THE SUPERVISED LEARNING WORKSHOP

A NEW, INTERACTIVE APPROACH TO UNDERSTANDING SUPERVISED LEARNING ALGORITHMS

SECOND EDITION

BLAINE BATEMAN, ASHISH RANJAN JHA,
BENJAMIN JOHNSTON, AND ISHITA MATHUR

The Supervised Learning Workshop

Second Edition

A New, Interactive Approach to Understanding Supervised Learning Algorithms

Blaine Bateman, Ashish Ranjan Jha,
Benjamin Johnston, and Ishita Mathur

The Supervised Learning Workshop

Second Edition

Authors: Blaine Bateman, Ashish Ranjan Jha, Benjamin Johnston, and Ishita Mathur

Reviewers: Tiffany Ford, Sukanya Mandal, Ashish Pratik Patil, and Ratan Singh

Managing Editor: Snehal Tambe

Acquisitions Editor: Anindya Sil

Production Editor: Samita Warang

First published: April 2019

Second edition: February 2020

Production reference: 1280220

ISBN 978-1-80020-904-6

Published by Packt Publishing Ltd.

Livery Place, 35 Livery Street

Birmingham B3 2PB, UK

Chapter 2: Exploratory Data Analysis and Visualization 39

Chapter 3: Linear Regression 107

Chapter 4: Autoregression 195

Chapter 6: Ensemble Modeling 297

Chapter 7: Model Evaluation 333

Preface

About

This section briefly introduces this book and software requirements in order to complete all of the included activities and exercises.

About the Book

You already know you want to learn about supervised learning, and a smarter way to do that is to learn by doing. *The Supervised Learning Workshop* focuses on building up your practical skills so that you can deploy and build solutions that leverage key supervised learning algorithms. You'll learn from real examples that lead to real results.

Throughout *The Supervised Learning Workshop*, you'll take an engaging step-by-step approach to understanding supervised learning. You won't have to sit through any unnecessary theory. If you're short on time, you can jump into a single exercise each day or spend an entire weekend learning how to predict future values with various regression and autoregression models. It's your choice. Learning on your terms, you'll build up and reinforce key skills in a way that feels rewarding.

Every physical print copy of *The Supervised Learning Workshop* unlocks access to the interactive edition. With videos detailing all exercises and activities, you'll always have a guided solution. You can also benchmark yourself against assessments, track your progress, and receive content updates. You'll even earn a secure credential that you can share and verify online upon completion. It's a premium learning experience that's included with your print copy. To redeem this, follow the instructions located at the start of the book.

Fast-paced and direct, *The Supervised Learning Workshop* is the ideal companion for those with some Python background who are getting started with machine learning. You'll learn how to apply key algorithms like a data scientist, learning along the way. This process means that you'll find that your new skills stick, embedded as best practice, establishing a solid foundation for the years ahead.

Audience

Our goal at Packt is to help you be successful in whatever it is you choose to do. *The Supervised Learning Workshop* is ideal for those with a Python background who are just starting out with machine learning. Pick up a copy of *The Supervised Learning Workshop* today and let Packt help you develop skills that stick with you for life.

About the Chapters

Chapter 1, *Fundamentals of Supervised Learning Algorithms*, introduces you to supervised learning, Jupyter notebooks, and some of the most common pandas data methods.

Chapter 2, *Exploratory Data Analysis and Visualization*, teaches you how to perform exploration and analysis on a new dataset.

Chapter 3, *Linear Regression*, teaches you how to tackle regression problems and analysis, introducing you to linear regression as well as multiple linear regression and gradient descent.

Chapter 4, *Autoregression*, teaches you how to implement autoregression as a method to forecast values that depend on past values.

Chapter 5, *Classification Techniques*, introduces classification problems, classification using linear and logistic regression, k-nearest neighbors, and decision trees.

Chapter 6, *Ensemble Modeling*, teaches you how to examine the different ways of ensemble modeling, including their benefits and limitations.

Chapter 7, *Model Evaluation*, demonstrates how you can improve a model's performance by using hyperparameters and model evaluation metrics.

Conventions

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Use the pandas **`read_csv`** function to load the CSV file containing the **`synth_temp.csv`** dataset, and then display the first five lines of data."

Words that you see on screen, for example, in menus or dialog boxes, also appear in the text like this: "Open the **`titanic.csv`** file by clicking on it on the Jupyter notebook home page."

A block of code is set as follows:

```
print(data[pd.isnull(data.damage_millions_dollars)].shape[0])
print(data[pd.isnull(data.damage_millions_dollars) &
           (data.damage_description != 'NA')].shape[0])
```

New terms and important words are shown like this: "**Supervised** means that the labels for the data are provided within the training, allowing the model to learn from these labels."

Before You Begin

Each great journey begins with a humble step. Before we can do awesome things with supervised learning, we need to be prepared with a productive environment. In this section, we will see how to do that.

Installation and Setup

Jupyter notebooks are available once you install Anaconda on your system. Anaconda can be installed for Windows systems using the steps available at https://packt.live/2P4XWqI.

For other systems, navigate to the respective installation guide from https://packt.live/32tU7Ro.

These installations will be executed in the 'C' drive of your system. You can choose to change the destination.

Installing the Code Bundle

Download the code files from GitHub at https://packt.live/2TlcKDf. Refer to these code files for the complete code bundle. Make sure to copy the code bundle to the same drive as your Anaconda installation.

If you have any issues or questions about installation, please email us at **`workshops@packt.com`**.

The high-quality color images used in this book can be found at https://packt.live/2T1BX6M.

1

Fundamentals of Supervised Learning Algorithms

Overview

This chapter introduces you to supervised learning, using Anaconda to manage coding environments, and using Jupyter notebooks to create, manage, and run code. It also covers some of the most common Python packages used in supervised learning: pandas, NumPy, Matplotlib, and seaborn. By the end of this chapter, you will be able to install and load Python libraries into your development environment for use in analysis and machine learning problems. You will also be able to load an external data source using pandas, and use a variety of methods to search, filter, and compute descriptive statistics of the data. This chapter will enable you to gauge the potential impact of various issues within the data source.

Introduction

The study and application of machine learning and artificial intelligence has recently been the source of much interest and research in the technology and business communities. Advanced data analytics and machine learning techniques have shown great promise in advancing many sectors, such as personalized healthcare and self-driving cars, as well as in solving some of the world's greatest challenges, such as combating climate change (see *Tackling Climate Change with Machine Learning*: https://packt.live/2SXh8Jo).

This book has been designed to help you to take advantage of the unique confluence of events in the field of data science and machine learning today. Across the globe, private enterprises and governments are realizing the value and efficiency of data-driven products and services. At the same time, reduced hardware costs and open source software solutions are significantly reducing the barriers to entry of learning and applying machine learning techniques.

Here, we will focus on supervised machine learning (or, supervised learning for short). We'll explain the different types of machine learning shortly, but let's begin with some quick information. The now-classic example of supervised learning is developing an algorithm to distinguish between pictures of cats and dogs. The supervised part arises from two aspects; first, we have a set of pictures where we know the correct answers. We call such data labeled data. Second, we carry out a process where we iteratively test our algorithm's ability to predict "cat" or "dog" given pictures, and we make corrections to the algorithm when the predictions are incorrect. This process, at a high level, is similar to teaching children. However, it generally takes a lot more data to train an algorithm than to teach a child to recognize cats and dogs! Fortunately, there are rapidly growing sources of data at our disposal. Note the use of the words learning and train in the context of developing our algorithm. These might seem to be giving human qualities to our machines and computer programs, but they are already deeply ingrained in the machine learning (and artificial intelligence) literature, so let's use them and understand them. Training in our context here always refers to the process of providing labeled data to an algorithm and making adjustments to the algorithm to best predict the labels given the data. Supervised means that the labels for the data are provided within the training, allowing the model to learn from these labels.

Let's now understand the distinction between supervised learning and other forms of machine learning.

When to Use Supervised Learning

Generally, if you are trying to automate or replicate an existing process, the problem is a supervised learning problem. As an example, let's say you are the publisher of a magazine that reviews and ranks hairstyles from various time periods. Your readers frequently send you far more images of their favorite hairstyles for review than you can manually process. To save some time, you would like to automate the sorting of the hairstyle images you receive based on time periods, starting with hairstyles from the 1960s and 1980s, as you can see in the following figure:

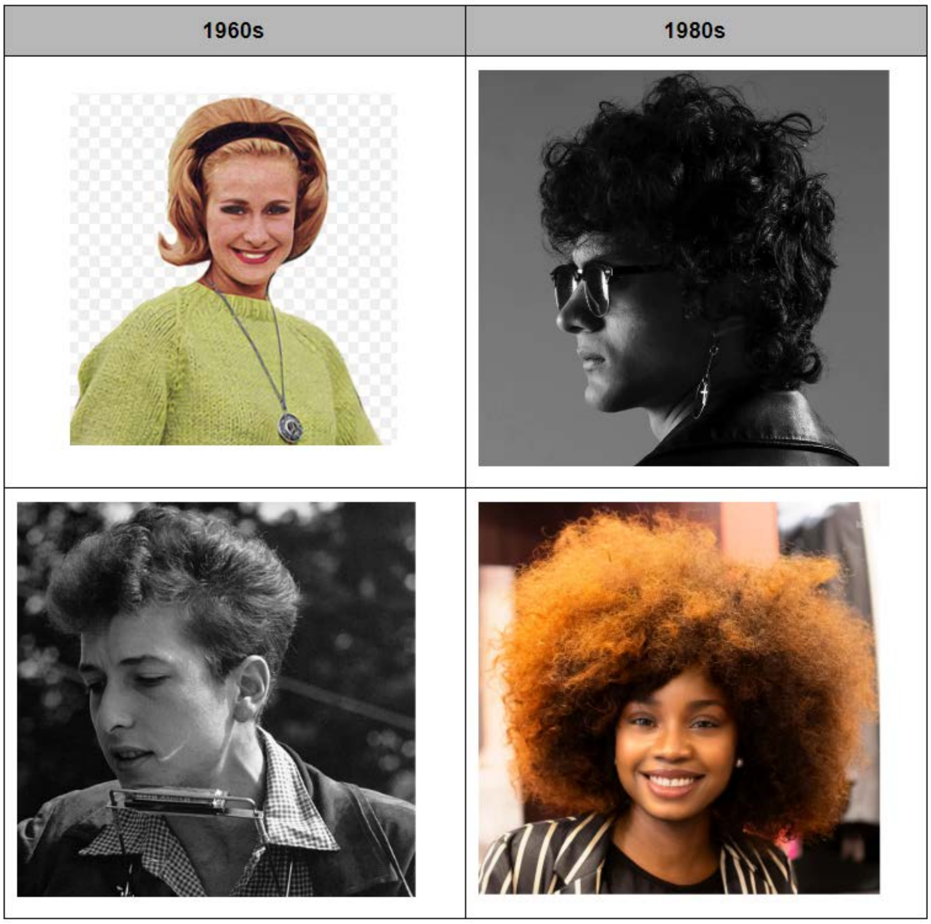

Figure 1.1: Images of hairstyles from different time periods

To create your hairstyles-sorting algorithm, you start by collecting a large sample of hairstyle images and manually labeling each one with its corresponding time period. Such a dataset (known as a labeled dataset) is the input data (hairstyle images) for which the desired output information (time period) is known and recorded. This type of problem is a classic supervised learning problem; we are trying to develop an algorithm that takes a set of inputs and learns to return the answers that we have told it are correct.

Python Packages and Modules

Python is one of the most popular programming languages used for machine learning, and is the language used here.

While the standard features that are included in Python are certainly feature-rich, the true power of Python lies in the additional libraries (also known as packages), which, thanks to open source licensing, can be easily downloaded and installed through a few simple commands. In this book, we generally assume your system has been configured using Anaconda, which is an open source environment manager for Python. Depending on your system, you can configure multiple virtual environments using Anaconda, each one configured with specific packages and even different versions of Python. Using Anaconda takes care of many of the requirements to get ready to perform machine learning, as many of the most common packages come pre-built within Anaconda. Refer to the preface for Anaconda installation instructions.

In this book, we will be using the following additional Python packages:

- NumPy (pronounced *Num Pie* and available at https://packt.live/2w1Kn4R): NumPy (short for numerical Python) is one of the core components of scientific computing in Python. NumPy provides the foundational data types from which a number of other data structures derive, including linear algebra, vectors and matrices, and key random number functionality.
- SciPy (pronounced *Sigh Pie* and available at https://packt.live/2w5Wfmm): SciPy, along with NumPy, is a core scientific computing package. SciPy provides a number of statistical tools, signal processing tools, and other functionality, such as Fourier transforms.
- pandas (available at https://packt.live/3cc4TAa): pandas is a high-performance library for loading, cleaning, analyzing, and manipulating data structures.
- Matplotlib (available at https://packt.live/2TmvKBk): Matplotlib is the foundational Python library for creating graphs and plots of datasets and is also the base package from which other Python plotting libraries derive. The Matplotlib API has been designed in alignment with the Matlab plotting library to facilitate an easy transition to Python.
- Seaborn (available at https://packt.live/2VniL4F): Seaborn is a plotting library built on top of Matplotlib, providing attractive color and line styles as well as a number of common plotting templates.

- Scikit-learn (available at https://packt.live/2MC1kJ9): Scikit-learn is a Python machine learning library that provides a number of data mining, modeling, and analysis techniques in a simple API. Scikit-learn includes a number of machine learning algorithms out of the box, including classification, regression, and clustering techniques.

These packages form the foundation of a versatile machine learning development environment, with each package contributing a key set of functionalities. As discussed, by using Anaconda, you will already have all of the required packages installed and ready for use. If you require a package that is not included in the Anaconda installation, it can be installed by simply entering and executing the following code in a Jupyter notebook cell:

```
!conda install <package name>
```

As an example, if we wanted to install Seaborn, we'd run the following command:

```
!conda install seaborn
```

To use one of these packages in a notebook, all we need to do is import it:

```
import matplotlib
```

Loading Data in Pandas

`pandas` has the ability to read and write a number of different file formats and data structures, including CSV, JSON, and HDF5 files, as well as SQL and Python Pickle formats. The pandas input/output documentation can be found at https://packt.live/2FiYB2O. We will continue to look into the **`pandas`** functionality by loading data via a CSV file.

> **Note**
>
> The dataset we will be using for this chapter is the *Titanic: Machine Learning from Disaster* dataset, available from https://packt.live/2wQPBkx.
>
> Alternatively, the dataset is available on our GitHub repository via the following link: https://packt.live/2vjyPK9

The dataset contains a roll of the guests on board the famous ship Titanic, as well as their age, survival status, and number of siblings/parents. Before we get started with loading the data into Python, it is critical that we spend some time looking over the information provided for the dataset so that we can have a thorough understanding of what it contains. Download the dataset and place it in the directory you're working in.

Looking at the description for the data, we can see that we have the following fields available:

- **`survival`**: This tells us whether a given person survived (**`0`** = No, **`1`** = Yes).
- **`pclass`**: This is a proxy for socio-economic status, where first class is upper, second class is middle, and third class is lower status.
- **`sex`**: This tells us whether a given person is male or female.
- **`age`**: This is a fractional value if less than 1; for example, 0.25 is 3 months. If the age is estimated, it is in the form of *xx*.5.
- **`sibsp`**: A sibling is defined as a brother, sister, stepbrother, or stepsister, and a spouse is a husband or wife.
- **`parch`**: A parent is a mother or father, while a child is a daughter, son, stepdaughter, or stepson. Children that traveled only with a nanny did not travel with a parent. Thus, 0 was assigned for this field.
- **`ticket`**: This gives the person's ticket number.
- **`fare`**: This is the passenger's fare.
- **`cabin`**: This tells us the passenger's cabin number.
- **`embarked`**: The point of embarkation is the location where the passenger boarded the ship.

Note that the information provided with the dataset does not give any context as to how the data was collected. The **survival**, **pclass**, and **embarked** fields are known as categorical variables as they are assigned to one of a fixed number of labels or categories to indicate some other information. For example, in **embarked**, the **C** label indicates that the passenger boarded the ship at Cherbourg, and the value of **1** in **survival** indicates they survived the sinking.

Exercise 1.01: Loading and Summarizing the Titanic Dataset

In this exercise, we will read our Titanic dataset into Python and perform a few basic summary operations on it:

1. Open a new Jupyter notebook.
2. Import the **pandas** and **numpy** packages using shorthand notation:

```
import pandas as pd
import numpy as np
```

3. Open the **titanic.csv** file by clicking on it in the Jupyter notebook home page as shown in the following figure:

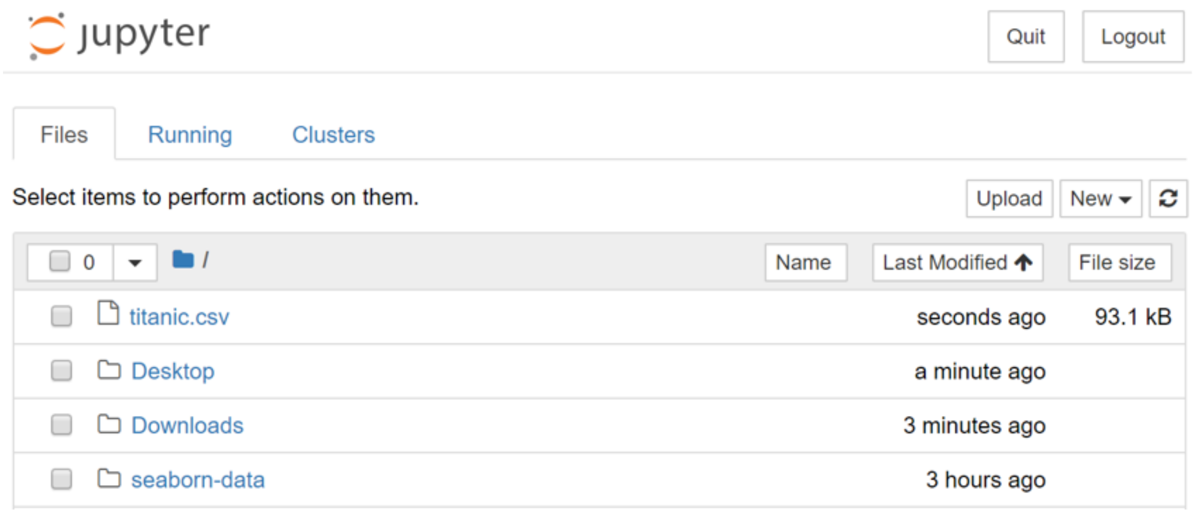

Figure 1.2: Opening the CSV file

The file is a CSV file, which can be thought of as a table, where each line is a row in the table and each comma separates columns in the table. Thankfully, we don't need to work with these tables in raw text form and can load them using **`pandas`**:

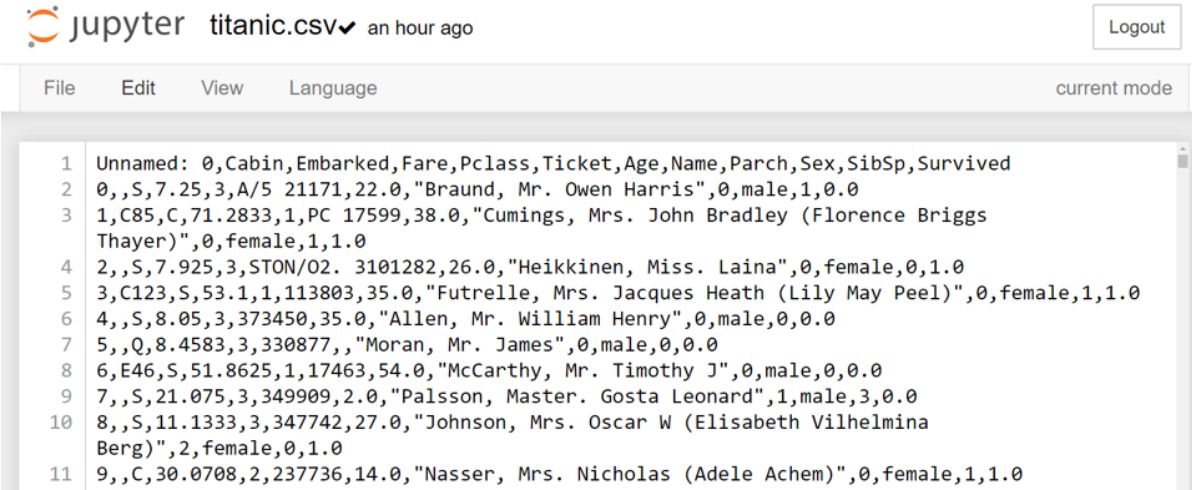

Figure 1.3: Contents of the CSV file

> **Note**
>
> Take a moment to look up the pandas documentation for the **`read_csv`** function at https://packt.live/2SkXerv. Note the number of different options available for loading CSV data into a pandas DataFrame.

4. In an executable Jupyter notebook cell, execute the following code to load the data from the file:

```
df = pd.read_csv(r'..\Datasets\titanic.csv')
```

The pandas DataFrame class provides a comprehensive set of attributes and methods that can be executed on its own contents, ranging from sorting, filtering, and grouping methods to descriptive statistics, as well as plotting and conversion.

> **Note**
>
> Open and read the documentation for pandas DataFrame objects at https://packt.live/2The2Pp.

5. Read the first five rows of data using the **`head()`** method of the DataFrame:

```
df.head(10) # Examine the first 10 samples
```

The output will be as follows:

	Unnamed: 0	Cabin	Embarked	Fare	Pclass	Ticket	Age	Name	Parch	Sex	SibSp	Survived
0	0	NaN	S	7.2500	3	A/5 21171	22.0	Braund, Mr. Owen Harris	0	male	1	0.0
1	1	C85	C	71.2833	1	PC 17599	38.0	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	female	1	1.0
2	2	NaN	S	7.9250	3	STON/O2. 3101282	26.0	Heikkinen, Miss. Laina	0	female	0	1.0
3	3	C123	S	53.1000	1	113803	35.0	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	female	1	1.0
4	4	NaN	S	8.0500	3	373450	35.0	Allen, Mr. William Henry	0	male	0	0.0
5	5	NaN	Q	8.4583	3	330877	NaN	Moran, Mr. James	0	male	0	0.0
6	6	E46	S	51.8625	1	17463	54.0	McCarthy, Mr. Timothy J	0	male	0	0.0
7	7	NaN	S	21.0750	3	349909	2.0	Palsson, Master. Gosta Leonard	1	male	3	0.0
8	8	NaN	S	11.1333	3	347742	27.0	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	2	female	0	1.0
9	9	NaN	C	30.0708	2	237736	14.0	Nasser, Mrs. Nicholas (Adele Achem)	0	female	1	1.0

Figure 1.4: Reading the first 10 rows

In this sample, we have a visual representation of the information in the DataFrame. We can see that the data is organized in a tabular, almost spreadsheet-like structure. The different types of data are organized into columns, while each sample is organized into rows. Each row is assigned an index value and is shown as the numbers **0** to **9** in bold on the left-hand side of the DataFrame. Each column is assigned to a label or name, as shown in bold at the top of the DataFrame.

The idea of a DataFrame as a kind of spreadsheet is a reasonable analogy. As we will see in this chapter, we can sort, filter, and perform computations on the data just as you would in a spreadsheet program. While it's not covered in this chapter, it is interesting to note that DataFrames also contain pivot table functionality, just like a spreadsheet (https://packt.live/38W0Bep).

Exercise 1.02: Indexing and Selecting Data

Now that we have loaded some data, let's use the selection and indexing methods of the DataFrame to access some data of interest. This exercise is a continuation of *Exercise 1.01, Loading and Summarizing the Titanic Dataset*:

1. Select individual columns in a similar way to a regular dictionary by using the labels of the columns, as shown here:

```
df['Age']
```

The output will be as follows:

```
0       22.0
1       38.0
2       26.0
3       35.0
4       35.0
        ...
1304     NaN
1305    39.0
1306    38.5
1307     NaN
1308     NaN
Name: Age, Length: 1309, dtype: float64
```

If there are no spaces in the column name, we can also use the dot operator. If there are spaces in the column names, we will need to use the bracket notation:

```
df.Age
```

The output will be as follows:

```
0       22.0
1       38.0
2       26.0
3       35.0
4       35.0
        ...
1304     NaN
1305    39.0
1306    38.5
1307     NaN
1308     NaN
Name: Age, Length: 1309, dtype: float64
```

2. Select multiple columns at once using bracket notation, as shown here:

```
df[['Name', 'Parch', 'Sex']]
```

The output will be as follows:

	Name	Parch	Sex
0	Braund, Mr. Owen Harris	0	male
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	female
2	Heikkinen, Miss. Laina	0	female
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	female
4	Allen, Mr. William Henry	0	male
...	...	...	...
1304	Spector, Mr. Woolf	0	male
1305	Oliva y Ocana, Dona. Fermina	0	female
1306	Saether, Mr. Simon Sivertsen	0	male
1307	Ware, Mr. Frederick	0	male
1308	Peter, Master. Michael J	1	male

1309 rows × 3 columns

Figure 1.5: Selecting multiple columns

> **Note**
>
> The output has been truncated for presentation purposes.

3. Select the first row using **`iloc`**:

```
df.iloc[0]
```

The output will be as follows:

```
Unnamed: 0                           0
Cabin                              NaN
Embarked                             S
Fare                              7.25
Pclass                               3
Ticket                       A/5 21171
Age                                 22
Name           Braund, Mr. Owen Harris
Parch                                0
Sex                               male
SibSp                                1
Survived                             0
Name: 0, dtype: object
```

Figure 1.6: Selecting the first row

4. Select the first three rows using **`iloc`**:

```
df.iloc[[0,1,2]]
```

The output will be as follows:

	Unnamed: 0	Cabin	Embarked	Fare	Pclass	Ticket	Age	Name	Parch	Sex	SibSp	Survived
0	0	NaN	S	7.2500	3	A/5 21171	22.0	Braund, Mr. Owen Harris	0	male	1	0.0
1	1	C85	C	71.2833	1	PC 17599	38.0	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	female	1	1.0
2	2	NaN	S	7.9250	3	STON/O2. 3101282	26.0	Heikkinen, Miss. Laina	0	female	0	1.0

Figure 1.7: Selecting the first three rows

5. Next, get a list of all of the available columns:

```
columns = df.columns # Extract the list of columns
print(columns)
```

The output will be as follows:

```
Index(['Unnamed: 0', 'Cabin', 'Embarked', 'Fare', 'Pclass', 'Ticket', 'Age',
       'Name', 'Parch', 'Sex', 'SibSp', 'Survived'],
      dtype='object')
```

Figure 1.8: Getting all the columns

6. Use this list of columns and the standard Python slicing syntax to get columns 2, 3, and 4, and their corresponding values:

```
df[columns[1:4]] # Columns 2, 3, 4
```

The output will be as follows:

	Cabin	Embarked	Fare
0	NaN	S	7.2500
1	C85	C	71.2833
2	NaN	S	7.9250
3	C123	S	53.1000
4	NaN	S	8.0500
...	...	...	...
1304	NaN	S	8.0500
1305	C105	C	108.9000
1306	NaN	S	7.2500
1307	NaN	S	8.0500
1308	NaN	C	22.3583

1309 rows × 3 columns

Figure 1.9: Getting the second, third, and fourth columns

7. Use the **`len`** operator to get the number of rows in the DataFrame:

```
len(df)
```

The output will be as follows:

```
1309
```

8. Get the value for the **Fare** column in row 2 using the row-centric method:

```
df.iloc[2]['Fare'] # Row centric
```

The output will be as follows:

```
7.925
```

9. Use the dot operator for the column, as follows:

```
df.iloc[2].Fare # Row centric
```

The output will be as follows:

```
7.925
```

10. Use the column-centric method, as follows:

```
df['Fare'][2] # Column centric
```

The output will be as follows:

```
7.925
```

11. Use the column-centric method with the dot operator, as follows:

```
df.Fare[2] # Column centric
```

The output will be as follows:

```
7.925
```

In this exercise, we have seen how to use pandas' **read_csv()** function to load data into Python within a Jupyter notebook. We then explored a number of ways that pandas, by presenting the data in a DataFrame, facilitates selecting specific items in a DataFrame and viewing the contents. With these basics understood, let's look at some more advanced ways to index and select data.

Exercise 1.03: Advanced Indexing and Selection

With the basics of indexing and selection under our belt, we can turn our attention to more advanced indexing and selection. In this exercise, we will look at a few important methods for performing advanced indexing and selecting data. This exercise is a continuation of *Exercise 1.01*, *Loading and Summarizing the Titanic Dataset*:

1. Create a list of the passengers' names and ages for those passengers under the age of 21, as shown here:

```
child_passengers = df[df.Age  < 21][['Name', 'Age']]
child_passengers.head()
```

The output will be as follows:

	Name	Age
7	Palsson, Master. Gosta Leonard	2.0
9	Nasser, Mrs. Nicholas (Adele Achem)	14.0
10	Sandstrom, Miss. Marguerite Rut	4.0
12	Saundercock, Mr. William Henry	20.0
14	Vestrom, Miss. Hulda Amanda Adolfina	14.0

Figure 1.10: List of passengers' names and ages for those passengers under the age of 21

2. Count how many child passengers there were, as shown here:

```
print(len(child_passengers))
```

The output will be as follows:

```
249
```

3. Count how many passengers were between the ages of 21 and 30. Do not use Python's **`and`** logical operator for this step, but rather the ampersand symbol (**`&`**). Do this as follows:

```
young_adult_passengers = df.loc[
    (df.Age > 21) & (df.Age < 30)
]
len(young_adult_passengers)
```

The output will be as follows:

```
279
```

4. Find the passengers who were either first- or third-class ticket holders. Again, we will not use the Python logical **`or`** operator but the pipe symbol (**`|`**) instead. Do this as follows:

```
df.loc[
    (df.Pclass == 3) | (df.Pclass ==1)
]
```

The output will be as follows:

	Unnamed: 0	Cabin	Embarked	Fare	Pclass	Ticket	Age	Name	Parch	Sex	SibSp	Survived
0	0	NaN	S	7.2500	3	A/5 21171	22.0	Braund, Mr. Owen Harris	0	male	1	0.0
1	1	C85	C	71.2833	1	PC 17599	38.0	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	female	1	1.0
2	2	NaN	S	7.9250	3	STON/O2. 3101282	26.0	Heikkinen, Miss. Laina	0	female	0	1.0
3	3	C123	S	53.1000	1	113803	35.0	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	female	1	1.0
4	4	NaN	S	8.0500	3	373450	35.0	Allen, Mr. William Henry	0	male	0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1304	NaN	S	8.0500	3	A.5. 3236	NaN	Spector, Mr. Woolf	0	male	0	NaN
1305	1305	C105	C	108.9000	1	PC 17758	39.0	Oliva y Ocana, Dona. Fermina	0	female	0	NaN
1306	1306	NaN	S	7.2500	3	SOTON/O.Q. 3101262	38.5	Saether, Mr. Simon Sivertsen	0	male	0	NaN
1307	1307	NaN	S	8.0500	3	359309	NaN	Ware, Mr. Frederick	0	male	0	NaN
1308	1308	NaN	C	22.3583	3	2668	NaN	Peter, Master. Michael J	1	male	1	NaN

Figure 1.11: The number of passengers who were either first- or third-class ticket holders

5. Find the passengers who were not holders of either first- or third-class tickets. Do not simply select those second-class ticket holders, but use the **~** symbol for the **`not`** logical operator instead. Do this as follows:

```
df.loc[
    ~((df.Pclass == 3) | (df.Pclass == 1))
]
```

The output will be as follows:

	Unnamed: 0	Cabin	Embarked	Fare	Pclass	Ticket	Age	Name	Parch	Sex	SibSp	Survived
1	1	C85	C	71.2833	1	PC 17599	38.0	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	female	1	1.0
3	3	C123	S	53.1000	1	113803	35.0	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	female	1	1.0
6	6	E46	S	51.8625	1	17463	54.0	McCarthy, Mr. Timothy J	0	male	0	0.0
9	9	NaN	C	30.0708	2	237736	14.0	Nasser, Mrs. Nicholas (Adele Achem)	0	female	1	1.0
11	11	C103	S	26.5500	1	113783	58.0	Bonnell, Miss. Elizabeth	0	female	0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1296	1296	D38	C	13.8625	2	SC/PARIS 2166	20.0	Nourney, Mr. Alfred (Baron von Drachstedt")"	0	male	0	NaN
1297	1297	NaN	S	10.5000	2	28666	23.0	Ware, Mr. William Jeffery	0	male	1	NaN
1298	1298	C80	C	211.5000	1	113503	50.0	Widener, Mr. George Dunton	1	male	1	NaN
1302	1302	C78	Q	90.0000	1	19928	37.0	Minahan, Mrs. William Edward (Lillian E Thorpe)	0	female	1	NaN
1305	1305	C105	C	108.9000	1	PC 17758	39.0	Oliva y Ocana, Dona. Fermina	0	female	0	NaN

Figure 1.12: Count of passengers who were not holders of either first- or third-class tickets

6. We no longer need the **`Unnamed: 0`** column, so delete it using the **`del`** operator:

```
del df['Unnamed: 0']
df.head()
```

The output will be as follows:

	Cabin	Embarked	Fare	Pclass	Ticket	Age	Name	Parch	Sex	SibSp	Survived
0	NaN	S	7.2500	3	A/5 21171	22.0	Braund, Mr. Owen Harris	0	male	1	0.0
1	C85	C	71.2833	1	PC 17599	38.0	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	female	1	1.0
2	NaN	S	7.9250	3	STON/O2. 3101282	26.0	Heikkinen, Miss. Laina	0	female	0	1.0
3	C123	S	53.1000	1	113803	35.0	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	female	1	1.0
4	NaN	S	8.0500	3	373450	35.0	Allen, Mr. William Henry	0	male	0	0.0

Figure 1.13: The del operator

In this exercise, we have seen how to select data from a **`DataFrame`** using conditional operators that inspect the data and return the subsets we want. We also saw how to remove a column we didn't need (in this case, the **`Unnamed`** column simply contained row numbers that are not relevant to analysis). Now, we'll dig deeper into some of the power of pandas.

Pandas Methods

Now that we are confident with some **`pandas`** basics, as well as some more advanced indexing and selecting tools, let's look at some other **`DataFrame`** methods. For a complete list of all methods available in a DataFrame, we can refer to the class documentation.

> **Note**
>
> The pandas documentation is available at https://packt.live/2The2Pp.

You should now know how many methods are available within a **`DataFrame`**. There are far too many to cover in detail in this chapter, so we will select a few that will give you a great start in supervised machine learning.

We have already seen the use of one method, **`head()`**, which provides the first five lines of the DataFrame. We can select more or fewer lines if we wish by providing the number of lines as an argument, as shown here:

```
df.head(n=20) # 20 lines
df.head(n=32) # 32 lines
```

Alternatively, you can use the **tail()** function to see a specified number of lines at the end of the DataFrame.

Another useful method is **describe**, which is a super-quick way of getting the descriptive statistics of the data within a DataFrame. We can see next that the sample size (count), mean, minimum, maximum, standard deviation, and the 25th, 50th, and 75th percentiles are returned for all columns of numerical data in the DataFrame (note that text columns have been omitted):

```
df.describe()
```

The output will be as follows:

	Unnamed: 0	Fare	Pclass	Age	Parch	SibSp	Survived
count	1309.000000	1308.000000	1309.000000	1046.000000	1309.000000	1309.000000	891.000000
mean	654.000000	33.295479	2.294882	29.881138	0.385027	0.498854	0.383838
std	378.020061	51.758668	0.837836	14.413493	0.865560	1.041658	0.486592
min	0.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	327.000000	7.895800	2.000000	21.000000	0.000000	0.000000	0.000000
50%	654.000000	14.454200	3.000000	28.000000	0.000000	0.000000	0.000000
75%	981.000000	31.275000	3.000000	39.000000	0.000000	1.000000	1.000000
max	1308.000000	512.329200	3.000000	80.000000	9.000000	8.000000	1.000000

Figure 1.14: The describe method

Note that only columns of numerical data have been included within the summary. This simple command provides us with a lot of useful information; looking at the values for **count** (which counts the number of valid samples), we can see that there are 1,046 valid samples in the **Age** category, but 1,308 in **Fare**, and only 891 in **Survived**. We can see that the youngest person was 0.17 years, the average age is 29.898, and the eldest passenger was 80. The minimum fare was £0, with £33.30 the average and £512.33 the most expensive. If we look at the **Survived** column, we have 891 valid samples, with a mean of 0.38, which means about 38% survived.

We can also get these values separately for each of the columns by calling the respective methods of the DataFrame, as shown here:

```
df.count()
```

The output will be as follows:

```
Cabin        295
Embarked    1307
Fare        1308
Pclass      1309
Ticket      1309
Age         1046
Name        1309
Parch       1309
Sex         1309
SibSp       1309
Survived     891
dtype: int64
```

But we have some columns that contain text data, such as **`Embarked`**, **`Ticket`**, **`Name`**, and **`Sex`**. So what about these? How can we get some descriptive information for these columns? We can still use **`describe`**; we just need to pass it some more information. By default, **`describe`** will only include numerical columns and will compute the 25th, 50th, and 75th percentiles, but we can configure this to include text-based columns by passing the **`include = 'all'`** argument, as shown here:

```
df.describe(include='all')
```

The output will be as follows:

	Unnamed: 0	Cabin	Embarked	Fare	Pclass	Ticket	Age	Name	Parch	Sex	SibSp	Survived
count	1309.000000	295	1307	1308.000000	1309.000000	1309	1046.000000	1309	1309.000000	1309	1309.000000	891.000000
unique	NaN	186	3	NaN	NaN	929	NaN	1307	NaN	2	NaN	NaN
top	NaN	C23 C25 C27	S	NaN	NaN	CA. 2343	NaN	Connolly, Miss. Kate	NaN	male	NaN	NaN
freq	NaN	6	914	NaN	NaN	11	NaN	2	NaN	843	NaN	NaN
mean	654.000000	NaN	NaN	33.295479	2.294882	NaN	29.881138	NaN	0.385027	NaN	0.498854	0.383838
std	378.020061	NaN	NaN	51.758668	0.837836	NaN	14.413493	NaN	0.865560	NaN	1.041658	0.486592
min	0.000000	NaN	NaN	0.000000	1.000000	NaN	0.170000	NaN	0.000000	NaN	0.000000	0.000000
25%	327.000000	NaN	NaN	7.895800	2.000000	NaN	21.000000	NaN	0.000000	NaN	0.000000	0.000000
50%	654.000000	NaN	NaN	14.454200	3.000000	NaN	28.000000	NaN	0.000000	NaN	0.000000	0.000000
75%	981.000000	NaN	NaN	31.275000	3.000000	NaN	39.000000	NaN	0.000000	NaN	1.000000	1.000000
max	1308.000000	NaN	NaN	512.329200	3.000000	NaN	80.000000	NaN	9.000000	NaN	8.000000	1.000000

Figure 1.15: The describe method with text-based columns

That's better–now we have much more information. Looking at the **`Cabin`** column, we can see that there are 295 entries, with 186 unique values. The most common values are **`C32`**, **`C25`**, and **`C27`**, and they occur 6 times (from the **`freq`** value). Similarly, if we look at the **`Embarked`** column, we see that there are 1,307 entries, 3 unique values, and that the most commonly occurring value is **`S`**, with 914 entries.

Notice the occurrence of **`NaN`** values in our **`describe`** output table. **`NaN`**, or **Not a Number**, values are very important within DataFrames as they represent missing or not available data. The ability of the pandas library to read from data sources that contain missing or incomplete information is both a blessing and a curse. Many other libraries would simply fail to import or read the data file in the event of missing information, while the fact that it can be read also means that the missing data must be handled appropriately.

When looking at the output of the **`describe`** method, you should notice that the Jupyter notebook renders it in the same way as the original DataFrame that we read in using **`read_csv`**. There is a very good reason for this, as the results returned by the **`describe`** method are themselves a pandas DataFrame and thus possess the same methods and characteristics as the data read in from the CSV file. This can be easily verified using Python's built-in **`type`** function, as in the following code:

```
type(df.describe(includes='all'))
```

The output will be as follows:

```
pandas.core.frame.DataFrame
```

Now that we have a summary of the dataset, let's dive in with a little more detail to get a better understanding of the available data.

> **Note**
>
> A comprehensive understanding of the available data is critical in any supervised learning problem. The source and type of the data, the means by which it is collected, and any errors potentially resulting from the collection process all have an effect on the performance of the final model.

Hopefully, by now, you are comfortable with using pandas to provide a high-level overview of the data. We will now spend some time looking into the data in greater detail.

Exercise 1.04: Splitting, Applying, and Combining Data Sources

We have already seen how we can index or select rows or columns from a DataFrame and use advanced indexing techniques to filter the available data based on specific criteria. Another handy method that allows for such selection is the **groupby** method, which provides a quick method for selecting groups of data at a time and provides additional functionality through the **DataFrameGroupBy** object. This exercise is a continuation of *Exercise 1.01*, *Loading and Summarizing the Titanic Dataset*:

1. Use the **groupby** method to group the data under the **Embarked** column to find out how many different values for **Embarked** there are:

```
embarked_grouped = df.groupby('Embarked')
print(f'There are {len(embarked_grouped)} Embarked groups')
```

   The output will be as follows:

```
There are 3 Embarked groups
```

2. Display the output of **embarked_grouped.groups** to find what the **groupby** method actually does:

```
embarked_grouped.groups
```

   The output will be as follows:

```
{'C': Int64Index([   1,    9,   19,   26,   30,   31,   34,   36,   39,   42,
             ...
             1260, 1262, 1266, 1288, 1293, 1295, 1296, 1298, 1305, 1308],
            dtype='int64', length=270),
 'Q': Int64Index([   5,   16,   22,   28,   32,   44,   46,   47,   82,  109,
             ...
             1206, 1249, 1271, 1272, 1279, 1287, 1290, 1299, 1301, 1302],
            dtype='int64', length=123),
 'S': Int64Index([   0,    2,    3,    4,    6,    7,    8,   10,   11,   12,
             ...
             1289, 1291, 1292, 1294, 1297, 1300, 1303, 1304, 1306, 1307],
            dtype='int64', length=914)}
```

Figure 1.16: Output of embarked_grouped.groups

   We can see here that the three groups are **C**, **Q**, and **S**, and that **embarked_grouped.groups** is actually a dictionary where the keys are the groups. The values are the rows or indexes of the entries that belong to that group.

3. Use the **iloc** method to inspect row **1** and confirm that it belongs to embarked group **C**:

```
df.iloc[1]
```

The output will be as follows:

```
Cabin                                                     C85
Embarked                                                    C
Fare                                                  71.2833
Pclass                                                      1
Ticket                                               PC 17599
Age                                                        38
Name        Cumings, Mrs. John Bradley (Florence Briggs Th...
Parch                                                       0
Sex                                                    female
SibSp                                                       1
Survived                                                    1
Name: 1, dtype: object
```

Figure 1.17: Inspecting row 1

4. As the groups are a dictionary, we can iterate through them and execute computations on the individual groups. Compute the mean age for each group, as shown here:

```
for name, group in embarked_grouped:
    print(name, group.Age.mean())
```

The output will be as follows:

```
C 32.33216981132075
Q 28.63
S 29.245204603580564
```

5. Another option is to use the **aggregate** method, or **agg** for short, and provide it with the function to apply across the columns. Use the **agg** method to determine the mean of each group:

```
embarked_grouped.agg(np.mean)
```

The output will be as follows:

	Fare	Pclass	Age	Parch	SibSp	Survived
Embarked						
C	62.336267	1.851852	32.332170	0.370370	0.400000	0.553571
Q	12.409012	2.894309	28.630000	0.113821	0.341463	0.389610
S	27.418824	2.347921	29.245205	0.426696	0.550328	0.336957

Figure 1.18: Using the agg method

So, how exactly does **agg** work and what type of functions can we pass it? Before we can answer these questions, we need to first consider the data type of each column in the DataFrame, as each column is passed through this function to produce the result we see here. Each DataFrame comprises a collection of columns of pandas series data, which, in many ways, operates just like a list. As such, any function that can take a list or a similar iterable and compute a single value as a result can be used with **agg**.

6. Define a simple function that returns the first value in the column and then pass that function through to **agg**, as an example:

```
def first_val(x):
    return x.values[0]

embarked_grouped.agg(first_val)
```

The output will be as follows:

	Cabin	Fare	Pclass	Ticket	Age	Name	Parch	Sex	SibSp	Survived
Embarked										
C	C85	71.2833	1	PC 17599	38.0	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	female	1	1.0
Q	NaN	8.4583	3	330877	NaN	Moran, Mr. James	0	male	0	0.0
S	NaN	7.2500	3	A/5 21171	22.0	Braund, Mr. Owen Harris	0	male	1	0.0

Figure 1.19: Using the .agg method with a function

In this exercise, we have seen how to group data within a DataFrame, which then allows additional functions to be applied using **.agg()**, such as to calculate group means. These sorts of operations are extremely common in analyzing and preparing data for analysis.

Quantiles

The previous exercise demonstrated how to find the mean. In statistical data analysis, we are also often interested in knowing the value in a dataset below or above which a certain fraction of the points lie. Such points are called quantiles. For example, if we had a sequence of numbers from 1 to 10,001, the quantile for 25% is the value 2,501. That is, at the value 2,501, 25% of our data lies below that cutoff. Quantiles are often used in data visualizations because they convey a sense of the distribution of the data. In particular, the standard boxplot in Matplotlib draws a box bounded by the first and third of 4 quantiles.