CHAPTER 9

Working with COVID-19 Data

Chapter 8 covered the basics of the data science library of SciPy, pandas. You learned the basics of the series and dataframe data structures and how to visualize the data in the dataframes and series.

This chapter is the culmination of all the knowledge you have gained in the earlier chapters. In this chapter, we are going to retrieve real-life data and visualize it. At the conclusion of this chapter, you should be comfortable with visualizations of a real-life data set.

The COVID-19 Pandemic and the Data Set

At the time of writing of this book, the world is facing an unprecedented natural calamity, a pandemic (an infectious disease spreading across continents) caused by the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) and the resulting disease known as COVID-19. This virus comes from the same family of viruses (Coronaviruses) that cause nonlethal diseases like the common cold and more lethal diseases like SARS and MERS. These viruses cause infections of the respiratory tract in mammals and humans and can be lethal if not treated in a timely manner.
Data Sources for COVID-19 Data

Many organizations are keeping track of COVID-19 cases worldwide and updating the data on their website and web services periodically. The most prominent are Johns Hopkins University (https://coronavirus.jhu.edu/map.html) and World-O-Meter (https://www.worldometers.info/coronavirus/). These are very reliable sources of data for COVID-19 and they update their statistics very frequently (at least once every 24 hours) so downstream systems get the latest data.

We can retrieve this data using custom libraries in Python. One such library can be found at https://ahmednafies.github.io/covid/. It can retrieve the data from both Johns Hopkins University and World-O-Meter. To install it, create a new notebook for this chapter and run the following command in a code cell:

!pip3 install covid

Next, import the library as follows:

from covid import Covid

You can fetch the data using this code:

covid = Covid()

It fetches the data from Johns Hopkins University by default. You can also explicitly mention the data source:

covid = Covid(source="john_hopkins")

To fetch the data from World-O-Meter, change the source value:

covid = Covid(source="worldometers")

You can display all the data using the following command:

covid.get_data()
This returns a list of dictionaries, as shown in Figure 9-1.

![Figure 9-1. COVID-19 data](image)

We can determine the source of the data as follows:

```python
covid.source
```

The output in this case is shown here:

'worldometers'

You can also retrieve the status by country name as follows:

```python
covid.get_status_by_country_name("italy")
```
The result is shown in Figure 9-2.

```python
In [37]: covid.get_status_by_country_name("italy")
Out[37]: {'country': 'Italy',
          'confirmed': 241956,
          'new_cases': 0,
          'deaths': 34899,
          'recovered': 192815,
          'active': 14242,
          'critical': 70,
          'new_deaths': 0,
          'total_tests': 5703673,
          'total_tests_per_million': Decimal('0'),
          'total_cases_per_million': Decimal('4002'),
          'total_deaths_per_million': Decimal('577'),
          'population': Decimal('60459826')}
```

**Figure 9-2. COVID-19 data by country**

You can retrieve the data by country ID, too (this function is only valid for the Johns Hopkins data source), with this code:

```python
covid.get_status_by_country_id(115)
```

To retrieve the list of countries affected by the COVID-19 pandemic, use this syntax:

```python
covid.list_countries()
```
It returns the list shown in Figure 9-3.

```
Out[17]: ['north america',
          'south america',
          'asia',
          'europe',
          'africa',
          'oceania',
          '',
          'world',
          'usa',
          'brazil',
          'india',
          'russia',
          'peru',
          'chile',
          'spain',
```

**Figure 9-3. Countries affected by COVID-19**

The total number of active cases can be obtained as follows:

```python
covid.get_total_active_cases()
```

The total number of confirmed cases can be obtained as follows:

```python
covid.get_total_confirmed_cases()
```

The total number of recovered cases can be obtained as follows:

```python
covid.get_totalRecovered()
```

The total number of deaths can be obtained as follows:

```python
covid.get_total_deaths()
```

Run those statements and examine the output.
Visualizing the COVID-19 Data

Now you can convert all this data into a pandas dataframe as follows:

```python
import pandas as pd
df = pd.DataFrame(covid.get_data())
print(df)
```

The output is shown in Figure 9-4.

```
   country        confirmed  new_cases  deaths  recovered  active
0  North America   3628797       7809    178674    1648713   1801410
1  South America   2614931      1036     96832    1717350   800749
2     Asia         2769746     20485     64867    1839062   796817
3     Europe      2513631      8954    194782    1452217   856632
4     Africa       511949       779     12026    248751    251172
```

**Figure 9-4. COVID-19 data converted to a dataframe**

Use this code to sort the data:

```python
sorted = df.sort_values(by=['confirmed'], ascending=False)
```

This data contains the cumulative data for all the continents and the world, too. We can exclude the data for the world and the continents as follows:

```python
excluded = sorted[~sorted.country.isin(['Europe', 'South America', 'Asia', 'World', 'North America', 'Africa'])]
```

142
The top 10 countries can be retrieved as follows:

```python
top10 = excluded.head(10)
print(top10)
```

The output is shown in Figure 9-5.

![Top 10 countries affected by COVID-19](image)

**Figure 9-5.** Top 10 countries affected by COVID-19

Now we can extract the data into variables as follows:

```python
x = top10.country
y1 = top10.confirmed
y2 = top10.active
y3 = top10.deaths
y4 = top10.recovered
```

Now, we can use the results to visualize. Import Matplotlib and enable plotting on the notebook with this command:

```python
%matplotlib inline
import matplotlib.pyplot as plt
```
Create a simple line graph as follows:

```python
plt.plot(x, y1)
plt.xticks(rotation=90)
plt.show()
```

The result is shown in Figure 9-6.

**Figure 9-6.** Top 10 countries affected by COVID-19 (line graph)

We can display the bar graph as follows:

```python
plt.bar(x, y1)
plt.xticks(rotation=90)
plt.show()
```
Figure 9-7 displays the result.

**Figure 9-7.** *Top 10 countries affected by COVID-19 (bar graph)*

Use this code to create a multiline graph:

```python
plt.plot(x, y1, label='Confirmed')
plt.plot(x, y2, label='Active')
plt.plot(x, y3, label='Deaths')
plt.plot(x, y4, label='Recovered')
plt.legend(loc='upper right')
plt.xticks(rotation=90)
plt.show()
```
The output is displayed in Figure 9-8.

![Figure 9-8. Top 10 countries affected by COVID-19 (multiline graph)](image)

We can rewrite the preceding example with compact code and better formatting as follows:

```python
labels = ['Confirmed', 'Active', 'Deaths', 'Recovered']
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.legend(labels, loc='upper right')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```
The result is shown in Figure 9-9.

**Figure 9-9.** Top 10 countries affected by COVID-19 (multiline graph with grids)

We can display the data in vertical multiple bar graphs as follows:

```python
df2 = pd.DataFrame([y1, y2, y3, y4])
df2.plot.bar();
plt.legend(x, loc='upper center')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```
Figure 9-10 displays the output.

**Figure 9-10.** Top 10 countries’ COVID-19 statistics (multiple bar graph with grids)

We can plot the data in vertical stacked bar graphs with the following code:

```python
df2.plot.bar(stacked=True);
plt.legend(x, loc='upper center')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```
That produces the output shown in Figure 9-11.

**Figure 9-11.** Top 10 countries’ COVID-19 statistics (vertical stacked bar graph with grids)

We can create horizontal bar graphs as follows:

df2.plot.barh();
plt.legend(x, loc='upper right')
plt.xticks(rotation=90)
plt.grid()
plt.show()
This produces the output shown in Figure 9-12.

![Figure 9-12. Top 10 countries’ COVID-19 statistics (horizontal graph with grid)](image)

Next, create a stacked horizontal bar visualization with the following code:

```python
df2.plot.barh(stacked=True);
plt.legend(x, loc='upper right')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```
The output is shown in Figure 9-13.

![Top 10 countries’ COVID-19 statistics (horizontal stacked bar graph with grid)](image)

Figure 9-13. Top 10 countries’ COVID-19 statistics (horizontal stacked bar graph with grid)

We can even use area graphs to visualize this data. By default, the area graph, as we know, is stacked. We can create it as follows:

```python
df2.plot.area();
plt.legend(x, loc='upper right')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```
The output is displayed in Figure 9-14.

![Figure 9-14. Top 10 countries’ COVID-19 statistics (stacked area graph)](image)

You can even create an overlapping area graph as follows:

```python
df2.plot.area(stacked=False);
plt.legend(x, loc='upper right')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```
Figure 9-15 depicts the result.

![Figure 9-15. Top 10 countries’ COVID-19 statistics (overlapping area graph)](image)

To create a nice scatter plot where the size of dots is proportional to the magnitude of data, use the following code:

```python
factor=0.0001
plt.scatter(x, y1, s=y1*factor);
plt.scatter(x, y2, s=y2*factor);
plt.scatter(x, y3, s=y3*factor);
plt.scatter(x, y4, s=y4*factor);
plt.legend(labels, loc='upper right')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```
The output is shown in Figure 9-16.

Finally, you can create a nice pie chart with the following code:

```python
plt.pie(y1, labels=x)
plt.title('Confirmed Cases')
plt.show()
```

**Figure 9-16.** Top 10 countries’ COVID-19 statistics (scatter graph)
The output is shown in Figure 9-17.

![Pie Chart: Top 10 countries’ COVID-19 statistics](image)

**Figure 9-17.** *Top 10 countries’ COVID-19 statistics (pie chart)*

**Summary**

In this chapter, you learned how to retrieve COVID-19 data from various online sources and convert the data to a pandas dataframe. You also saw how to prepare various types of visualizations to represent that COVID-19 data graphically.

This is how we conclude our data visualization journey, working with the real-life data. I hope you enjoyed reading and following the examples in this book as much as I enjoyed writing them. Data visualization is an expansive domain. There are many other Python libraries for data visualization that you can explore, such as ggplot, plotly, and seaborn. These libraries provide advanced data visualization capabilities that might meet your business and scientific visualization requirements.