurrence of a particular tag sequence.

As you can probably surmise, *rule-based methods* involve rules. An example of a simple rule involves treating words that have an "ing" suffix as verbs. More complex rules can be devised that involve regular expressions. A rule-based system can consist of hundreds of rules for deciding on the tag for a specific word.

Other POS tagging techniques exist, which are also more complex, such as Hidden Markov Model (HMM) POS tagging.

Perform an online search for articles that contain more detailed information about the preceding POS tagging techniques.

CLEANING DATA WITH REGULAR EXPRESSIONS

This section contains a simple preview of what you can accomplish with regular expressions when you need to clean your data. If you are new to regular expressions, look for various blog posts that contain introductory material. The main concepts for understanding the code samples in this section are listed here:

- the range `[A-Z]` matches any uppercase letter
- the range `[a-z]` matches any lowercase letter
- the range `[a-zA-Z]` matches any lowercase or uppercase letter
- the range `[^a-zA-Z]` matches anything *except* lowercase or uppercase letters

Listing 6.1 displays the contents of `text_clean_regex.py` that illustrates how to remove any symbols that are not characters from a text string.

LISTING 6.1: text_clean_regex.py

```
import re # this is for regular expressions

text = "I have 123 apples for sale. Call me at 650-555-1212
or send me email at apples@acme.com."

print("text:")
print(text)
print()

# replace the '@' symbol with the string ' at ':
cleaned1 = re.sub('@', ' at ',text)
print("cleaned1:")
print(cleaned1)

# replace non-letters with ' ':
cleaned2 = re.sub('[^a-zA-Z]', ' ',cleaned1)
print("cleaned2:")
print(cleaned2)

# replace multiple adjacent spaces with a single ' ':
cleaned3 = re.sub('[ ]+', ' ',cleaned2)
print("cleaned3:")
print(cleaned3)
```

Listing 6.1 contains an `import` statement so that we can use regular expressions in the subsequent code. After initializing the variable `text` with a sentence and displaying its contents, the variable `cleaned1` is defined, which involves replacing the "@" symbol with the text string " at. " (Notice the white space before and after the text "at"). Next, the variable `cleaned2` is defined, which involves replacing anything that is not a letter with a white space. Finally, the variable `cleaned3` is defined, which involves "squeezing" multiple white spaces into a single white space. Launch the code in Listing 6.1 and you will see the following output:

```
text:
I have 123 apples for sale. Call me at 650-555-1212 or send
me email at apples@acme.com.

cleaned1:
I have 123 apples for sale. Call me at 650-555-1212 or send
me email at apples at acme.com.
cleaned2:
I have     apples for sale  Call me at              or send
me email at apples at acme com
cleaned3:
I have apples for sale Call me at or send me email at
apples at acme com
```

Listing 6.2 displays the contents of `text_clean_regex2.py` that illustrates how to remove HTML tags from a text string.

LISTING 6.2: text_clean_regex2.py

```
import re # this is for regular expressions

# [^>]:  matches anything except a '>'
# <[^>]: matches anything after '<' except a '>'
tagregex = re.compile(r'<[^>]+>')

def remove_html_tags(doc):
  return tagregex.sub(':', doc)

doc1 = "<html><head></head></body><p>paragraph1</p><div>div
element</div></html>"

print("doc1:")
print(doc1)
print()

doc2 = remove_html_tags(doc1)
print("doc2:")
print(doc2)
```

Listing 6.2 contains an `import` statement, followed by the variable `tagregex`, which is a regular expression that matches a left angle bracket "<", followed by any character except for a right angle bracket ">". Next, the `Python` function `remove_html_tags()` removes all the HTML tags in a text string.

The next portion of Listing 6.2 initializes the variable `doc1` as an HTML string and displays its contents. The final code block invokes the `Python` function `remove_html_tags()` to remove the HTML tags from `doc1`, and then prints the results. Launch the code in Listing 6.2 and you will see the following output:

```
doc1:
<html><head></head></body><p>paragraph1</p><div>div
element</div></html>

doc2:
::::paragraph1::div element::
```

The third (and final) code sample for this section also involves regular expressions, and it is useful when you need to remove contractions (e.g., replacing "that's" with "that is").

Listing 6.3 displays the contents of `text_clean_regex3.py` that illustrates how to replace contractions with the original words.

LISTING 6.3: text_clean_regex3.py

```
import re # this is for regular expressions

def clean_text(text):
  text = text.lower()

  text = re.sub(r"i'm",     "i am", text)
  text = re.sub(r"he's",    "he is", text)
  text = re.sub(r"she's",   "she is", text)
  text = re.sub(r"that's",  "that is", text)
  text = re.sub(r"what's",  "what is", text)
  text = re.sub(r"where's", "where is", text)
  text = re.sub(r"how's",   "how is", text)
  text = re.sub(r"it is",    "it is", text)
  text = re.sub(r"\'ll",    " will", text)
  text = re.sub(r"\'ve",    " have", text)
  text = re.sub(r"\'re",    " are", text)
  text = re.sub(r"\'d",     " would", text)
  text = re.sub(r"n't",     "not", text)
  text = re.sub(r"won't",   "will not", text)
  text = re.sub(r"can't",   "can not", text)

  text = re.sub(r"[-()\"#/@;:<>{}'+=~|.!?,]", "", text)
  return text

sentences = ["It is a hot day and i'm sweating",
             "How's the zoom class going?",
             "She's smarter than me - that's a fact"]

for sent in sentences:
  print("Sentence:",sent)
  print("Cleaned: ",clean_text(sent))
  print("----------------")
```

Listing 6.3 contains an `import` statement, followed by the `Python` function `clean_text()` that performs a brute-force replacement of hyphenated strings with their unhyphenated counterparts. The final code snippet in this function also removes any special characters in a text string.

The next portion of Listing 6.3 initializes the variable `sentences` that contains multiple sentences, followed by a loop that passes individual sentences to the `Python` `clean_text()` function. Launch the code in Listing 6.3 and you will see the following output:

```
Sentence: It is a hot day and i'm sweating
Cleaned:  it is a hot day and i am sweating
----------------
Sentence: How's the zoom class going?
Cleaned:  how is the zoom class going
----------------
Sentence: She's smarter than me - that's a fact
Cleaned:  she is smarter than me  that is a fact
----------------
```

The `Python` package `contractions` is an alternative to regular expressions for expanding contractions. An example is shown later in this chapter.

Listing 6.4 displays the contents of `remove_urls.py` that illustrates how to remove URLs from an array of strings.

LISTING 6.4: remove_urls.py

```
import re
import pandas as pd

arr1 = ["https://www.yahoo.com",
        "http://www.acme.com"]

data = pd.DataFrame(data=arr1)

print("before:")
print(data)
print()

no_urls = []
for url in arr1:
  clean = re.sub(r"http\S+", "", url)
  no_urls.append(clean)

data["cleaned"] = no_urls

print("after:")
print(data)
```

Listing 6.4 starts with two `import` statements, followed by the initialization of the variable `arr1` with two URLs. Next, the variable `data` is a `Pandas` data frame whose contents are based on the contents of `arr1`. The contents of `data` are displayed, followed by a `for` loop that removes the `http` prefix from each element of `arr1`. The last code section appends a new column called `cleaned` to the `data` data frame and then displays the contents of that data frame. Launch the code in Listing 6.4 and you will see the following output:

```
before:
                                0
0  https://www.yahoo.com web page
1    http://www.acme.com web page

after:
                                0    cleaned
0  https://www.yahoo.com web page   web page
1    http://www.acme.com web page   web page
```

This concludes the brief introduction to cleaning data with basic regular expressions.

CLEANING DATA WITH THE CLEANTEXT LIBRARY

This section contains a simple `Python` code sample that uses the `cleantext` library to clean a text string.

Listing 6.1 displays the contents of `clean_text.py` that illustrates how to remove any symbols that are not characters from a text string.

LISTING 6.1: clean_text.py

```
import sys
sys.path.append('/usr/local/lib/python3.9/site-packages')

# pip3 install clean-text

#Importing the clean text library
from cleantext import clean

text = """
私わ悪ガキですよう Göteborg is a city in Sweden (https://
google.com), whose currency is obviously not the same as
the Japanese ¥
"""

#Cleaning the "text" with clean text:
result = clean(text,
      fix_unicode=True,
      to_ascii=True,
      lower=True,
      no_urls=True,
      no_numbers=True,
      no_digits=True,
      no_currency_symbols=True,
      no_punct=True,
      replace_with_punct=" ",
      replace_with_url="",
      replace_with_number="",
        replace_with_digit=" ",
        replace_with_currency_symbol="yen")

print("original:")
print(text)
print()

print("cleaned:")
print(result)
```

Listing 6.4 contains an `import` statement and then initializes the variable `text` as a comment string that contains Hiragana, a city name (with an umlaut), and the Japanese yen currency symbol.

The next portion of Listing 6.4 initializes the variable `result` with the result of invoking the `clean()` API with a long set of parameters and values for those parameters. For example, `no_punct` is equal to `True`, which means that punctuation will be suppressed. Now launch the code in Listing 6.3 and you will see the following output:

```
original:
私わ悪ガキですよう Göteborg is a city in Sweden (https://
google.com), whose currency is obviously not the same as
the Japanese ¥


cleaned:
si wae gakidesuyou goteborg is a city in sweden whose
currency is obviously not the same as the japanese yen
```

Notice that the preceding output translates Hiragana and Kanji to Romaji, but it is only partially correct.

Another `Python` library for cleaning text is called `textcleaner`, which you can also install via `pip3` and more details are here:

https://pypi.org/project/textcleaner/

HANDLING CONTRACTED WORDS

This section contains an example of expanding contractions via the Python package `contractions` that is available here:

https://github.com/kootenpv/contractions

Install the contractions package with this command:

```
pip3 install contractions
```

Listing 6.5 displays the contents of `contract.py` that illustrates how to expand English contractions and also how to add custom expansion rules.

LISTING 6.5: contract.py

```
import contractions

sentences = ["what's new?",
             "how's the weather",
                "it is humid today",
                "we've been there before",
                "you should've been there!",
                "the sky's the limit"]

for sent in sentences:
  result = contractions.fix(sent)
  print("sentence:",sent)
  print("expanded:",result)
  print()

print("=> updating contraction rules...")
contractions.add("sky's", "sky is")

sent = "the sky's the limit"
result = contractions.fix(sent)
print("sentence:",sent)
print("expanded:",result)
print()
```

Listing 6.5 contains an `import` statement and then initializes the variable `sentences` to an array of text strings. The next portion of Listing 6.5 contains a loop that iterates through the array of strings in `sentences`, and then prints each sentence as well as the expanded version of the sentence.

As you will see in the output, the contraction `sky's` in the last sentence in the array `sentences` is not expanded, so we will add a new expansion rule, as shown here:

```
contractions.add("sky's", "sky is")
```

Now when we process this string again, the contraction `sky's` is expanded correctly. Launch the code in Listing 6.5 and you will see the following output:

```
sentence: what's new?
expanded: what is new?

sentence: how's the weather
expanded: how is the weather

sentence: it is humid today
expanded: it is humid today

sentence: we've been there before
expanded: we have been there before

sentence: you should've been there!
expanded: you should have been there!

sentence: the sky's the limit
expanded: the sky's the limit

=> updating contraction rules...
sentence: the sky's the limit
expanded: the sky is the limit
```

As an alternative, you can also write `Python` code to expand contractions, as shown in the following code block:

```
CONTRACT = {"how's":"how is", "what's":"what is", "it is":"it
is", "we've":"we have", "should've" :"should have", "sky's":
"sky is"}

sentences = ["what's new?",
             "how's the weather",
             "it is humid today",
             "we've been there before",
             "you should've been there!",
             "the sky's the limit"]

for sent in sentences:
  words = sent.split()
  expanded = [CONTRACT[w] if w in CONTRACT else w for w in words]
  new_sent = " ".join(expanded)
  print("sentence:",sent)
  print("expanded:",new_sent)
  print()
```

The next section contains some `Python`-based code samples that involve `BoW` (bag of words).

WHAT IS BEAUTIFULSOUP?

`BeautifulSoup` is a very useful `Python` module that provides a plethora of APIs for retrieving the contents of HTML Web pages and extracting subsets of their content using `XPath`-based expressions. If need be, you can find online tutorials that discuss basic concepts of `XPath`.

This section contains three code samples: how to retrieve the contents of an HTML Web page, how to display the contents of HTML anchor (“a”) tags, and how to remove nonalphanumeric characters from HTML anchor (“a”) tags.

Listing 6.16 displays the contents of `scrape_text1.py` that illustrates how to retrieve the contents of the HTML Web page *https://www.github.com*.

LISTING 6.16: scrape_text1.py

```
import requests
import re
from bs4 import BeautifulSoup

src = "https://www.github.com"

# retrieve html web page as text
text = requests.get(src).text
#print("text:",text)

# parse into BeautifulSoup object
soup = BeautifulSoup(text, "html.parser")
print("soup:",soup)
```

Listing 6.16 contains `import` statements, followed by initializing the variable `src` as the URL for Github. Next, the variable `text` is initialized with the contents of the Github Web page, and then the variable `soup` is initialized as an instance of the `BeautifulSoup` class. Notice that `html.parser` is specified, which is why the HTML tags are removed. Launch the code in Listing 6.16 and you will see 1,036 lines of output, and only the first portion of the output is displayed here:

```
soup:
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8"/>
<link href="https://github.githubassets.com" rel="dns-
prefetch"/>
<link href="https://avatars0.githubusercontent.com"
rel="dns-prefetch"/>
<link href="https://avatars1.githubusercontent.com"
rel="dns-prefetch"/>
<link href="https://avatars2.githubusercontent.com"
rel="dns-prefetch"/>
<link href="https://avatars3.githubusercontent.com"
rel="dns-prefetch"/>
<link href="https://github-cloud.s3.amazonaws.com"
rel="dns-prefetch"/>
<link href="https://user-images.githubusercontent.com/"
rel="dns-prefetch"/>
<link crossorigin="anonymous" href="https://github.
githubassets.com/assets/frameworks-146fab5ea30e8afac08dd110
13bb4ee0.css" integrity="sha512-FG+rXqMOivrAjdEQE7tO4BwM1p
oGmg70hJFTlNSxjX87grtrZ6UnPR8NkzwUHlQEGviu9XuRYeO8zH9YwvZh
dg==" media="all" rel="stylesheet">
```

Listing 6.17 displays the contents of `scrape_text2.py` that illustrates how to retrieve the contents of the HTML Web page *https://www.github.com* and display the contents of the HTML anchor ("a") tags. The new code is shown in bold.

LISTING 6.17: scrape_text2.py

```
import requests
import re
from bs4 import BeautifulSoup

src = "https://www.github.com"

# retrieve html web page as text
text = requests.get(src).text
#print("text:",text)

# parse into BeautifulSoup object
soup = BeautifulSoup(text, "html.parser")
print("soup:",soup)
```

```
# display contents of anchors ("a"):
for item in soup.find_all("a"):
  if len(item.contents) > 0:
    print("anchor:",item.get('href'))
```

Listing 6.17 contains three `import` statements and then initializes the variable src with the URL for Github. Next, the variable `text` is initialized with the contents of the Github page. The next snippet initializes the variable `soup` as an instance of the `BeautifulSoup` class. The final block of code is a loop (shown in bold) that iterates through all the `<a>` elements in the variable `text`, and displays the `href` value embedded in each `<a>` element (after determining that its contents are not empty).

Launch the code in Listing 6.17 and you will see 102 lines of output, and only the first portion of the output is displayed here:

```
anchor: #start-of-content
anchor: https://help.github.com/articles/supported-browsers
anchor: https://github.com/
anchor: /join?ref_cta=Sign+up&ref_loc=header+logged+out&ref_
page=%2F&source=header-home
anchor: /features
anchor: /features/code-review/
anchor: /features/project-management/
anchor: /features/integrations
anchor: /features/actions
anchor: /features/packages
anchor: /features/security
anchor: /features#team-management
anchor: /features#hosting
anchor: /customer-stories
anchor: /security
anchor: /team
```

Listing 6.18 displays the contents of `scrape_text3.py` that illustrates how to remove the nonalphanumeric characters from the HTML anchor ("a") tags in the HTML Web page *https://www.github.com*. The new code is shown in bold.

LISTING 6.18: scrape_text3.py

```
import requests
import re
from bs4 import BeautifulSoup

# removes non-alphanumeric characters
def remove_non_alpha_chars(text):
  # define the pattern to keep
  regex = r'[^a-zA-Z0-9]'
  return re.sub(regex, '', text)

src = "https://www.github.com"
```

```
# retrieve html web page as text
text = requests.get(src).text
#print("text:",text)

# parse into BeautifulSoup object
soup = BeautifulSoup(text, "html.parser")
print("soup:",soup)
# display contents of anchors ("a"):
for item in soup.find_all("a"):
  if len(item.contents) > 0:
   #print("anchor:",item.get('href'))
    cleaned = remove_non_alpha_chars(item.get('href'))
    print("cleaned:",cleaned)
```

Listing 6.18 is similar to Listing 6.16, and differs in the `for` loop (shown in bold) and that displays the `href` values after removing nonalphabetic characters and also all whitespaces. Launch the code in Listing 6.18 and you will see 102 lines of output, and only the first portion of the output is displayed here:

```
cleaned: startofcontent
cleaned: httpshelpgithubcomarticlessupportedbrowsers
cleaned: httpsgithubcom
cleaned: joinrefctaSignupreflocheaderloggedoutrefpage2F-
sourceheaderhome
cleaned: features
cleaned: featurescodereview
cleaned: featuresprojectmanagement
cleaned: featuresintegrations
cleaned: featuresactions
cleaned: featurespackages
cleaned: featuressecurity
cleaned: featuresteammanagement
cleaned: featureshosting
cleaned: customerstories
cleaned: security
cleaned: team
```

WEB SCRAPING WITH PURE REGULAR EXPRESSIONS

The previous section contains several examples of using `BeautifulSoup` to scrape HTML Web pages, and this section contains an example that involves only regular expressions. Once again, you need some familiarity with regular expressions, which are discussed in one of the appendices.

Listing 6.19 displays the contents of `scrape_pure_regex.py` that illustrates how to retrieve the contents of the HTML Web page *https://www.github.com* and remove the HTML tags with a single regular expression.

LISTING 6.19: scrape_pure_regex.py

```
import requests
import re
import os
```

```
src = "https://www.github.com"

# retrieve the web page contents:
r = requests.get(src)
print(r.text)

# remove HTML tags (notice the "?"):
pattern = re.compile(r'<.*?>')

cleaned = pattern.sub('', r.text)

#remove leading whitespaces:
cleaned = os.linesep.join([s.lstrip() for s in cleaned.
splitlines() if s])

#remove embedded blank lines:
cleaned = os.linesep.join([s for s in cleaned.splitlines()
if s])
print("cleaned text:")
print(cleaned)

#remove ALL whitespaces:
#cleaned = cleaned.replace(" ", "")

#this does not work:
#cleaned = cleaned.trim(" ", "")
```

Listing 6.19 also removes the HTML elements from the variable `text` that contains the contents of the Github Web page. However, there is a subtle yet very important detail regarding the regular expression `<.*>` versus the regular expression `<.*?>`.

The regular expression `<.*>` performs a *greedy* match, which means that it will continue matching characters until the right-most ">" is encountered. However, we want the greedy match to *stop* after finding the first ">" character, which matches the previous "<" character. The solution is simple: specify the regular expression `<.*?>`, which contains a question mark ("?") whose purpose is to disable the greedy matching nature of the metacharacter "*".

Launch the code in Listing 6.19 and you will see the following output (some output omitted for brevity):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
  <link rel="dns-prefetch" href="https://github.
githubassets.com">
  <link rel="dns-prefetch" href="https://avatars0.
githubusercontent.com">
  <link rel="dns-prefetch" href="https://avatars1.
githubusercontent.com">
  <link rel="dns-prefetch" href="https://avatars2.
githubusercontent.com">
```

```
  <link rel="dns-prefetch" href="https://avatars3.
githubusercontent.com">
  <link rel="dns-prefetch" href="https://github-cloud.
s3.amazonaws.com">
  <link rel="dns-prefetch" href="https://user-images.
githubusercontent.com/">

  <link crossorigin="anonymous" media="all"
integrity="sha512-/uy49LxdzjR0L36uT6CnmV1omP/8ZHxvOg4
zq/dczzABHq9atntjJDmo5B7sV0J+AwVmv0fR0ZyW3EQawzdLFA=="
rel="stylesheet" href="https://github.githubassets.com/
assets/frameworks-feecb8f4bc5dce34742f7eae4fa0a799.css" />
  <link crossorigin="anonymous" media="all"
integrity="sha512-37pLQI8klDWPjWVVWFB9ITJLwVTTkp3Rt4bV
f+yixrViURK9OoGHEJDbTLxBv/rTJhsLm8pb00H2H5AG3hUJfg=="
rel="stylesheet" href="https://github.githubassets.com/
assets/site-dfba4b408f2494358f8d655558507d21.css" />
  <meta name="viewport" content="width=device-width">

  <title>The world's leading software development platform
· GitHub</title>

[details omitted for brevity]
    <div class="position-relative js-header-wrapper ">
      <a href="#start-of-content" class="px-2 py-4 bg-
blue text-white show-on-focus js-skip-to-content">Skip to
content</a>
      <span class="Progress progress-pjax-loader position-
fixed width-full js-pjax-loader-bar">
        <span class="progress-pjax-loader-bar top-0 left-0"
style="width: 0%;"></span>
      </span>
[details omitted for brevity]
cleaned text:
The world's leading software development platform · GitHub
Skip to content
GitHub no longer supports this web browser.
Learn more about the browsers we support.
<a href="/join?ref_cta=Sign+up&amp;ref_
loc=header+logged+out&amp;ref_page=%2F&amp;source=header-
home"
[details omitted for brevity]
```

WHAT IS SCRAPY?

In the previous section, you learned that `BeautifulSoup` is a `Python`-based library for scraping HTML Web pages. `BeautifulSoup` also supports `XPath` (which is an integral component of XSLT), whose APIs enable you to parse the scraped data, and also to extract portions of that data.

However, `Scrapy` is a `Python`-based library that provides data extraction and an assortment of additional APIs for a wide range of operations, including redirections, HTTP caching, filtering duplicated requests, preserving sessions/cookies across multiple requests, and various other features. `Scrapy` supports both CSS selectors and XPath expressions for data extraction. Moreover, you can also use BeautifulSoup or PyQuery as a data extraction mechanism.

Based on the preceding paragraph, you can probably see that while Scrapy and BeautifulSoup can do some of the same things (i.e., Web scraping), they have fundamentally different purposes. As a rule of thumb: use BeautifulSoup if you need a "one-off" Web page scraper. However, if you need to perform Web scraping and perform additional operations for one or more Web pages, then Scrapy is probably a better choice.

At the same time, keep in mind that Scrapy does have a steeper learning curve than BeautifulSoup, so decide whether or not the extra features of Scrapy are necessary for your requirements before you invest your time learning Scrapy. As a starting point, the Scrapy documentation Web page is here:

https://doc.scrapy.org/en/latest/intro/tutorial.html

SUMMARY

This chapter described some common NLP tasks and NLP techniques, such as stemming and lemmatization. Then you learned about POS (parts of speech) in text as well as POS tagging techniques.

You also how to clean text via regular expression (REs), and also how to handle contractions of words. Then you learned about BeautifulSoup, which is a Python module for scraping HTML Web pages.

Finally, you learned about Scrapy, which is a Python-based library that provides Web scraping functionality and various other APIs.

CHAPTER 7

DATA VISUALIZATION

This chapter introduces data visualization, along with a wide-ranging collection of Python-based code samples that use various visualization tools (including Matplotlib and Seaborn) to render charts and graphs. In addition, this chapter contains Python code samples that combine Pandas, Matplotlib, and built-in datasets.

The first part of this chapter briefly discusses data visualization, with a short list of some data visualization tools, and a list of various types of visualization (bar graphs, pie charts, and so forth).

The second part of this chapter introduces you to Matplotlib, which is an open source Python library that is modeled after MatLab. This section also provides the Python code samples for the line graphs (horizontal, vertical, and diagonal) in the Euclidean plane that you saw in a previous chapter.

The third part of the chapter introduces you to Seaborn for data visualization, which is a layer above Matplotlib. Although Seaborn does not have all of the features that are available in Matplotlib, Seaborn provides an easier set of APIs for rendering charts and graphs.

The final portion of this chapter contains a very short introduction to Bokeh, along with a code sample that illustrates how to create a more artistic graphics effect with relative ease in Bokeh.

WHAT IS DATA VISUALIZATION?

Data visualization refers to presenting data in a graphical manner, such as bar charts, line graphs, heat maps, and many other specialized representations. As you probably know, big data comprises massive amounts of data, which leverages data visualization tools to assist in making better decisions.

A key role for good data visualization is to tell a meaningful story, which in turn focuses on useful information that resides in datasets that can contain many data points (i.e., billions of rows of data). Another aspect of data visualization is its effectiveness: how well does it convey the trends that might exist in the dataset?

There are many open source data visualization tools available, some of which are listed here (many others are available):

- Matplotlib
- Seaborn
- Bokeh
- YellowBrick
- Tableau
- D3.js (JavaScript and SVG)

Incidentally, in case you have not already done so, it would be helpful to install the following `Python` libraries (using `pip3`) on your computer so that you can launch the code samples in this chapter:

```
pip3 install matplotlib
pip3 install seaborn
pip3 install bokeh
```

Types of Data Visualization

Bar graphs, line graphs, and pie charts are common ways to present data, and yet many other types exist, some of which are listed here:

- 2D/3D area chart
- bar chart
- Gantt chart
- heat map
- histogram
- polar area
- scatter plot (2D or 3D)
- timeline

The `Python` code samples in the next several sections illustrate how to perform visualization via rudimentary APIs from `matplotlib`.

WHAT IS MATPLOTLIB?

`Matplotlib` is a plotting library that supports `NumPy`, `SciPy`, and toolkits such as `wxPython` (among others). `Matplotlib` supports only version 3 of `Python`: support for version 2 of Python was available only through 2020. `Matplotlib` is a multi-platform library that is built on `NumPy` arrays.

The plotting-related code samples in this chapter use `pyplot`, which is a `Matplotlib` module that provides a `MATLAB`-like interface.

Here is an example of using `pyplot` (copied from *https://www.biorxiv.org/content/10.1101/120378v1.full.pdf*) to plot a smooth curve based on negative powers of Euler's constant e:

```
import matplotlib.pyplot as plt
import numpy as np

a = np.linspace(0, 10, 100)
b = np.exp(-a)
plt.plot(a, b)
plt.show()
```

The `Python` code samples for visualization in this chapter use primarily `Matplotlib`, along with some code samples that use `Seaborn`. Keep in mind that the code samples that plot line segments assume that you are familiar with the equation of a (non-vertical) line in the plane: `y = m*x + b`, where `m` is the slope and `b` is the y-intercept.

Furthermore, some code samples use `NumPy` APIs such as `np.linspace()`, `np.array()`, `np.random.rand()`, and `np.ones()` and you read relevant online articles if you need to refresh your memory for these APIs.

LINES IN A GRID IN MATPLOTLIB

Listing 7.1 displays the contents of `diagonallines.py` that illustrate how to plot lines in a grid.

LISTING 7.1: diagonallines.py

```
import numpy as np
import pylab
from itertools import product
import matplotlib.pyplot as plt

fig = plt.figure()
graph = fig.add_subplot(1,1,1)
graph.grid(which='major', linestyle='-', linewidth='0.5', color='red')

x1 = np.linspace(-5,5,num=200)
y1 = 1*x1
graph.plot(x1,y1, 'r-o')

x2 = np.linspace(-5,5,num=200)
y2 = -x2
graph.plot(x2,y2, 'b-x')

fig.show() # to update
plt.show()
```

Listing 7.1 contains some `import` statements and then initializes the variables `x1` and `y1` as an array of 200 equally space points, where the points in `y1` equal the values in `x1`.

Similarly, the variables `x2` and `y2` are defined, except that the values in `y2` are the negative of the values in `x2`. The `Pyplot` API `plot()` uses the `points` variable to display a pair of diagonal line segments.

Figure 7.1 displays a set of "dashed" line segment whose equations are contained in Listing 7.1.

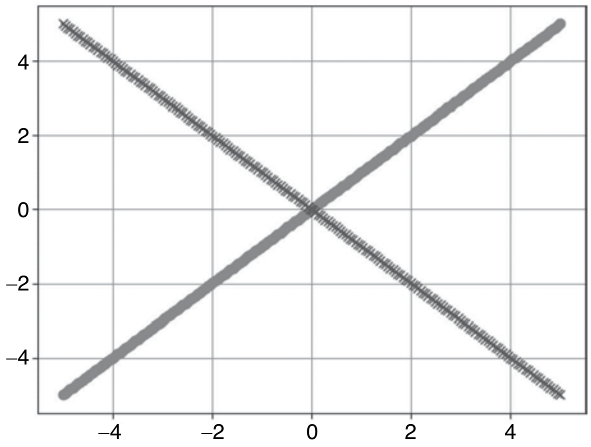

FIGURE 7.1. A pair of diagonal line segments.

A COLORED GRID IN `MATPLOTLIB`

Listing 7.2 displays the contents of `plotgrid2.py` that illustrate how to display a colored grid.

LISTING 7.2: plotgrid2.py

```
import matplotlib.pyplot as plt
from matplotlib import colors
import numpy as np

data = np.random.rand(10, 10) * 20

# create discrete colormap
cmap = colors.ListedColormap(['red', 'blue'])
bounds = [0,10,20]
norm = colors.BoundaryNorm(bounds, cmap.N)

fig, ax = plt.subplots()
ax.imshow(data, cmap=cmap, norm=norm)

# draw gridlines
ax.grid(which='major', axis='both', linestyle='-',
color='k', linewidth=2)
ax.set_xticks(np.arange(-.5, 10, 1));
ax.set_yticks(np.arange(-.5, 10, 1));

plt.show()
```

Listing 7.2 defines the `NumPy` variable `data` that defines a 2D set of points with 10 rows and 10 columns. The `Pyplot` API `plot()` uses the `data` variable to display a colored grid-like pattern.

Figure 7.2 displays a colored grid whose equations are contained in Listing 7.2.

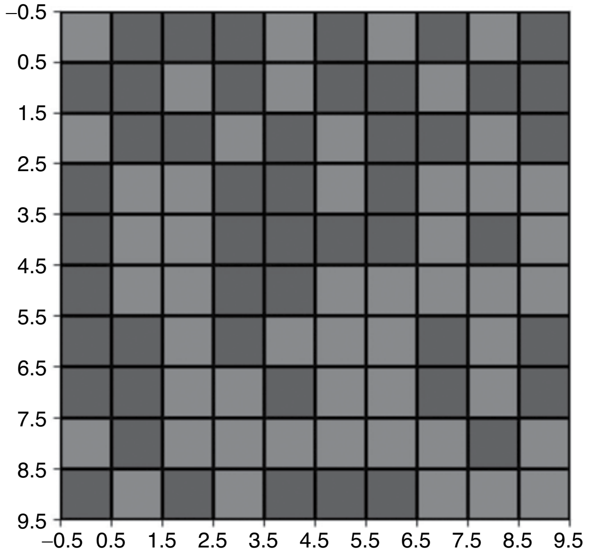

FIGURE 7.2. A colored grid of line segments.

RANDOMIZED DATA POINTS IN MATPLOTLIB

Listing 7.3 displays the contents of `lin_reg_plot.py` that illustrate how to plot a graph of random points.

LISTING 7.3: lin_plot_reg.py

```
import numpy as np
import matplotlib.pyplot as plt

trX = np.linspace(-1, 1, 101) # Linear space of 101 and [-1,1]

#Create the y function based on the x axis
trY = 2*trX + np.random.randn(*trX.shape)*0.4+0.2

#create figure and scatter plot of the random points
plt.figure()
plt.scatter(trX,trY)

# Draw one line with the line function
plt.plot (trX, .2 + 2 * trX)
plt.show()
```

Listing 7.3 defines the NumPy variable trX that contains 101 equally spaced numbers that are between –1 and 1 (inclusive). The variable trY is defined in two parts: the first part is 2*trX and the second part is a random value that is partially based on the length of the one-dimensional array trX. The variable trY is the sum of these two "parts", which creates a "fuzzy" line segment. The next portion of Listing 7.3 creates a scatterplot based on the values in trX and trY, followed by the Pyplot API plot() that renders a line segment.

Figure 7.3 displays a random set of points based on the code in Listing 7.3.

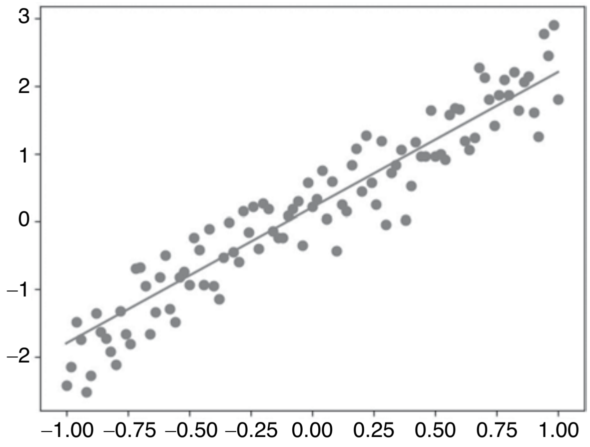

FIGURE 7.3. A random set of points.

A HISTOGRAM IN MATPLOTLIB

Listing 7.4 displays the contents of histogram1.py that illustrate how to plot a histogram using Matplotlib.

LISTING 7.4: histogram1.py

```
import numpy as np
import Matplotlib.pyplot as plt

max1 = 500
max2 = 500

appl_count = 28 + 4 * np.random.randn(max1)
bana_count = 24 + 4 * np.random.randn(max2)

plt.hist([appl_count, appl_count],stacked=True,color=['r','b'])
plt.show()
```

Listing 7.4 is straightforward: the NumPy variables appl_count and bana_count contain a random set of values whose upper bound is max1 and max2, respectively. The Pyplot API hist() uses the points appl_count and bana_count in order to display a histogram. Figure 7.4 displays a histogram whose shape is based on the code in Listing 7.4.

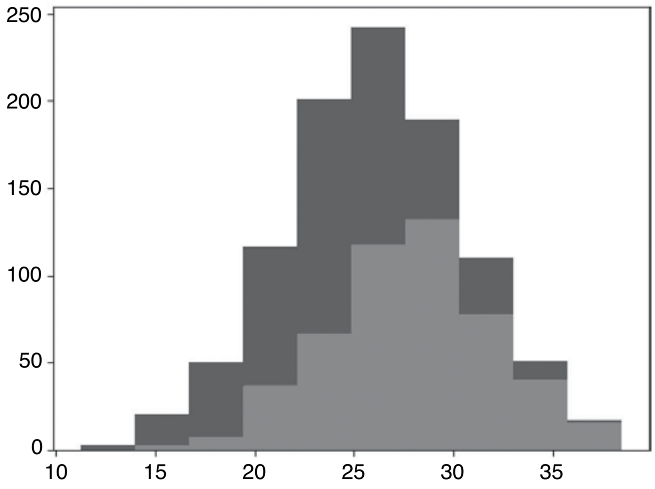

FIGURE 7.4. A histogram based on random values.

A SET OF LINE SEGMENTS IN MATPLOTLIB

Listing 7.5 displays the contents of line_segments.py that illustrate how to plot a set of connected line segments in Matplotlib.

LISTING 7.5: line_segments.py

```
import numpy as np
import matplotlib.pyplot as plt

x = [7,11,13,15,17,19,23,29,31,37]

plt.plot(x) # OR: plt.plot(x, 'ro-') or bo
plt.ylabel('Height')
plt.xlabel('Weight')
plt.show()
```

Listing 7.5 defines the array x that contains a hard-coded set of values. The Pyplot API plot() uses the variable x to display a set of connected line segments. Figure 7.5 displays the result of launching the code in Listing 7.5.

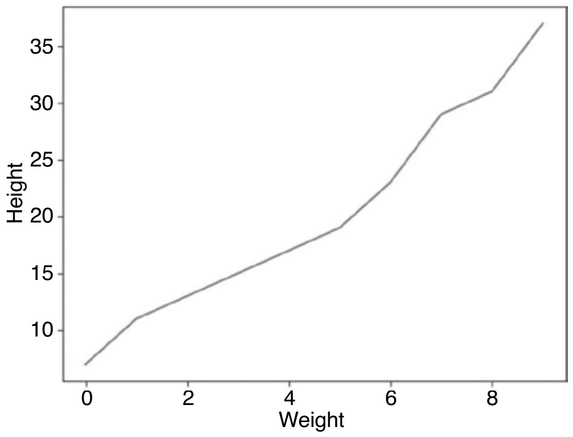

FIGURE 7.5 A set of connected line segments.

PLOTTING MULTIPLE LINES IN `MATPLOTLIB`

Listing 7.6 displays the contents of `plt_array2.py` that illustrate the ease with which you can plot multiple lines in `Matplotlib`.

LISTING 7.6: plt_array2.py

```
import matplotlib.pyplot as plt

x = [7,11,13,15,17,19,23,29,31,37]
data = [[8, 4, 1], [5, 3, 3], [6, 0, 2], [1, 7, 9]]
plt.plot(data, 'd-')
plt.show()
```

Listing 7.6 defines the array `data` that contains a hard-coded set of values. The `Pyplot` API `plot()` uses the variable `data` to display a line segment. Figure 7.6 displays multiple lines based on the code in Listing 7.6.

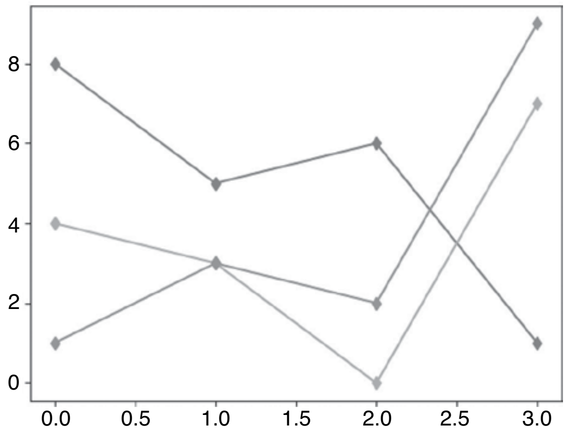

FIGURE 7.6. Multiple lines in Matplotlib.

TRIGONOMETRIC FUNCTIONS IN `MATPLOTLIB`

In case you are wondering, you can display the graph of trigonometric functions as easily as you can render "regular" graphs using Matplotlib. Listing 7.7 displays the contents of `sincos.py` that illustrates how to plot a sine function and a cosine function in `Matplotlib`.

LISTING 7.7: sincos.py

```
import numpy as np
import math

x = np.linspace(0, 2*math.pi, 101)
s = np.sin(x)
c = np.cos(x)

import matplotlib.pyplot as plt
plt.plot (s)
plt.plot (c)
plt.show()
```

Listing 7.7 defines the `NumPy` variables `x`, `s`, and `c` using the `NumPy` APIs `linspace()`, `sin()`, and `cos()`, respectively. Next, the `Pyplot` API `plot()` uses these variables to display a sine function and a cosine function.

Figure 7.7 displays a graph of two trigonometric functions based on the code in Listing 7.7.

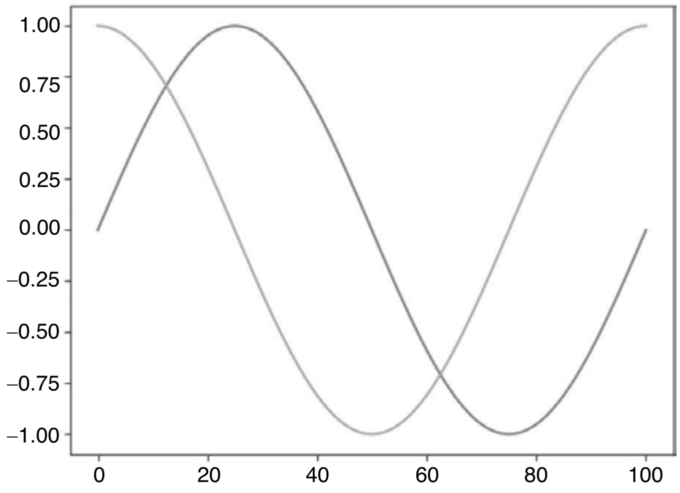

FIGURE 7.7. Sine and cosine trigonometric functions.

Now let's look at a simple dataset consisting of discrete data points, which is the topic of the next section.

DISPLAY IQ SCORES IN **`MATPLOTLIB`**

Listing 7.8 displays the contents of `iq_scores.py` that illustrates how to plot a histogram that displays IQ scores (based on a normal distribution).

LISTING 7.8: iq_scores.py

```
import numpy as npf
import matplotlib.pyplot as plt

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.xlabel('Intelligence')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

Listing 7.8 defines the scalar variables `mu` and `sigma`, followed by the `NumPy` variable `x` that contains a random set of points. Next, the variables `n`, `bins`, and `patches` are initialized via the return values of the `NumPy` `hist()` API. Finally, these points are plotted via the usual `plot()` API to display a histogram.

Figure 7.8 displays a histogram whose shape is based on the code in Listing 7.8.

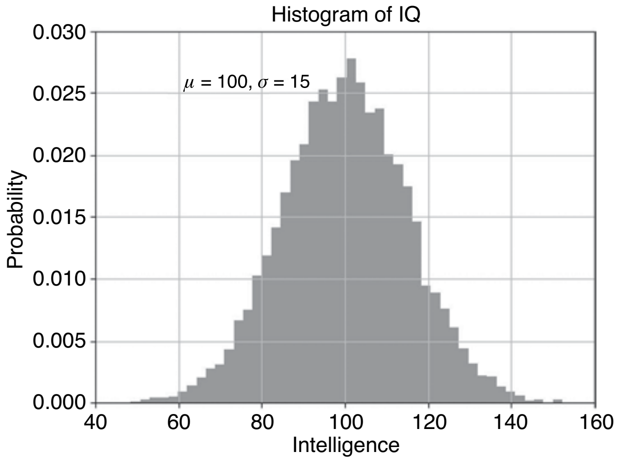

FIGURE 7.8. A histogram to display IQ scores.

PLOT A BEST-FITTING LINE IN **`MATPLOTLIB`**

Listing 7.9 displays the contents of `plot_best_fit.py` that illustrates how to plot a best-fitting line in `Matplotlib`.

LISTING 7.9: plot_best_fit.py

```
import numpy as np

xs = np.array([1,2,3,4,5], dtype=np.float64)
ys = np.array([1,2,3,4,5], dtype=np.float64)

def best_fit_slope(xs,ys):
  m = (((np.mean(xs)*np.mean(ys))-np.mean(xs*ys)) /
       ((np.mean(xs)**2) - np.mean(xs**2)))
  b = np.mean(ys) - m * np.mean(xs)

  return m, b

m,b = best_fit_slope(xs,ys)
print('m:',m,'b:',b)

regression_line = [(m*x)+b for x in xs]

import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')

plt.scatter(xs,ys,color='#0000FF')
plt.plot(xs, regression_line)
plt.show()
```

Listing 7.9 defines the `NumPy` array variables `xs` and `ys` that are "fed" into the `Python` function `best_fit_slope()` that calculates the slope `m` and the y-intercept `b` for the best-fitting line. The `Pyplot` API `scatter()` displays a scatter plot of the points `xs` and `ys`, followed by the `plot()` API that displays the best-fitting line. Figure 7.9 displays a simple line based on the code in Listing 7.9.

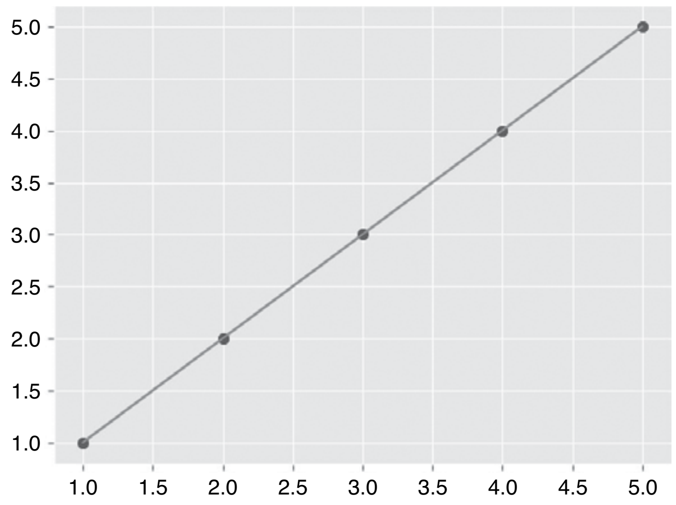

FIGURE 7.9. A best-fitting line for a 2D dataset.

This concludes the portion of the chapter regarding NumPy and Matplotlib. The next section introduces you to Sklearn, which is a powerful Python-based library that supports many algorithms for machine learning. After you have read the short introduction, subsequent sections contain Python code samples that combine Pandas, Matplotlib, and Sklearn built-in datasets.

THE IRIS DATASET IN SKLEARN

Listing 7.10 displays the contents of sklearn_iris.py that illustrates how to access the Iris dataset in Sklearn.

In addition to support for machine learning algorithms, Sklearn provides various built-in datasets that you can access with literally one line of code. In fact, Listing 7.10 displays the contents of sklearn_iris1.py that illustrates how you can easily load the Iris dataset into a Pandas data frame.

LISTING 7.10: sklearn_iris1.py

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

print("=> iris keys:")
for key in iris.keys():
  print(key)
print()

#print("iris dimensions:")
#print(iris.shape)
#print()

print("=> iris feature names:")
for feature in iris.feature_names:
  print(feature)
print()

X = iris.data[:, [2, 3]]
y = iris.target
print('=> Class labels:', np.unique(y))
print()

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

print("=> target:")
print(iris.target)
print()

print("=> all data:")
print(iris.data)
```

Listing 7.10 contains several `import` statements and then initializes the variable `iris` with the `Iris` dataset. Next, a loop displays the keys in the dataset, followed by another loop that displays the feature names.

The next portion of Listing 7.10 initializes the variable `X` with the feature values in columns 2 and 3, and then initializes the variable `y` with the values of the target column.

The variable `x_min` is initialized as the minimum value of column 0 and then an additional 0.5 is subtracted from x_min. Similarly, the variable `x_max` is initialized as the maximum value of column 0, and then an additional 0.5 is added to `x_max`. The variables `y_min` and `y_max` are the counterparts to `x_min` and `x_max`, applied to column 1 instead of column 0.

Launch the code in Listing 7.10 and you will see the following output (truncated to save space):

```
Pandas df1:

=> iris keys:
data
target
target_names
DESCR
feature_names
filename

=> iris feature names:
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

=> Class labels: [0 1 2]

=> x_min: 0.5 x_max: 7.4
=> y_min: -0.4 y_max: 3.0

=> target:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]

=> all data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 // details omitted for brevity
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

Sklearn, Pandas, and the Iris Dataset

Listing 7.11 displays the contents of `pandas_iris.py` that illustrates how to load the contents of the `Iris` dataset (from `Sklearn`) into a `Pandas` data frame.

LISTING 7.11: pandas_iris.py

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

print("=> IRIS feature names:")
for feature in iris.feature_names:
  print(feature)
print()

# Create a data frame with the feature variables
df = pd.Data frame(iris.data, columns=iris.feature_names)

print("=> number of rows:")
print(len(df))
print()

print("=> number of columns:")
print(len(df.columns))
print()

print("=> number of rows and columns:")
print(df.shape)
print()

print("=> number of elements:")
print(df.size)
print()

print("=> IRIS details:")
print(df.info())
print()

print("=> top five rows:")
print(df.head())
print()

X = iris.data[:, [2, 3]]
y = iris.target
print('=> Class labels:', np.unique(y))
```

Listing 7.11 contains several `import` statements and then initializes the variable `iris` with the `IRIS` dataset. Next, a `for` loop displays the feature names. The next code snippet initializes the variable `df` as a `Pandas` data frame that contains the data from the `IRIS` dataset.

The next block of code invokes some attributes and methods of a `Pandas` data frame to display the number of rows, columns, and elements in the data frame, as well as the details of the `Iris` dataset, the first five rows, and the unique labels in the `IRIS` dataset. Launch the code in Listing 7.20 and you will see the following output:

```
=> IRIS feature names:
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

=> number of rows:
150

=> number of columns:
4

=> number of rows and columns:
(150, 4)

=> number of elements:
600

=> IRIS details:
<class 'pandas.core.frame.Data frame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
dtypes: float64(4)
memory usage: 4.8 KB
None

=> top five rows:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

=> Class labels: [0 1 2]
```

Now we will turn our attention to Seaborn, which is an efficient data visualization package for Python.

WORKING WITH SEABORN

Seaborn is a Python package for data visualization that also provides a high-level interface to Matplotlib. Seaborn is easier to work with than Matplotlib, and actually extends Matplotlib, but keep in mind that Seaborn is not as powerful as Matplotlib.

Seaborn addresses two challenges of Matplotlib. The first involves the default Matplotlib parameters. Seaborn works with different parameters, which provides greater flexibility than the default rendering of Matplotlib plots. Seaborn addresses the limitations of the Matplotlib default values for features such as colors, tick marks on the upper and right axes, and the style (among others).

In addition, Seaborn makes it easier to plot entire data frames (somewhat like pandas) than doing so in Matplotlib. Nevertheless, since Seaborn extends Matplotlib, knowledge of the latter is advantageous and will simplify your learning curve.

Features of Seaborn

Some of the features of Seaborn include:

- scale seaborn plots
- set the plot style
- set the figure size
- rotate label text
- set xlim or ylim
- set log scale
- add titles

Some useful methods:

- `plt.xlabel()`
- `plt.ylabel()`
- `plt.annotate()`
- `plt.legend()`
- `plt.ylim()`
- `plt.savefig()`

Seaborn supports various built-in datasets, just like NumPy and Pandas, including the Iris dataset and the Titanic dataset, both of which you will see in subsequent sections. As a starting point, the three-line code sample in the next section shows you how to display the rows in the built-in "tips" dataset.

SEABORN BUILT-IN DATASETS

Listing 7.12 displays the contents of seaborn_tips.py that illustrate how to read the tips dataset into a data frame and display the first five rows of the dataset.

LISTING 7.23: seaborn_tips.py

```
import seaborn as sns
df = sns.load_dataset("tips")
print(df.head())
```

Listing 7.12 is very simple: after importing seaborn, the variable df is initialized with the data in the built-in dataset tips, and the print() statement displays the first five rows of df. Note that the load_dataset() API searches for online or built-in datasets. The output from Listing 7.12 is here:

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

THE IRIS DATASET IN SEABORN

Listing 7.13 displays the contents of `seaborn_iris.py` that illustrate how to plot the `Iris` dataset.

LISTING 7.13: seaborn_iris.py

```
import seaborn as sns
import Matplotlib.pyplot as plt

# Load iris data
iris = sns.load_dataset("iris")

# Construct iris plot
sns.swarmplot(x="species", y="petal_length", data=iris)

# Show plot
plt.show()
```

Listing 7.13 imports `seaborn` and `Matplotlib.pyplot` and then initializes the variable `iris` with the contents of the built-in `Iris` dataset. Next, the `swarmplot()` API displays a graph with the horizontal axis labeled `species`, the vertical axis labeled `petal_length`, and the displayed points are from the `Iris` dataset.

Figure 7.10 displays the images in the `Iris` dataset based on the code in Listing 7.13.

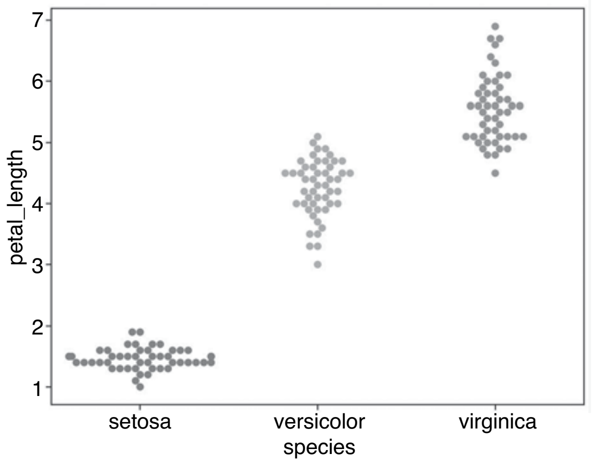

FIGURE 7.10 The Iris dataset.

THE TITANIC DATASET IN SEABORN

Listing 7.14 displays the contents of `seaborn_titanic_plot.py` that illustrates how to plot the `Titanic` dataset.

LISTING 7.14: seaborn_titanic_plot.py

```
import matplotlib.pyplot as plt
import seaborn as sns

titanic = sns.load_dataset("titanic")
g = sns.factorplot("class", "survived", "sex",
data=titanic, kind="bar", palette="muted", legend=False)

plt.show()
```

Listing 7.14 contains the same `import` statements as Listing 7.13, and then initializes the variable `titanic` with the contents of the built-in `Titanic` dataset. Next, the `factorplot()` API displays a graph with dataset attributes that are listed in the API invocation.

Figure 7.11 displays a plot of the data in the `Titanic` dataset based on the code in Listing 7.14.

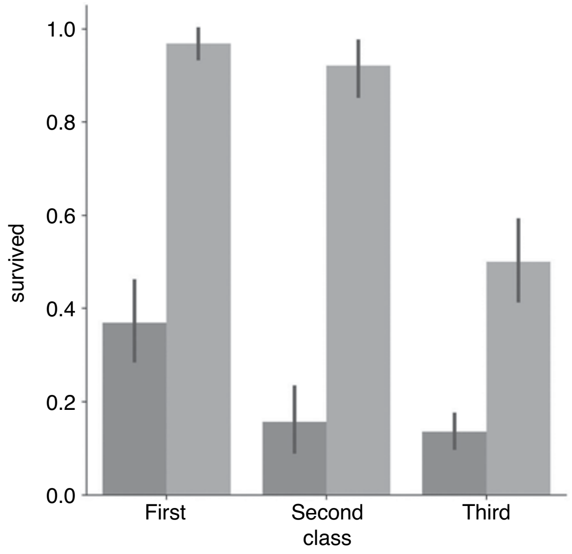

FIGURE 7.11. A histogram of the Titanic dataset.

EXTRACTING DATA FROM THE TITANIC DATASET IN SEABORN (1)

Listing 7.15 displays the contents of `seaborn_titanic.py` that illustrates how to extract subsets of data from the `Titanic` dataset.

LISTING 7.15: seaborn_titanic.py

```
import matplotlib.pyplot as plt
import seaborn as sns

titanic = sns.load_dataset("titanic")
print("titanic info:")
titanic.info()

print("first five rows of titanic:")
print(titanic.head())

print("first four ages:")
print(titanic.loc[0:3,'age'])

print("fifth passenger:")
print(titanic.iloc[4])

#print("first five ages:")
#print(titanic['age'].head())

#print("first five ages and gender:")
#print(titanic[['age','sex']].head())

#print("descending ages:")
#print(titanic.sort_values('age', ascending = False).head())

#print("older than 50:")
#print(titanic[titanic['age'] > 50])

#print("embarked (unique):")
#print(titanic['embarked'].unique())

#print("survivor counts:")
#print(titanic['survived'].value_counts())

#print("counts per class:")
#print(titanic['pclass'].value_counts())

#print("max/min/mean/median ages:")
#print(titanic['age'].max())
#print(titanic['age'].min())
#print(titanic['age'].mean())
#print(titanic['age'].median())
```

Listing 7.15 contains the same `import` statements as Listing 7.13, and then initializes the variable `titanic` with the contents of the built-in `Titanic` dataset. The next portion of Listing 7.15 displays various aspects of the Titanic dataset, such as its structure, the first five rows, the first four ages, and the details of the fifth passenger.

As you can see, there is a large block of "commented out" code that you can uncomment in order to see the associated output, such as age, gender, persons over 50, unique rows, and so forth. The output from Listing 7.15 is here:

```
#print(titanic['age'].mean())
titanic info:
<class 'pandas.core.frame.Data frame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
survived       891 non-null int64
pclass         891 non-null int64
sex            891 non-null object
age            714 non-null float64
sibsp          891 non-null int64
parch          891 non-null int64
fare           891 non-null float64
embarked       889 non-null object
class          891 non-null category
who            891 non-null object
adult_male     891 non-null bool
deck           203 non-null category
embark_town    889 non-null object
alive          891 non-null object
alone          891 non-null bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.6+ KB
first five rows of titanic:
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
first four ages:
0    22.0
1    38.0
2    26.0
3    35.0
Name: age, dtype: float64
fifth passenger:
survived                 0
pclass                   3
sex                   male
age                     35
sibsp                    0
parch                    0
fare                  8.05
embarked                 S
class                Third
who                    man
adult_male            True
deck                   NaN
embark_town    Southampton
alive                   no
alone                 True
Name: 4, dtype: object
counts per class:
3    491
```

```
1    216
2    184
Name: pclass, dtype: int64
max/min/mean/median ages:
80.0
0.42
29.69911764705882
28.0
```

EXTRACTING DATA FROM TITANIC DATASET IN SEABORN (2)

Listing 7.16 displays the contents of `seaborn_titanic2.py` that illustrates how to extract subsets of data from the `Titanic` dataset.

LISTING 7.16: seaborn_titanic2.py

```
import matplotlib.pyplot as plt
import seaborn as sns

titanic = sns.load_dataset("titanic")

# Returns a scalar
# titanic.ix[4, 'age']
print("age:",titanic.at[4, 'age'])

# Returns a Series of name 'age', and the age values associated
# to the index labels 4 and 5
# titanic.ix[[4, 5], 'age']
print("series:",titanic.loc[[4, 5], 'age'])

# Returns a Series of name '4', and the age and fare values
# associated to that row.
# titanic.ix[4, ['age', 'fare']]
print("series:",titanic.loc[4, ['age', 'fare']])

# Returns a Data frame with rows 4 and 5, and columns 'age' and 'fare'
# titanic.ix[[4, 5], ['age', 'fare']]
print("data frame:",titanic.loc[[4, 5], ['age', 'fare']])

query = titanic[
    (titanic.sex == 'female')
    & (titanic['class'].isin(['First', 'Third']))
    & (titanic.age > 30)
    & (titanic.survived == 0)
]
print("query:",query)
```

Listing 7.16 contains the same `import` statements as Listing 7.15, and then initializes the variable `titanic` with the contents of the built-in `Titanic` dataset. The next code snippet displays the age of the passenger with index 4 in the dataset (which equals 35).

The following code snippet displays the ages of passengers with index values 4 and 5 in the dataset:

```
print("series:",titanic.loc[[4, 5], 'age'])
```

The next snippet displays the age and fare of the passenger with index 4 in the dataset, followed by another code snippet displays the age and fare of the passengers with index 4 and index 5 in the dataset.

The final portion of Listing 7.16 is the most interesting part: it defines a variable `query` as shown here:

```
query = titanic[
    (titanic.sex == 'female')
    & (titanic['class'].isin(['First', 'Third']))
    & (titanic.age > 30)
    & (titanic.survived == 0)
]
```

The preceding code block will retrieve information about the female passengers who were in first class or third class, and who were also over 30, and did not survive the accident. The entire output from Listing 7.16 is here:

```
age: 35.0
series: 4    35.0
5     NaN
Name: age, dtype: float64
series: age       35
fare    8.05
Name: 4, dtype: object
data frame:     age    fare
4  35.0  8.0500
5   NaN  8.4583
query:      survived  pclass     sex   age  sibsp  parch
fare embarked  class  \
18          0       3  female  31.0      1      0  18.0000
S  Third
40          0       3  female  40.0      1      0   9.4750
S  Third
132         0       3  female  47.0      1      0  14.5000
S  Third
167         0       3  female  45.0      1      4  27.9000
S  Third
177         0       1  female  50.0      0      0  28.7125
C  First
254         0       3  female  41.0      0      2  20.2125
S  Third
276         0       3  female  45.0      0      0   7.7500
S  Third
362         0       3  female  45.0      0      1  14.4542
C  Third
396         0       3  female  31.0      0      0   7.8542
S  Third
503         0       3  female  37.0      0      0   9.5875
S  Third
610         0       3  female  39.0      1      5  31.2750
S  Third
638         0       3  female  41.0      0      5  39.6875
S  Third
```

```
657          0       3  female  32.0      1      1  15.5000
Q  Third
678          0       3  female  43.0      1      6  46.9000
S  Third
736          0       3  female  48.0      1      3  34.3750
S  Third
767          0       3  female  30.5      0      0   7.7500
Q  Third
885          0       3  female  39.0      0      5  29.1250
Q  Third
```

VISUALIZING A PANDAS DATASET IN SEABORN

Listing 7.17 displays the contents of `pandas_seaborn.py` that illustrate how to display a `Pandas` dataset in `Seaborn`.

LISTING 7.17: pandas_seaborn.py

```
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.Data frame()

df['x'] = random.sample(range(1, 100), 25)
df['y'] = random.sample(range(1, 100), 25)

print("top five elements:")
print(df.head())

# display a density plot
#sns.kdeplot(df.y)

# display a density plot
#sns.kdeplot(df.y, df.x)

#sns.distplot(df.x)

# display a histogram
#plt.hist(df.x, alpha=.3)
#sns.rugplot(df.x)

# display a boxplot
#sns.boxplot([df.y, df.x])

# display a violin plot
#sns.violinplot([df.y, df.x])

# display a heatmap
#sns.heatmap([df.y, df.x], annot=True, fmt="d")

# display a cluster map
#sns.clustermap(df)
```

```
# display a scatterplot of the data points
sns.lmplot('x', 'y', data=df, fit_reg=False)
plt.show()
```

Listing 7.17 contains several familiar `import` statements, followed by the initialization of the `Pandas` variable `df` as a `Pandas` data frame. The next two code snippets initialize the columns and rows of the data frame and the `print()` statement displays the first five rows.

For your convenience, Listing 7.17 contains an assortment of "commented out" code snippets that use Seaborn in order to render a density plot, a histogram, a boxplot, a violin plot, a heatmap, and a cluster. Uncomment the portions that interest you in order to see the associated plot. The output from Listing 7.17 is here:

```
top five elements:
    x   y
0  52  34
1  31  47
2  23  18
3  34  70
4  71   1
```

Figure 7.12 displays a plot of the data in the `Titanic` dataset based on the code in Listing 7.17.

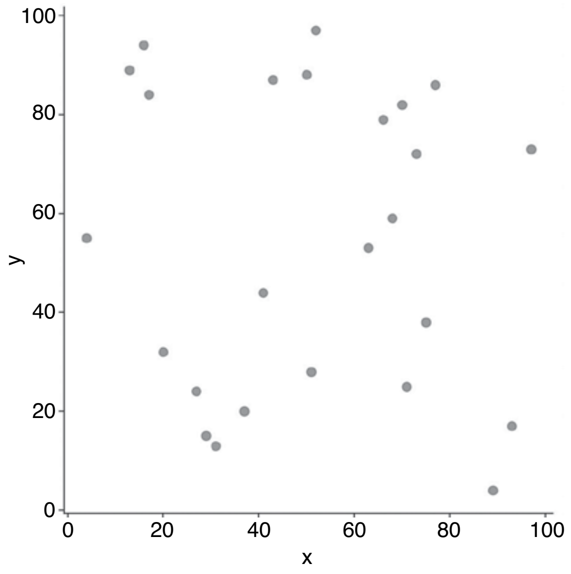

FIGURE 7.12. A Pandas data frame displayed via Seaborn.

DATA VISUALIZATION IN PANDAS

Although Matplotlib and Seaborn are often the “go to” Python libraries for data visualization, you can also use Pandas for such tasks.

Listing 7.18 displays the contents pandas_viz1.py that illustrates how to render various types of charts and graphs using Pandas and Matplotlib.

LISTING 7.18: pandas_viz1.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.Data frame(np.random.rand(16,3), columns=['X1','X2','X3'])
print("First 5 rows:")
print(df.head())
print()

print("Diff of first 5 rows:")
print(df.diff().head())
print()

# bar chart:
#ax = df.plot.bar()

# horizontal stacked bar chart:
#ax = df.plot.barh(stacked=True)

# vertical stacked bar chart:
ax = df.plot.bar(stacked=True)

# stacked area graph:
#ax = df.plot.area()

# non-stacked area graph:
#ax = df.plot.area(stacked=False)

#plt.show(ax)
```

Listing 7.18 initializes the data frame df with a 16x3 matrix of random numbers, followed by the contents of df. The bulk of Listing 7.18 contains code snippets for generating a bar chart, a horizontal stacked bar chart, a vertical stacked bar chart, a stacked area graph, and a nonstacked area graph. You can uncomment the individual code snippet that displays the graph of your choice with the contents of df. Launch the code in Listing 7.18 and you will see the following output:

```
First 5 rows:
         X1        X2        X3
0  0.051089  0.357183  0.344414
1  0.800890  0.468372  0.800668
2  0.492981  0.505133  0.228399
3  0.461996  0.977895  0.471315
4  0.033209  0.411852  0.347165

Diff of first 5 rows:
         X1        X2        X3
0       NaN       NaN       NaN
1  0.749801  0.111189  0.456255
2 -0.307909  0.036760 -0.572269
3 -0.030984  0.472762  0.242916
4 -0.428787 -0.566043 -0.124150
```

WHAT IS BOKEH?

`Bokeh` is an open source project that depends on `Matplotlib` as well as `Sklearn`. As you will see in the subsequent code sample, `Bokeh` generates an `HTML` Web page that is based on `Python` code, and then launches that Web page in a browser. `Bokeh` and `D3.js` (which is a `JavaScript` layer of abstraction over `SVG`) both provide elegant visualization effects that support animation effects and user interaction.

`Bokeh` enables the rapid creation statistical visualization, and it works with other tools with as `Python Flask` and `Django`. In addition to `Python`, `Bokeh` supports `Julia`, `Lua`, and `R` (`JSON` files are generated instead of `HTML` Web pages).

Listing 7.19 displays the contents `bokeh_trig.py` that illustrates how to create a graphics effect using various `Bokeh` APIs.

LISTING 7.19: bokeh_trig.py

```
# pip3 install bokeh
from bokeh.plotting import figure, output_file, show
from bokeh.layouts import column
import bokeh.colors as colors
import numpy as np
import math

deltaY = 0.01
maxCount = 150
width  = 800
height = 400
band_width = maxCount/3

x = np.arange(0, math.pi*3, 0.05)
y1 = np.sin(x)
y2 = np.cos(x)

white = colors.RGB(255,255,255)
```

```
fig1 = figure(plot_width = width, plot_height = height)

for i in range(0,maxCount):
  rgb1 = colors.RGB(i*255/maxCount, 0, 0)
  rgb2 = colors.RGB(i*255/maxCount, i*255/maxCount, 0)
  fig1.line(x, y1-i*deltaY,line_width = 2, line_color = rgb1)
  fig1.line(x, y2-i*deltaY,line_width = 2, line_color = rgb2)

for i in range(0,maxCount):
  rgb1 = colors.RGB(0, 0, i*255/maxCount)
  rgb2 = colors.RGB(0, i*255/maxCount, 0)
  fig1.line(x, y1+i*deltaY,line_width = 2, line_color = rgb1)
  fig1.line(x, y2+i*deltaY,line_width = 2, line_color = rgb2)
  if (i % band_width == 0):
    fig1.line(x, y1+i*deltaY,line_width = 5, line_color = white)

show(fig1)
```

Listing 7.19 starts with a commented out `pip3` code snippet that you can launch from the command line in order to install `Bokeh` (in case you haven't done so already).

The next code block contains several Bokeh-related statements as well as `NumPy` and `Math`.

Notice that the variable `white` is defined as an (R, G, B) triple of integers, which represents the red, green, and blue components of a color. In particular, (255, 255, 255) represents the color white (check online if you are unfamiliar with RGB). The next portion of Listing 7.19 initializes some scalar variables that are used in the two `for` loops that are in the second half of Listing 7.19.

Next, the `NumPy` variable `x` is a range of values from 0 to `math.PI/3`, with an increment of 0.05 between successive values. Then the `NumPy` variables `y1` and `y2` are defined as the sine and cosine values, respectively, of the values in `x`. The next code snippet initializes the variable `fig1` that represents a context in which the graphics effects will be rendered. This completes the initialization of the variables that are used in the two loops.

The next portion of Listing 7.19 contains the first `for` loop that creates a gradient-like effect by defining (R, G, B) triples whose values are based partially on the value of the loop variable `i`. For example, the variable `rgb1` ranges in a linear fashion from (0, 0, 0) to (255, 0, 0), which represent the colors black and red, respectively. The variable `rgb2` ranges in a linear fashion from (0, 0, 0) to (255, 255, 0), which represent the colors black and yellow, respectively. The next portion of the `for` loop contains two invocations of the `fig1.line()` API that renders a sine wave and a cosine wave in the context variable `fig1`.

The second `for` loop is similar to the first for loop: the main difference is that the variable `rgb1` varies from black to blue, and the variable `rgb2` variables from black to green. The final code snippet in Listing 7.19 invokes the `show()` method that generates an `HTML` Web page (with the same prefix as the `Python` file) and then launches the Web page in a browser.

Figure 7.13 displays the graphics effect based on the code in Listing 7.19. If this image is displayed as black and white, launch the code from the command line and you will see the gradient-like effects in the image.

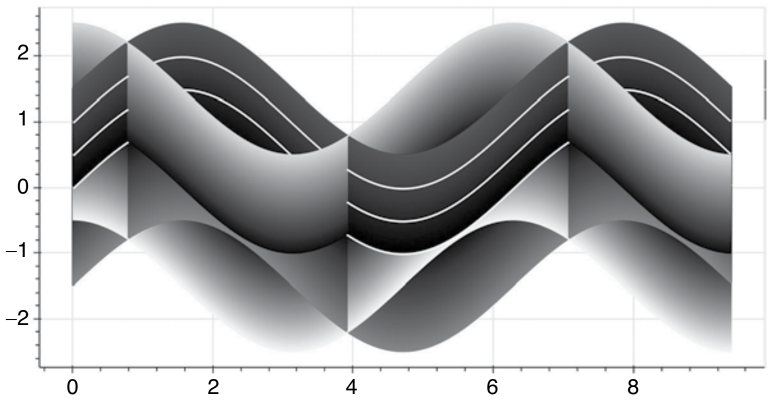

FIGURE 7.13 A Bokeh graphics sample.

SUMMARY

This chapter started with a brief introduction of a short list of data visualization tools, and a list of various types of visualization (bar graphs, pie charts, and so forth).

Then you learned about `Matplotlib`, which is an open source `Python` library that is modeled after `MatLab`. You saw some examples of plotting histograms and simple trigonometric functions.

In addition, you were introduced to `Seaborn`, which is an extension of `Matplotlib`, and you saw examples of plotting lines and histograms, and also how to plot a `Pandas` data frame using `Seaborn`.

Finally, you got a very short introduction to `Bokeh`, along with a code sample that illustrates how to create a more artistic graphics effect with relative ease in `Bokeh`.

INDEX