4.2.4

Collecting Data from a CSV File

To learn how to collect data from a CSV file, we'll read data from a CSV file into a Jupyter Notebook file and then store it in a container—specifically, in a Pandas DataFrame.

NOTE

If you want to follow along, download the CSV file by clicking the following link: **new_student_data.csv (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/v2/module_4/new_student_data.csv)** .

First, we launch Jupyter Notebook. Then, we'll read the CSV file. Finally, we'll check the data that we read.

Read a CSV File

To read a CSV file into a Pandas DataFrame, we first need to import the libraries that we need to run our code. Previously, we used the Python `csv` library to import spreadsheet data. In this module, we'll use the Pandas library to read data into a DataFrame. Because Pandas is both designed for data analysis and compatible with many types of files (including CSV files), we'll find it straightforward to read our data.

To import the Pandas library, we use the following code:

```
import pandas as pd
import os
```

To then read a CSV file into a DataFrame, we use only a single function, named `read_csv`. This function accepts a path to the location of the CSV file and automatically reads the file, importing all the data into a Pandas DataFrame.

Say that we need to read a CSV file containing student data, named `new_student_data.csv`, that exists in a folder named `Resources`. To do so, we use the following code:

```
import pandas as pd
import os

student_data = os.path.join('..', 'Resources', 'new_student_data.csv')
student_df = pd.read_csv(student_data)
```

Notice that we used just two lines of code to find our CSV file and read its content into a DataFrame.

Also notice that the preceding code uses a relative path for the path to the location of the CSV file. A **relative path** refers to a location that's relative to the current directory. Relative paths make use of two special symbols: the dot (`.`) and the double dot (`..`). These translate into the current directory and the parent directory, respectively. We use double dots to move up in the directory hierarchy.

The following table lists various relative paths that we can use:

```
.    = this directory
..   = the parent directory
../  = the parent directory
~/   = the user's home directory or the application's
/    = the root directory
../../ = the parent's parent directory
```

Check the Data

Now that we've read in our data, we need to make sure that Pandas did so correctly. To do this, we use the Pandas `head` function, which shows us the first five lines of a DataFrame. This will give us a brief overview of the data, which we can use to check if the data seems as expected.

Here's the code:

```
import pandas as pd
from os

student_data = os.path.join('..', 'Resources', 'new_student_data.csv')
student_df = pd.read_csv(student_data)
student_df.head()
```

Running the preceding code produces the output that the following image shows:

	student_id	student_name	grade	school_name	reading_score	math_score	school_type
0	127008367	Sarah Douglas	11th	Chang High School	87.2	64.1	Public
1	33365505	Francisco Osborne	9th	Fisher High School	NaN	NaN	Public
2	44359500	Ryan Haas	12th	Campbell High School	91.6	54.7	Public
3	24791243	Kathryn Mack	11th	Richard High School	68.9	73.3	Charter
4	121467881	Harold Reynolds	12th	Chang High School	68.7	43.4	Public

In the preceding image, notice that the output of the `student_data.csv` import shows a header row that contains the column names. The headers in the CSV file define these column names: "student_id", "student_name", "grade", "school_name", "reading_score", "math_score", and "school_type". Rows of data follow the header row and contain information that corresponds to each column header.

IMPORTANT

Each column—which includes the header and the rows of data that follow—makes up a Pandas Series. The group of Series together forms the DataFrame.

Notice also that a column of numbers, starting with 0, precedes the output (on the left side). This is the index column. Unless the code indicates which column from the CSV file to set as the index, Pandas automatically generates this index column, which doesn't have a header.

NOTE

If an issue with the data import occurs, an error message will appear that supplies details. Perhaps the file path has an error, or a library didn't properly import. Pandas does a terrific job at supplying error information. And with a bit of practice, you'll be able to debug errors from the error messages.

So with just three lines of code, we read our CSV file and reviewed the output.

SHOW PRO TIP

To review a single column of the DataFrame, we use brackets—just like we would to select a value from a Python dictionary. For example, the following code selects the “student_name” column:

```
student_df["student_name"]
```

Running the preceding code produces output that consist of the index column followed by the “student_name” column (without the header row), as the following image shows:

```
0              Sarah Douglas
1          Francisco Osborne
2                  Ryan Haas
3              Kathryn Mack
4            Harold Reynolds
                   ...
13935            Kelly Myers
13936         Kimberly Burke
13937        Crystal Merritt
13938          Misty Wiggins
13939          Michele Jones
Name: student_name, Length: 13940, dtype: object
```

Using Pandas gives us shortcuts for data analysis. And, there’s no time like the present to start putting those shortcuts to work!

So now that you’ve learned about collecting data from a CSV file, you’ll have the opportunity to do so yourself in the following activity.