Chapter 7 covered the visualization of graphs using the Python library networkx. This chapter focuses on the basics of the data science and analytics library of SciPy, pandas. First, we will explore the data structures in this library. You will also learn how to read the data from a .csv data set. Finally, you will learn how to create simple demonstrations of visualizations. These are the topics that are covered in the chapter:

- Introduction to pandas
- Dataframe in pandas
- Visualizing with pandas

After you complete this chapter, you should be comfortable with basic visualizations using pandas.

Introduction to Pandas

Pandas is the data analytics component and an integral part of the SciPy ecosystem. It includes very versatile data structures and routines to manage them. It also has the capability to visualize the data in a scientific data format.
The first step is to install it on the computer by running the following command in Jupyter Notebook:

!pip3 install pandas

You can import it to the current session by running the following command:

import pandas as pd

You can read more about pandas at https://pandas.pydata.org/.

**Series in Pandas**

A series is a one-dimensional array with labels. It can hold data of any type. The labels are collectively known as the index.

We can create a series as follows:

```python
s1 = pd.Series([1, 2, 3, 5, -3])
```

To determine its datatypes, use the following command:

```python
type(s1)
```

You can see the values and index associated with them using the following statement:

```python
print(s1)
```

You can explicitly mention the datatype as follows:

```python
s2 = pd.Series([1, 2, 3, 5, -3], dtype=np.int32)
print(s2)
```
To pass a list as an argument to the constructor function to create a series, use this code:

```python
x = [1, 2, 3, 5, -3]
s3 = pd.Series(x)
```

We can even pass a NumPy ndarray as an argument to the constructor function to create a series as follows:

```python
import numpy as np
y = np.array(x)
s4 = pd.Series(y)
```

To display the values, use this code:

```python
print(s4.values)
```

This provides the following output:

```
[ 1  2  3  5 -3]
```

You can retrieve the index as follows:

```python
print(s4.index)
```

This results in the following output:

```
RangeIndex(start=0, stop=5, step=1)
```

To assign a custom index, use the following code:

```python
s5 = pd.Series(x, index = ['a', 'b', 'c', 'd', 'e'])
print(s5)
```

This provides the following output:

```
da    1
b     2
c     3
```

Chapter 8  Getting Started with Pandas
Basic Operations on Series

We can perform a few basic operations on series. For example, we can display the negative numbers as follows:

```python
print(s5[s5 < 0])
```

This results in the following output:

```
  e   -3
dtype: int64
```

We can also retrieve the positive numbers as follows:

```python
print(s5[s5 > 0])
```

The output is shown here:

```
a    1
b    2
c    3
d    5
dtype: int64
```

These are examples of comparison operations. To perform a multiplication operation, use this syntax:

```python
c = 3
print(s5 * c)
```
The output is as follows:

|   | 3  |
|---|----|
| a | 3  |
| b | 6  |
| c | 9  |
| d | 15 |
| e | -9 |

dtype: int64

Dataframes in Pandas

A dataframe is a two-dimensional labeled data structure with columns of that can be of different datatypes. We can create dataframes from series, ndarrays, lists, and dictionaries.

Dataframes have labels and they are collectively referred to as an index. We can easily view and manipulate data in the dataframes. The data is stored in a rectangular grid format in dataframes.

We can create a dataframe from a list of dictionary data as follows:

data = {'city': ['Mumbai', 'Mumbai', 'Mumbai',
                 'Hyderabad', 'Hyderabad', 'Hyderabad'],
        'year': [2010, 2011, 2012, 2010, 2011, 2012],
        'population': [10.0, 10.1, 10.2, 5.2, 5.3, 5.5]}

to create a dataframe from this, use the following code:

df1 = pd.DataFrame(data)
print(df1)
The output is as follows:

| city      | year | population |
|-----------|------|------------|
| Mumbai    | 2010 | 10.0       |
| Mumbai    | 2011 | 10.1       |
| Mumbai    | 2012 | 10.2       |
| Hyderabad | 2010 | 5.2        |
| Hyderabad | 2011 | 5.3        |
| Hyderabad | 2012 | 5.5        |

Use the following line of code to display the top five records:

df1.head()

The output is as follows:

| city      | year | population |
|-----------|------|------------|
| Mumbai    | 2010 | 10.0       |
| Mumbai    | 2011 | 10.1       |
| Mumbai    | 2012 | 10.2       |
| Hyderabad | 2010 | 5.2        |
| Hyderabad | 2011 | 5.3        |

You can also pass other numbers as arguments to the function head() and it will show that number of the top records from the dataframe. Similarly, you can use df1.tail() to show the last records. It has 5 as the default argument, but you can customize the argument passed to it.

You can create a dataframe with a particular order of columns as follows:

```python
df2 = pd.DataFrame(data, columns=['year', 'city', 'population'])
print(df2)
```
This results in the following output:

| year | city      | population |
|------|-----------|------------|
| 0    | Mumbai    | 10.0       |
| 1    | Mumbai    | 10.1       |
| 2    | Mumbai    | 10.2       |
| 3    | Hyderabad | 5.2        |
| 4    | Hyderabad | 5.3        |
| 5    | Hyderabad | 5.5        |

Next let's create a dataframe with an additional column and custom index:

```python
df3 = pd.DataFrame(data, columns=['year', 'city', 'population', 'GDP'],
                   index=['one', 'two', 'three', 'four', 'five', 'six'])
print(df3)
```

The following is the new dataframe:

| year | city      | population | GDP  |
|------|-----------|------------|------|
| one  | Mumbai    | 10.0       | NaN  |
| two  | Mumbai    | 10.1       | NaN  |
| three| Mumbai    | 10.2       | NaN  |
| four | Hyderabad | 5.2        | NaN  |
| five | Hyderabad | 5.3        | NaN  |
| six  | Hyderabad | 5.5        | NaN  |

Use the following command to print the list of columns:

```python
print(df3.columns)
```

Here is the resulting output:

Index(['year', 'city', 'population', 'GDP'], dtype='object')
We can print the list of indexes as follows:

```python
print(df3.index)
```

That provides the following output:

```python
Index(['one', 'two', 'three', 'four', 'five', 'six'],
dtype='object')
```

You can display the data of a column with the following statement:

```python
print(df3.year)
```

Alternatively, you can also use the following statement:

```python
print(df3['year'])
```

Here is the output:

|    |   |
|----|---|
| one| 2010 |
| two| 2011 |
| three| 2012 |
| four| 2010 |
| five| 2011 |
| six| 2012 |

Name: year, dtype: int64

You can display the datatype of a column with the following statement:

```python
print(df3['year'].dtype)
```

You can also use this code:

```python
print(df3.year.dtype)
```

That provides the following output:

```
int64
```
To display the datatype of all the columns, use this statement:

```python
print(df3.dtypes)
```

The output is as follows:

|       |       |
|-------|-------|
| year  | int64 |
| city  | object|
| population | float64 |
| GDP   | object|

dtype: object

We can retrieve any record using the index as follows:

```python
df3.loc['one']
```

Here is the resulting output:

|       |       |
|-------|-------|
| year  | 2010  |
| city  | Mumbai|
| population | 10 |
| GDP   | NaN   |

Name: one, dtype: object

You can assign the same value to all the members of a column as follows:

```python
df3.GDP = 10
print(df3)
```

The output is shown here:

|       |       |     |     |
|-------|-------|-----|-----|
| year  | city  | population | GDP |
| one   | 2010  | 10.0 | 10  |
| two   | 2011  | 10.1 | 10  |
| three | 2012  | 10.2 | 10  |
| four  | 2010  | 5.2  | 10  |
We can assign an ndarray to the GDP column as follows:

```python
import numpy as np
df3.GDP = np.arange(6)
print(df3)
```

That gives the following output:

| year | city     | population | GDP |
|------|----------|------------|-----|
| one  | 2010     | Mumbai     | 10.0| 0   |
| two  | 2011     | Mumbai     | 10.1| 1   |
| three| 2012     | Mumbai     | 10.2| 2   |
| four | 2010     | Hyderabad  | 5.2 | 3   |
| five | 2011     | Hyderabad  | 5.3 | 4   |
| six  | 2012     | Hyderabad  | 5.5 | 5   |

You can also assign it a list as follows:

```python
df3.GDP = [3, 2, 0, 9, -0.4, 7]
print(df3)
```

The output is as follows:

| year | city     | population | GDP |
|------|----------|------------|-----|
| one  | 2010     | Mumbai     | 10.0| 3.0 |
| two  | 2011     | Mumbai     | 10.1| 2.0 |
| three| 2012     | Mumbai     | 10.2| 0.0 |
| four | 2010     | Hyderabad  | 5.2 | 9.0 |
| five | 2011     | Hyderabad  | 5.3 | -0.4|
| six  | 2012     | Hyderabad  | 5.5 | 7.0 |
Let’s assign a series to it as shown here:

```python
val = pd.Series([-1.4, 1.5, -1.3], index=['two', 'four', 'five'])
df3.GDP = val
print(df3)
```

The following output is the result:

| year | city      | population | GDP  |
|------|-----------|------------|------|
| one  | Mumbai    | 10.0       | NaN  |
| two  | Mumbai    | 10.1       | -1.4 |
| three| Mumbai    | 10.2       | NaN  |
| four | Hyderabad | 5.2        | 1.5  |
| five | Hyderabad | 5.3        | -1.3 |
| six  | Hyderabad | 5.5        | NaN  |

**Reading Data Stored in CSV Format**

We can read the data stored in a comma-separated value (CSV) format with the method `read_csv()`. The CSV file could be stored at a remote URL or a location on the disk. Here is an example of reading a CSV file hosted on a URL over the Internet:

```python
df = pd.read_csv('https://raw.githubusercontent.com/cs109/2014_data/master/countries.csv')
df.head(5)
```

The output is as follows:

| Country  | Region |
|----------|--------|
| 0        | Algeria|
| 1        | Angola |
| 2        | Benin  |
| 3        | Botswana|
| 4        | Burkina|
We can read the data stored in a file on the disk into a dataframe in the same way.

**Visualizing with Pandas**

A pandas dataframe has methods for visualization like Matplotlib. Basically, these methods are wrappers over the methods in Matplotlib. Let’s look at how we can use these methods to visualize the data stored in the pandas dataframes.

Let’s import the pyplot module of Matplotlib and use the magic command to enable Matplotlib visualization in the notebook as follows:

```python
%matplotlib inline
import matplotlib.pyplot as plt
```

The NumPy function `randn()` generates a NumPy array with a standard normal random distribution. We will use this to create a series and plot it as follows:

```python
ts = pd.Series(np.random.randn(5))
ts
```

The output is as follows (it will be different every time you execute the code, as it is randomly generated at the time of execution):

```
  0   0.257543
  1   1.405170
  2   1.290728
  3   0.068451
  4  -0.923677
dtype: float64
```
This can be visualized as follows:

```python
ts.plot()
plt.grid(True)
plt.show()
```

It produces the output shown in Figure 8-1.

![Figure 8-1. Simple plot](image)

We can compute the cumulative sum with the function `cumsum()` and visualize it as follows:

```python
ts.cumsum().plot()
plt.grid(True)
plt.show()
```
The output is displayed in Figure 8-2.

**Figure 8-2. Simple plot of the cumulative sum**

Now let's create a dataframe as follows:

```python
df1 = pd.DataFrame(np.random.randn(10, 4),
                   columns=['A', 'B', 'C', 'D'])
print(df1)
```

The following output is the result:

```
   A         B         C         D
0  0.474219  1.821673 -0.296638 -0.566934
1  1.820044  2.199264  2.196097  0.203744
2  0.086325 -1.056730  0.937690 -1.283733
3  0.087798  1.145512 -0.407545  0.747684
4 -0.179241  0.290476  1.823487 -0.059593
5  1.964211 -0.525957  1.615896  0.046840
6 -0.179241  0.290476  1.823487 -0.059593
7  1.964211 -0.525957  1.615896  0.046840
8 -0.463331  0.032999  1.130027  1.151667
9 -1.212102 -0.610992  0.258653 -1.885551
```

130
We can visualize this with vertical bar graphs as follows:

df1.plot.bar()
plt.grid(True)
plt.show()

The output is displayed in Figure 8-3.

Figure 8-3. Vertical bar plot

You can even use horizontal bar graphs as follows:

df1.plot.barh()
plt.grid(True)
plt.show()
Figure 8-4 displays the output.

![Horizontal bar plot](image)

**Figure 8-4. Horizontal bar plot**

Let’s create another dataframe with similar dimensions. We will use the function `rand()` that generates a NumPy ndarray using uniform normal distribution. Here is an example:

```python
df2 = pd.DataFrame(np.random.rand(10, 4), columns=['A', 'B', 'C', 'D'])
print(df2)
```

The output is shown here:

| A         | B         | C         | D         |
|-----------|-----------|-----------|-----------|
| 0.416596  | 0.725513  | 0.707631  | 0.286249  |
| 0.166804  | 0.370956  | 0.680678  | 0.938911  |
| 0.330940  | 0.426264  | 0.667221  | 0.741184  |
| 0.879112  | 0.409153  | 0.460051  | 0.968562  |
| 0.248149  | 0.021732  | 0.072309  | 0.186000  |
| 0.666609  | 0.692510  | 0.574111  | 0.519540  |
| 0.178994  | 0.437883  | 0.036931  | 0.063519  |
| 0.057269  | 0.079832  | 0.025361  | 0.150671  |

132
We can create stacked vertical bar graphs as follows:

```python
df2.plot.bar(stacked=True)
plt.grid(True)
plt.show()
```

That produces the output shown in Figure 8-5.

![Vertical stacked bar plot](image)

**Figure 8-5. Vertical stacked bar plot**

Similarly, we can produce horizontal stacked bar plots as follows:

```python
df2.plot.barh(stacked=True)
plt.grid(True)
plt.show()
```
The output is displayed in Figure 8-6.

Figure 8-6. Horizontal stacked bar plot

To create an area plot, use this code:

df2.plot.area()
plt.grid(True)
plt.show()

By default, the area plot is stacked, as shown in Figure 8-7.

Figure 8-7. Stacked area plot
You can create an unstacked overlapping area plot as follows:

df2.plot.area(stacked=False)
plt.grid(True)
plt.show()

That produces the output shown in Figure 8-8.

![Overlapping area plot](image)

**Figure 8-8. Overlapping area plot**

Next let's demonstrate a pie chart. Create a simple dataframe as follows:

df3 = pd.DataFrame(np.random.rand(4), index=['A', 'B', 'C', 'D'])
print(df3)

The following is the output:

|     |     |     |     |
|-----|-----|-----|-----|
|     | 0.292772 | 0.569819 | 0.835805 | 0.479885 |
Use this code to create the pie chart:

df3.eplot.pie(subplots=True)
plt.show()

The result is shown in Figure 8-9.

![Pie chart](image)

**Figure 8-9. Pie chart**

**Summary**

This chapter explored the basics of the data science library of SciPy, pandas. You learned the basics of creation and operations on the fundamental pandas data structures, series and dataframes. You also learned the basics of visualizing dataframe data and how to read the data from a CSV file.

All these concepts will be very useful to in completing the exercise in the next chapter. The next chapter concludes our data visualization journey by working with data related to the ongoing COVID-19 pandemic. It provides an interesting real-life case study.