9
Plotting, Visualization, and Storytelling

This chapter will teach you how to visualize data by exploring additional chart options such as histograms, box plots, and scatter plots to advance your data literacy skills. Storytelling with data starts with understanding the relationships that exist within the numbers, so we will learn about distribution curves and how they apply to analysis. During this discovery phase of analysis of your data, you will learn how to identify outliers and patterns along with best practices in visualizing geographic data. We will wrap up this chapter by learning the difference between correlation versus causation.

We will cover the following topics in this chapter:

- Explaining distribution analysis
- Understanding outliers and trends
- Geoanalytical techniques and tips
- Finding patterns in data

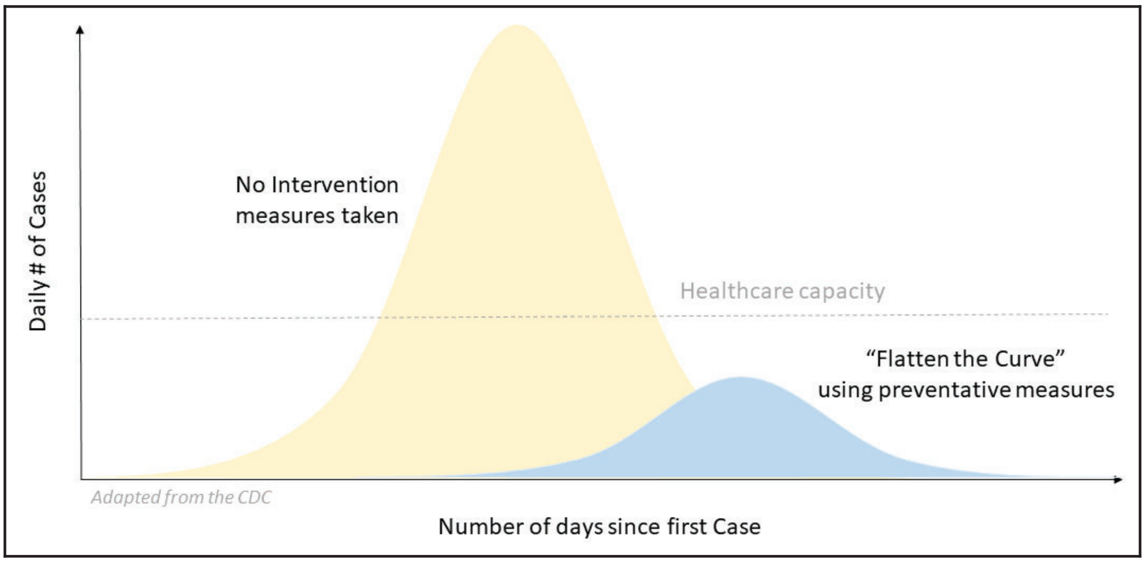
Technical requirements

The GitHub repository of this book can be found at `https://github.com/PacktPublishing/Practical-Data-Analysis-using-Jupyter-Notebook/tree/master/Chapter09`.

You can download and install the required software from `https://www.anaconda.com/products/individual`.

Explaining distribution analysis

I cannot recall a time in history where data, statistics, and science consumed daily lives as it does today. The news cycles are presenting a crisis as it unfolds in real time where changes to human behavior are happening and social norms are being redefined. As I'm writing this book, the concept of **flattening the curve** has gone mainstream and has become a globally understood concept because of the coronavirus (COVID-19) pandemic. You have probably seen something similar to what is shown in the following diagram, which was adapted from the **Centers for Disease Control and Prevention** (**CDC**). These types of visualizations are commonly used to communicate the importance of preventing the spread of a disease. The following visualization has two curves, one in yellow labeled **No Intervention measures taken** and the other in blue named "**Flatten the Curve**" **using preventative measures**. A dotted reference line labeled **Healthcare capacity** is available for relative comparison between the two curves. From a data literacy perspective, we can identify them as distribution curves shown side by side to measure the **Daily # of Cases** on the *y* axis with a common dimension of duration, which is labeled as **Number of days since first Case** on the *x* axis. A distribution curve is common in data analysis and visually represents the numeric data values of a single variable:



For reference, let's review the most well-known distribution curve, which is called a **Gaussian**, **normal**, or a **bell** curve because of the visual similarity to the shape of a physical bell. In a normal distribution, the values would be represented on the line with the highest point representing the mean or average of the sample or entire population of numeric values. As the line stretches away from the top in either direction, it shows how the numbers have variance from the mean. This spread or dispersion of the numbers from the average value is commonly known as the **Standard Deviation** (**SD**). In a perfect distribution, the values would fall within plus or minus one, two, or three standard deviations from the mean, which creates symmetry in the line shape.

Some interesting facts to understand about a normal distribution are the following:

- About 68% of the population of data values fall within plus or minus one standard deviation.
- 96% of the data falls without plus or minus two standard deviation.
- 99.7% of the data falls within plus or minus three standard deviation.
- The calculated mean, median, and mode are all equal.
- There is a 50/50% split between the data left and right of the median.

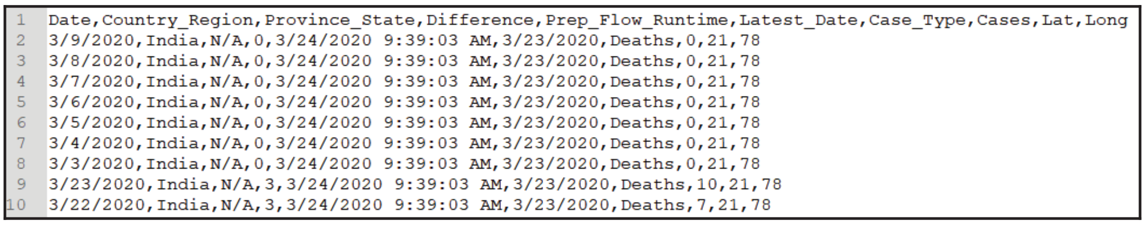
Back to our example, in the first curve labeled **No Intervention measures taken**, the numbers actually double every few days, which creates a steep curve as it approaches the highest point. The second curve, which is identified as the **"Flatten the Curve" using preventative measures**, helps the consumer of this data to visualize the importance of stopping the spread of the virus because the height of the curve has been significantly reduced. I believe this chart became relevant to mass communications because of its simplicity of explaining a distribution curve without going into the statistics behind it. Even without showing the data behind the curves, anyone can visually understand the importance of this critical information.

At this point in time of writing this book, I do not know whether people around the world have successfully achieved the desired result of reducing the mortality rate of COVID-19. Furthermore, it is unknown at this time whether enough preventive measures such as social distancing are helping to **flatten the curve**. Many people all around the world have already suffered from the COVID-19 pandemic travesty. My heartfelt condolences go out to everyone who has suffered.

KYD

To see how the COVID-19 data is distributed, I have provided a snapshot CSV file that is available in the GitHub repository for this book. To support our data analyst **Know Your Data** (**KYD**) mantra, I'll provide some additional information on how the data was collected and its format. The `COVID-19 Cases.csv` file was collected from authoritative open source COVID-19 sources. A GitHub repository maintained by the **Center for Systems Science and Engineering** (**CSSE**) at Johns Hopkins University is available in the *Further reading* section. The CDC has also been distributing COVID-19 data for the greater good.

A sample of the first few records in the CSV file will look similar to the following screenshot, which was retrieved from authoritative sources found in the *Further reading* section:



```
Date,Country_Region,Province_State,Difference,Prep_Flow_Runtime,Latest_Date,Case_Type,Cases,Lat,Long
3/9/2020,India,N/A,0,3/24/2020 9:39:03 AM,3/23/2020,Deaths,0,21,78
3/8/2020,India,N/A,0,3/24/2020 9:39:03 AM,3/23/2020,Deaths,0,21,78
3/7/2020,India,N/A,0,3/24/2020 9:39:03 AM,3/23/2020,Deaths,0,21,78
3/6/2020,India,N/A,0,3/24/2020 9:39:03 AM,3/23/2020,Deaths,0,21,78
3/5/2020,India,N/A,0,3/24/2020 9:39:03 AM,3/23/2020,Deaths,0,21,78
3/4/2020,India,N/A,0,3/24/2020 9:39:03 AM,3/23/2020,Deaths,0,21,78
3/3/2020,India,N/A,0,3/24/2020 9:39:03 AM,3/23/2020,Deaths,0,21,78
3/23/2020,India,N/A,3,3/24/2020 9:39:03 AM,3/23/2020,Deaths,10,21,78
3/22/2020,India,N/A,3,3/24/2020 9:39:03 AM,3/23/2020,Deaths,7,21,78
```

This data source contains a daily snapshot of the COVID-19 cases by country. The key fields used in our analysis are as follows:

- `Date`, which formatted as `M/D/YYYY`, is the date a positive COVID-19 case was identified.
- `Country_Region` is the country of origin where the COVID-19 cases are tracked.
- The `Cases` field is the accumulated count of the number of COVID-19 cases by country and date.
- The `Difference` field is the daily number of COVID-19 cases by country and date.
- `Case_Type` is the type of case that is assigned to each value and is either `Confirmed` or `Deaths`.

From this data source, we can answer multiple questions about the data but will require some filtering to isolate records by `Country`, `Date`, and `Case_Type`.

Shape of the curve

Now that we have more information about the data, we can launch a new Jupyter Notebook for analysis to identify the shape of the curve.

Launch a new Jupyter Notebook and name it `ch_09_exercises`. To import the data from a CSV to a `pandas` DataFrame so we can create a histogram, we use the following steps:

1. Import the following libraries by adding the following codes in your Jupyter Notebook and run the cell. Feel free to follow along by creating your own Notebook; I have also placed a copy in GitHub for reference:

```
In[]: import pandas as pd
     import numpy as np
      import matplotlib.pyplot as plt
      %matplotlib inline
```

TIP These libraries should already be available using Anaconda. Refer to `Chapter 2`, *Overview of Python and Installing Jupyter Notebook*, in case you need help with setting up your environment. `%matplotlib inline` is a magic command required to display the visual results inside your Jupyter Notebook after you run the cell.

2. Next, we create a new DataFrame by importing the CSV file:

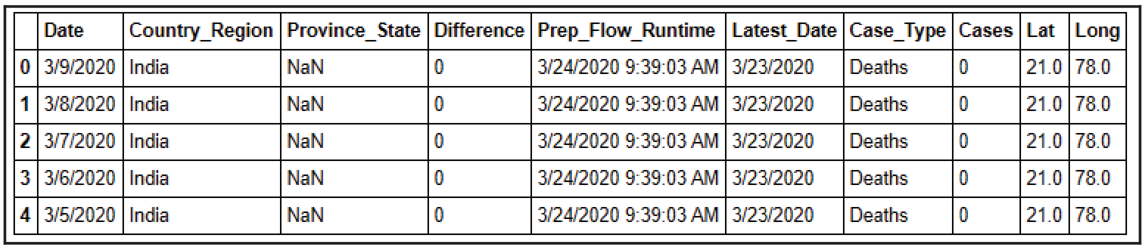
```
In[]: covid_df = pd.read_csv("COVID-19 Cases.csv", header=0)
```

TIP Be sure you copied the `COVID-19 Cases.csv` file to the correct Jupyter folder directory to avoid errors with the connection.

3. To verify the DataFrame has loaded correctly, we can run the `head()` function to display the first few records:

```
In[]: covid_df.head()
```

The output would look like the following screenshot where the source CSV file has been loaded into a DataFrame with a labeled header row with the index column to the left starting with a value of `0`:



	Date	Country_Region	Province_State	Difference	Prep_Flow_Runtime	Latest_Date	Case_Type	Cases	Lat	Long
0	3/9/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
1	3/8/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
2	3/7/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
3	3/6/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
4	3/5/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0

4. Next, we will isolate the data we want to focus our attention on by creating a new DataFrame from the source and applying a few filters against it. We want to isolate the records where all of the following conditions are true. First, the daily `Difference` count is greater than zero. Next, the `Case_Type` should be `Confirmed`. Finally, the `Country_Region` should be only `Italy`:

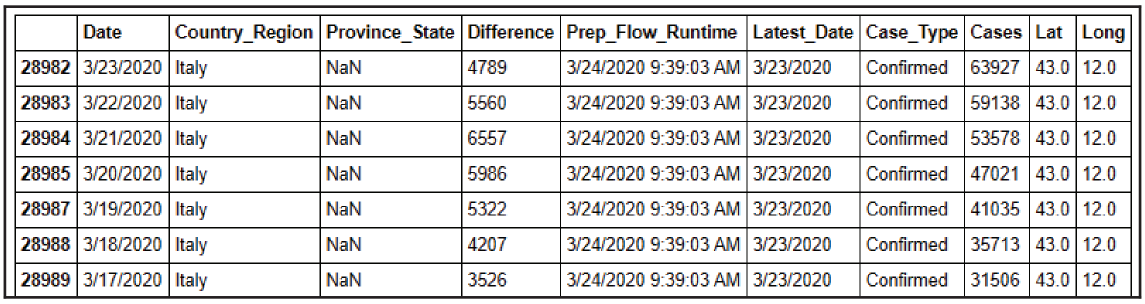
```
In[]: df_results = covid_df[(covid_df.Difference >0) &
(covid_df.Case_Type == 'Confirmed') & (covid_df.Country_Region ==
'Italy')]
```

The new `df_results` DataFrame will not display results in Jupyter Notebook by default.

5. To see the results sorted, we run the following command:

```
In[]: df_results.sort_values(by='Cases', ascending=False)
```

The output should look like the following screenshot, where a new `df_results` DataFrame is displayed with the values sorted by `Cases` in descending order:

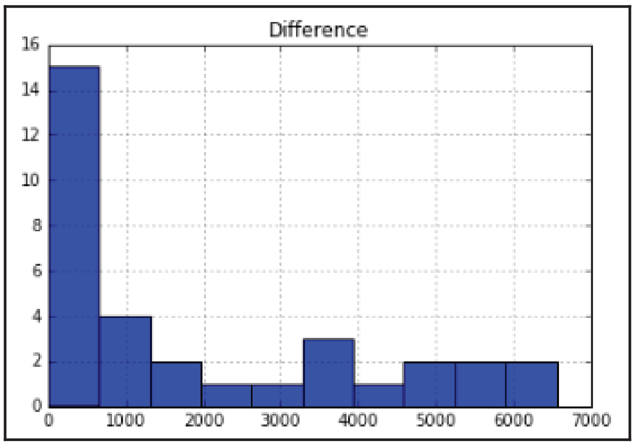


	Date	Country_Region	Province_State	Difference	Prep_Flow_Runtime	Latest_Date	Case_Type	Cases	Lat	Long
28982	3/23/2020	Italy	NaN	4789	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	63927	43.0	12.0
28983	3/22/2020	Italy	NaN	5560	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	59138	43.0	12.0
28984	3/21/2020	Italy	NaN	6557	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	53578	43.0	12.0
28985	3/20/2020	Italy	NaN	5986	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	47021	43.0	12.0
28987	3/19/2020	Italy	NaN	5322	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	41035	43.0	12.0
28988	3/18/2020	Italy	NaN	4207	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	35713	43.0	12.0
28989	3/17/2020	Italy	NaN	3526	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	31506	43.0	12.0

6. Now, we want to visually display the distribution of the values in the `Difference` column. We can pass an array of values into the default `hist()` plot using the following command:

```
In[]: df_results.hist(column='Difference');
```

The output would look like the following screenshot, where the array of values are displayed in a default histogram chart. The default settings bin the data values in equal sizes, which are displayed on the *x* axis. The frequency or count of the number of occurrences of the values falling inside of each bin are measured by the *y* axis:



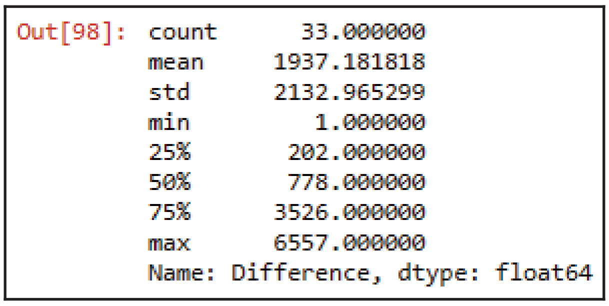
So, what does this histogram plot tell us about the data in the preceding chart at first glance? First, most of the data values are less than **1,000** since the highest bar is the first bar to the left. Second, we know the data is not a normal distribution based on the shape because we would have expected the most frequent results to be in the middle, closer to the mean of the data. How does this analysis apply to the COVID-19 data itself? This shape is good since the number of daily increases could be much larger.

To learn more details about the shape of this data, we can use the `describe()` function against this specific column in the DataFrame.

7. Use the `describe()` function against this DataFrame to see summary statistics. We can look at one column by explicitly passing it in the square brackets along with the column/field name in double quotes:

```
In[]: df_results["Difference"].describe()
```

The output would look like the following screenshot where summary statistics about the data in this specific field are displayed:



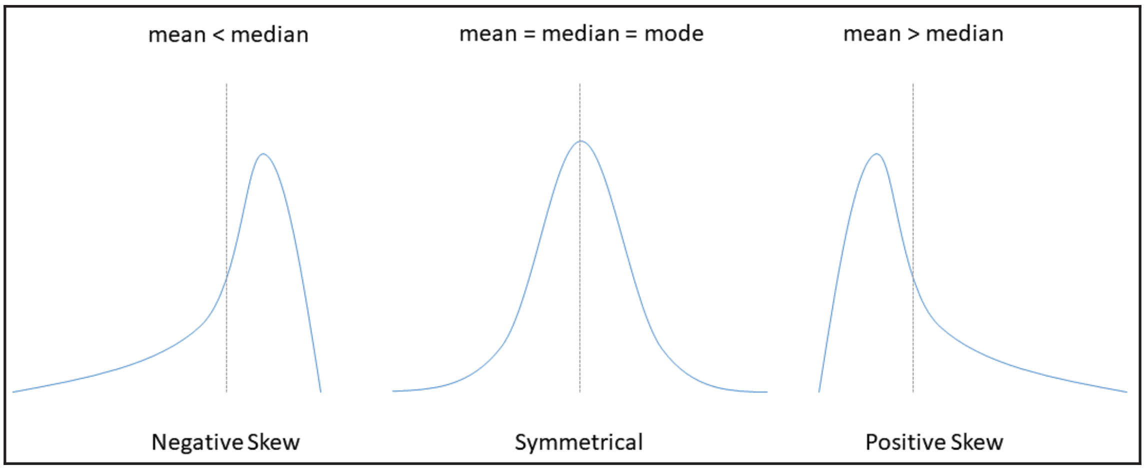
```
Out[98]: count      33.000000
         mean     1937.181818
         std      2132.965299
         min         1.000000
         25%       202.000000
         50%       778.000000
         75%      3526.000000
         max      6557.000000
         Name: Difference, dtype: float64
```

In the preceding screenshot, we have identified some key statistics to help to better understand the shape of this data. There are `33` values that are identified as `count` in the summary table with a `mean` of `1937.181818` and a `std` (standard deviation) of `2132.965299`. The range of those thirty-three values is from `1` to `6557`, which is identified by `min` and `max`. With that high of a standard deviation value, we know the numbers are pretty spread out.

TIP

The values will be displayed with a datatype of `float64` with a precision of six decimal places regardless of the source number value.

The **25%**, **50%**, and **75%** labels return the respective percentiles for the series of values in this field. These values are also known as the **Interquartile Range** (**IRQ**) with the 50% or second quartile equal to the **median**. Having the data in quartiles creates equal bins or buckets for the data values to help us understand how the numeric values are distributed. If a majority of the values fall into one specific bucket, you know the data is not evenly distributed. With our example, we have a large gap between our mean and median with our data (1937 versus 778) so we can classify this data as skewed. Having a skew in our data helps to understand that the visual shape of the distribution curve or histogram is not symmetrical. To help you to remember the distribution types and skewness, I have summarized them in the following graph. When the mean is greater than the median, it would have a positive skew and when the opposite is true, a negative skew exists. As described at the top of each visual trend in the following diagram, the type of skew (negative or positive) is directly correlated with the mean and median values. When all of the mean, median, and mode values are equal, you have a symmetrical distribution:

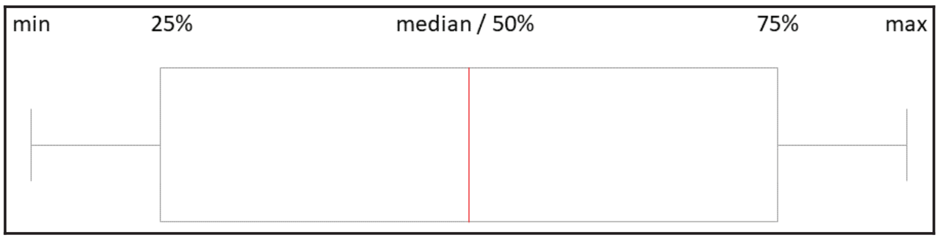


From a data analysis perspective, having these key statistics defined helps us to understand the spread of the data values. Calculating the mean, median, and mode against your data values are collectively known as the measures of **central tendency**, which we introduced in `Chapter 8`, *Understanding Joins, Relationships, and Aggregates*. A very important and practical use of central tendency is in data science models. In predictive regression models that use historical data, the ability to calculate a prediction is based on finding a *best fit* to the distribution curve. If the data has a dramatic positive or negative skew with *long tails*, which is when values trail off by multiple standard deviations from the central tendency, the algorithm becomes less accurate.

So, now we understand the importance of calculating central tendency and how symmetrical data is visually represented as a normal distribution curve. A normal distribution, also known as the Gaussian distribution and the bell curve, occurs when the mean (average) is equal to the median (middle) and is equal to the mode (most frequent). I find adding a normal distribution line useful as a reference to compare against the actual data results in charts. This helps the consumer of the analysis to visually compare the ideal shape of the data versus the actual results. So, what causes data to skew or not fit into a normal distribution? As a data analyst, your job is to find out why and the first step is to isolate outliers that may exist in the data. We will discuss this in the next section by understanding outliers and trends.

Understanding outliers and trends

Finding outliers begins by looking at the distribution curve but requires additional techniques that we will walk through together. Additionally, don't underestimate the need for soft skills where you must reach out to others to better understand why an outlier exists in your data. An outlier is commonly known as one or more data values that are significantly different than the rest of the data. Spotting outliers in data is easy depending on the data visualization used, but in many cases, especially when data volumes are very large, they can be obscured when data is aggregated. If you recall from `Chapter 7`, *Exploring Cleaning, Refining, and Blending Datasets*, we worked with hits created by a user for a website. A good example of obscuring outliers is when those user hits are aggregated by date. If a specific user has 1,000 hits per day when the average is 2, it would be difficult to identify that outlier user after the data was aggregated by week. So, what does an outlier look like visually in a series of data values? A good approach would be to use a box plot because it visually represents the data found in the `describe()` function:



As you can see in the preceding diagram, the box isolates the quartiles of **25%**, **50%**, and **75%**, and the min/max range of values is displayed at the most extreme vertical lines. The space between the box and the min/max lines is known as the whiskers of the box plot. If you see a plus symbol (+) displayed, they are known as fliers, which are outliers in this chart type.

A box plot can be displayed horizontally or vertically and can include multiple dimensions so you can compare the distribution between them.

Let's continue to analyze our existing dataset and see how it would be visualized using a box plot. Similar to the prior example, we will load all of the data from the source into a single DataFrame and then create a subset DataFrame using filters. We will continue using the `ch_09_exercises` Jupyter Notebook:

1. Import the following libraries by adding the following command in your Jupyter Notebook and run the cell. Feel free to follow along by creating your own Notebook; I have placed a copy on GitHub for reference:

```
In[]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      %matplotlib inline
```

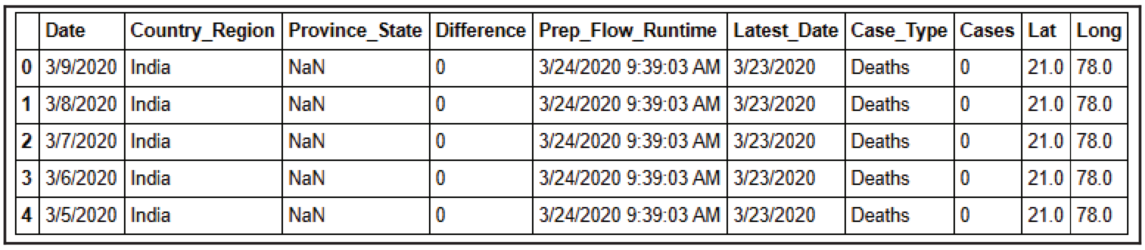
2. Create a new DataFrame by importing the CSV file:

```
In[]: covid_df = pd.read_csv("COVID-19 Cases.csv", header=0)
```

3. To verify the DataFrame has loaded correctly, we can run the `head()` function to display the first few records:

```
In[]: covid_df.head()
```

The output would look like the following screenshot where the source CSV file has been loaded into a DataFrame with a labeled header row with the index column to the left starting with a value of `0`:



	Date	Country_Region	Province_State	Difference	Prep_Flow_Runtime	Latest_Date	Case_Type	Cases	Lat	Long
0	3/9/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
1	3/8/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
2	3/7/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
3	3/6/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0
4	3/5/2020	India	NaN	0	3/24/2020 9:39:03 AM	3/23/2020	Deaths	0	21.0	78.0

Similar to the prior exercise, we will isolate the data we want to focus attention on by creating a new DataFrame from the source and applying a few filters against it. We want to isolate the records where all of the following conditions are true. First, the daily `Difference` count is greater than zero. Next, `Case_Type` should be `Confirmed`. Finally, we use the pipe symbol, `|`, to create an `or` condition to allow for multiple `Country_Region`:

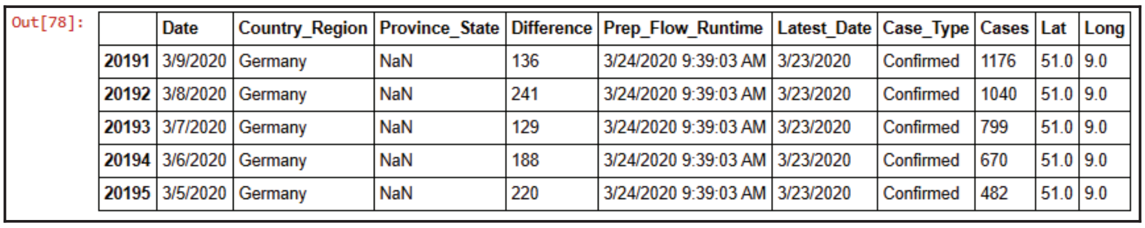
```
In[]: df_results = covid_df[(covid_df.Difference >0) &
(covid_df.Case_Type == 'Confirmed') & ((covid_df.Country_Region ==
'Italy') | (covid_df.Country_Region == 'Spain') |
(covid_df.Country_Region == 'Germany'))]
```

The new `df_results` DataFrame will not display results in Jupyter Notebook by default.

4. To see the results, we run the following command:

```
In[]: df_results.head()
```

The output should look like the following screenshot where a new `df_results` DataFrame is displayed:



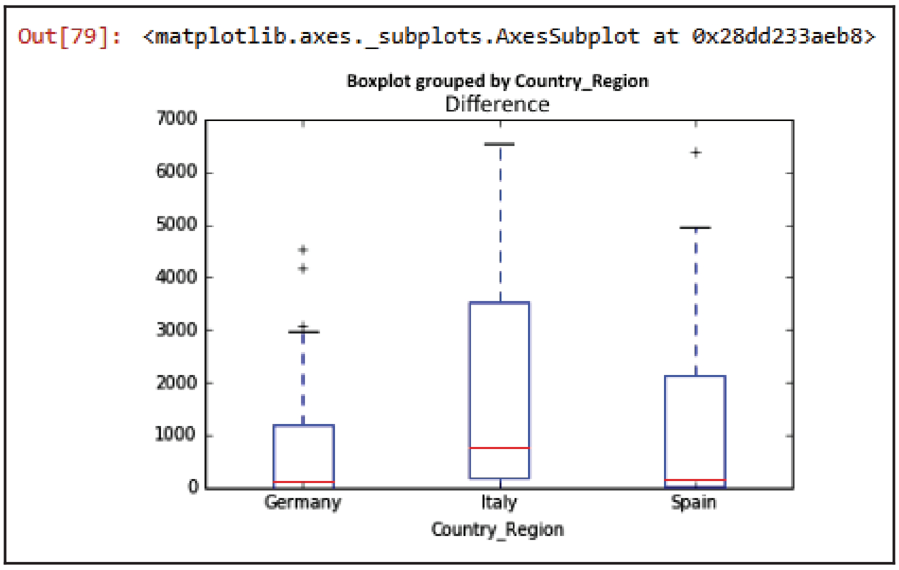
Out[78]:

	Date	Country_Region	Province_State	Difference	Prep_Flow_Runtime	Latest_Date	Case_Type	Cases	Lat	Long
20191	3/9/2020	Germany	NaN	136	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	1176	51.0	9.0
20192	3/8/2020	Germany	NaN	241	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	1040	51.0	9.0
20193	3/7/2020	Germany	NaN	129	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	799	51.0	9.0
20194	3/6/2020	Germany	NaN	188	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	670	51.0	9.0
20195	3/5/2020	Germany	NaN	220	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	482	51.0	9.0

5. To display a box plot by `Country`, we use the following command. `boxplot()` has a few parameters such as `by=`, which allows us to group the data by `Country_Region`. We also include `column=` to isolate the values in the `Difference` field. Finally, we pass in `grid=False` to turn off the gridlines in the chart:

```
In[]: df_results.boxplot(by='Country_Region',
column=['Difference'], grid=False)
```

The output would look like the following screenshot where a box plot will be displayed:



```
Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x28dd233aeb8>
```

So, having the data limited to only three countries allows us to narrow our analysis, and having the data presented side by side in the box plot chart, as shown in the preceding screenshot, allows us to visually compare the results. First, we notice a few box sizes, which, if you recall, are the quartiles of the data and are different sizes depending on `Country`. Germany has a smaller box that is closer to a square than a rectangle, which typically tells us the data spread is much tighter. Another observation we can identify in this chart is we have multiple plus symbols (+) displayed; these highlight outliers that exist in the countries of Germany and Spain.

Analyzing data by country and other attributes related to geography is a common requirement today for a data analyst. Next, we will explore best practices related to visually representing data in maps and spatial data, which is known as geoanalytics.

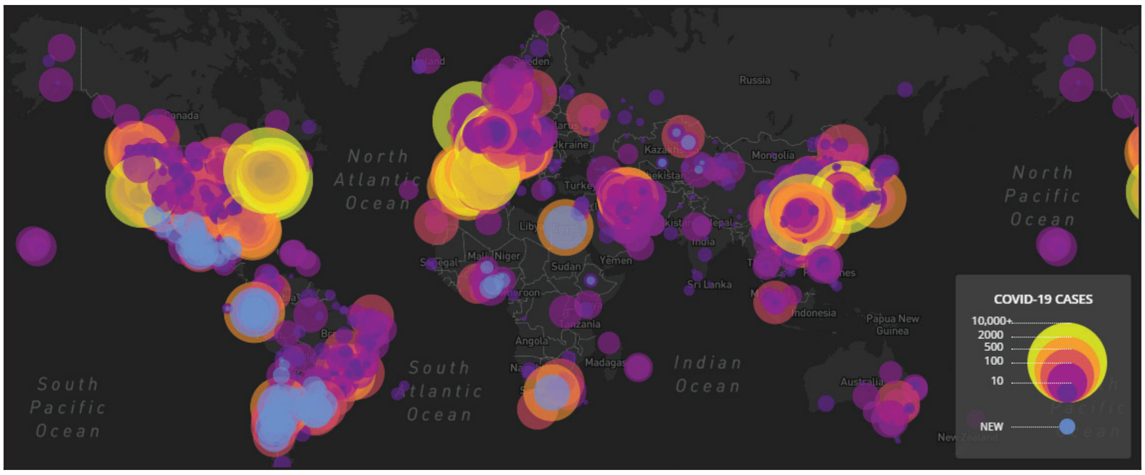
Geoanalytical techniques and tips

For a data analyst, the concept of geoanalytics is a relatively new technique applied to spatial data to understand where data is geographically located. However, cartography, which is the study of maps, has been around for centuries and traditionally requires training, expertise, and niche software to provided insights from data by location. Today, there are multiple add-on modules and software available to create charts and visualizations that use maps to visualize data in exciting ways that provide a different perspective.

First, you need to understand the grain of the data you have available. Having precision of the exact latitude and longitude available in your source data is a luxury unless the source system was built to capture that information. For example, mobile app source data will commonly have this level of detail available because a smartphone can track your location. However, if we go back to our COVID-19 source data, the individual cases' lowest level of detail available is by `Province_State` so you lose the ability to display data below that level.

Next, if your source data does not include latitude and longitude values, you will have to add it, which sounds simple at first but usually has some challenges that you will have to overcome. When you profile the data, ensure it has conformity and consistency in specific fields such as `Country`, `City`, `Parcel`, or `Zip Code`. If the `Country` values exist, do they have **International Organization for Standardization** (**ISO**) codes so you can join the data to commonly available sources? If so, I have included a source a link to the World Bank data source in the *Further reading* section.

Finally, I find including a world or regional map as a good complement to, but not a replacement for, good data analysis solutions. In many cases, having a single map chart even with color gradients does not provide enough answers to common questions. For example, let's look at the following screenshot, which shows a global map of the COVID-19 outbreak found on the HealthMap site:



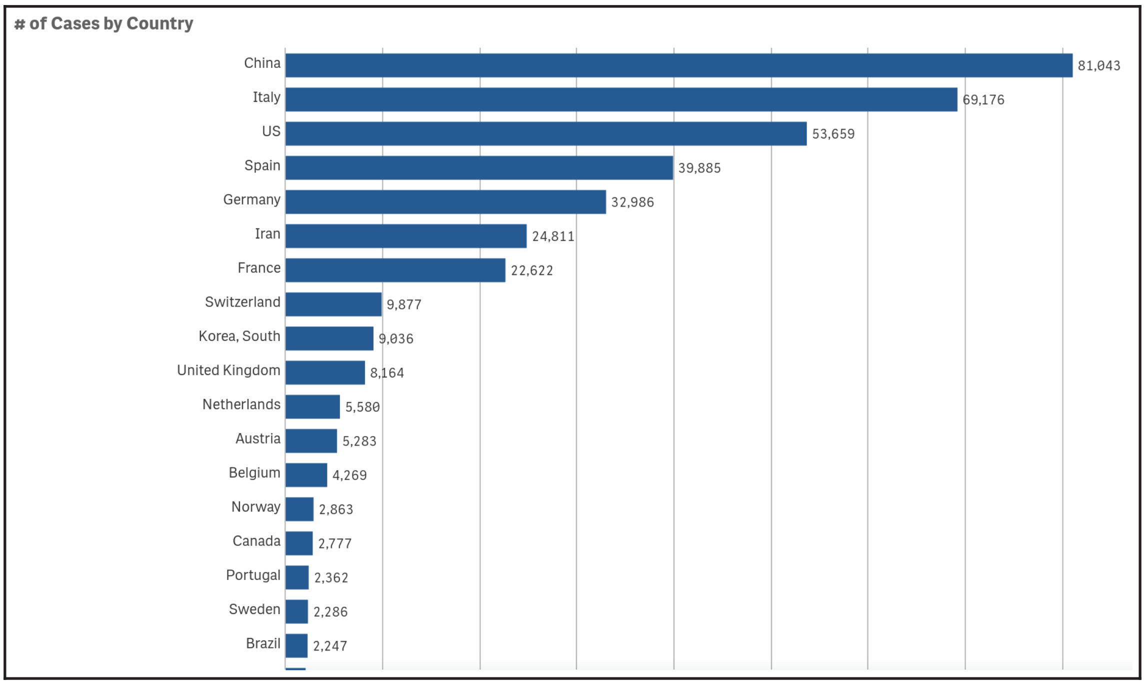
As displayed in the preceding screenshot, the legend at the bottom right of the chart provides insights on the outbreak by location using the color and size of the bubbles. If you look at this visualization as a static map, it leaves the audience asking more questions about the overlap of the bubbles with colors. For example, how do you distinguish between the details of closely clustered cities such as New York and Philadelphia? Also, the consumer of this geoanalytic chart might want to know whether the number of cases is accelerating per location, which is unclear.

However, these criticisms change once you visit the site, which allows the user to interact with the map.

When you use the HealthMap site created for the COVID-19 outbreak, you are offered a solution that provides the following features:

- Zoom capabilities to drill down to the lowest level of detail and zoom out to compare results across different locations
- Mouse hover over the circles that provide aggregated counts by location along with the context of which location is selected or has the focus
- A timeline feature that allows the user to see before and after results

Once you appreciate having any or all of these features available in geoanalytics data, you expect it for any solution you are building as a data producer. If you are missing the data or tools to create it, I would recommend creating a complementary chart to support it. For example, in the following screenshot, we have a horizontal bar chart sorted in descending order by country:



The legend at the top of the chart indicates the measure used, which is titled **# of Cases by Country**, to help the consumer of the data easily answer questions such as which country has the highest number of cases as of a specific date. When you bring together multiple charts that complement each other, they provide context to the consumer of the geographic data.

TIP

Be sure the common join key between the data sources behind these charts is consistent. If you select a country on the map, the bar chart should filter to match along with any date selections.

Using multiple charts in tandem helps to tell a story with the information to build a connection with the audience. The consumers of data today are sophisticated so telling the same story over and over will not be effective. I recommend using time-honored techniques such as having a lesson or moral for the story to work well and you should feel empowered to adjust to include your own personal style.

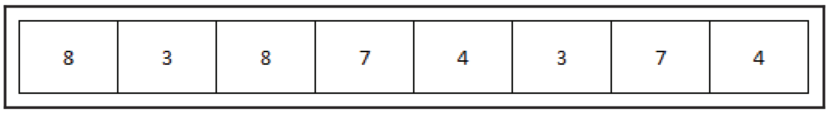
I find creative inspiration from art and learning more about the masters of their craft. For example, the artist Pablo Picasso is well known for works created during his **Blue Period**, which defined a time in his life when all variations of the color blue were the primary color used commonly across different subjects he painted. This period lasted a few years and reflected his personal struggles living with depression and financial distress. In comparison, the COVID-19 pandemic is creating personal struggles for people all around the world. The high levels of mortality related to the COVID-19 data are causing global financial distress and numerous stories of personal loss. Picasso produced a staggering volume of work during his lifetime, with well over 100,000 pieces over a 70 plus year timeframe. Even during times of emotional distress, Picasso continued to find the strength to create new works of art and master his craft. I can relate to his struggles during this pandemic and am inspired to spend time on my data craft to help me through these trying times.

The power of storytelling with data becomes a critical skill to build trust with your audience so they can understand the information. Using data visualizations breaks down the technical barriers that can exist when working with data. As a person fluent in data literacy, you now have the additional skills required to create your own story using data.

Now that you understand the geographic techniques that are effective in data storytelling, let's focus our attention on identifying patterns within data.

Finding patterns in data

Now that we have a better understanding of distribution curves and spotting outliers that can exist in your data, let's break down how to find patterns in your data. In my experience, as you work with more and more data, you will start to develop a **sixth sense** that will help you identify patterns faster, for example, the following diagram may appear like a random list of numbers where no distinguishing pattern is obvious, until you make some minor changes to make it easier to identify:



8	3	8	7	4	3	7	4

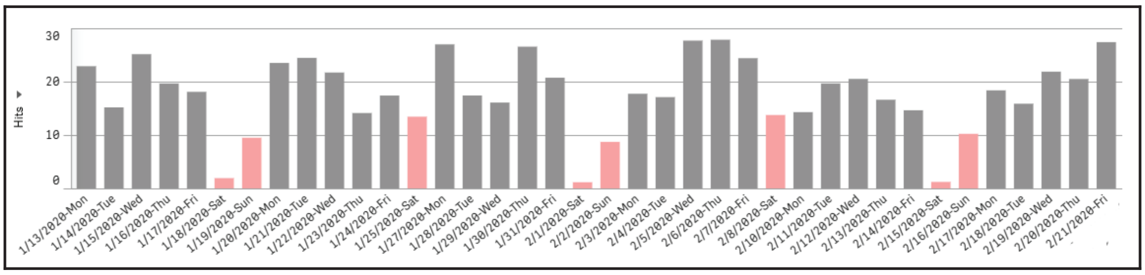
Having the data sorted allows you to see groupings and clusters that exist within the data values. In this case, we have pairings of numbers that were not as evident until you sorted them together. With a quick sort, we can now see all of the numbers are duplicated, as in the following diagram:

3	3	4	4	7	7	8	8

To hammer this point home, look at the following diagram where those same numbers from the preceding two diagrams are now colored by pairs, which creates a pattern to make it easier to visually identify:

3	3	4	4	7	7	8	8

Data visualizations and charts will help you to identify these patterns as well, but some charts are better suited than others depending on the data. For example, to see patterns in the data over time, a line or bar chart will display the trends better and help to create recognizable patterns in your data. A good example of that can be seen in the following screenshot, which is a bar chart with a measure of **Hits** on the y axis and a dimension of date with the day of the week in the `dtype` format of `M/DD/YYYY-DDD` displayed on the x axis:



In the preceding screenshot, we have a trend chart that displays a pattern of web hit usage over each date. What becomes more evident when you look at this data sorted is the peaks and valleys that naturally appear every few days. Even without understanding all of the details behind it, having this pattern suggests our weekday data usage is higher than weekends, which I highlighted in a different color to make it more obvious.

During this time of analysis and identifying patterns, you will find yourself coming to conclusions about the data. Doing this is natural and if based on business or domain expertise, is viable. If you find patterns in your data that apply to multiple dimensions or variables, you can identify a correlation between them. Correlations are common in data, especially when overlaying patterns over the same timeframe. A more formal definition is when one variable or series of values increase or decrease, a second variable will follow in parallel. A common example of a correlation between two values would be ice-cream store sales and the weather. If the weather has snow or heavy rain or is cold, ice-cream sales are typically lower. If the weather is warm and sunny, sales would be higher. Based on this information, you could say there is a **positive** correlation between the variables of sales and weather over the same period of time.

If the opposite relationship exists, where the inverse pattern between the two variables occurs, this would be considered a **negative** correlation.

To determine whether two variables are statically correlated, there is a concept called the **correlation coefficient.** This is a measurement that falls between `1` and `-1` and is denoted by ***r***. If the value is `0`, there is no correlation between the two variables. The closer the values are to `1`, they have a positive correlation, which means when one variable's value changes, the other will trend in the same direction. The opposite is true when the values are closer to `-1`, where a negative correlation creates an inverse relationship between the two variables. So, can we visually see a correlation and pattern with data? A good approach would be to use a scatter plot.

Let's continue to analyze our existing dataset and see how it would be visualized using the scatter plot. Similar to the prior example, we will load all of the data from the source into a single DataFrame and then create a subset DataFrame using filters. In this example, we are going to create two subsets to allow for comparisons. We will continue using the `ch_09_exercises` Jupyter Notebook:

1. Import the following libraries by adding the following command in your Jupyter Notebook and run the cell. Feel free to follow along by creating your own Notebook; I have placed a copy on GitHub for reference:

   ```
   In[]: import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   %matplotlib inline
   ```

2. Create a new DataFrame by importing the CSV file:

   ```
   In[]: covid_df = pd.read_csv("COVID-19 Cases.csv", header=0)
   ```

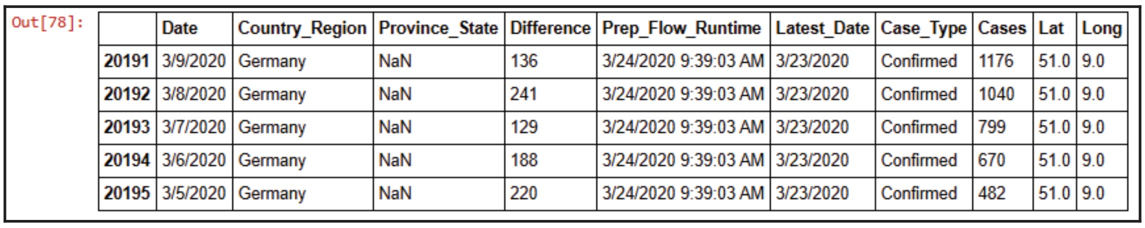
3. We will now create two new DataFrames, which will be subsets from the original source. The advantage of naming them generically as `df_results_1` and `df_results_2` is that it allows you to adjust the filters such as `Country_Region` used in this one line without changing any other code in the additional steps:

   ```
   In[]: df_results_1 = covid_df[(covid_df.Case_Type == 'Confirmed') &
   (covid_df.Country_Region == 'Germany')]
   ```

4. Run the `head()` function to validate the results:

   ```
   In[]: df_results_1.head()
   ```

   The output will look like the following table where a new `df_results_1` DataFrame is displayed:



Out[78]:

	Date	Country_Region	Province_State	Difference	Prep_Flow_Runtime	Latest_Date	Case_Type	Cases	Lat	Long
20191	3/9/2020	Germany	NaN	136	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	1176	51.0	9.0
20192	3/8/2020	Germany	NaN	241	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	1040	51.0	9.0
20193	3/7/2020	Germany	NaN	129	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	799	51.0	9.0
20194	3/6/2020	Germany	NaN	188	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	670	51.0	9.0
20195	3/5/2020	Germany	NaN	220	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	482	51.0	9.0

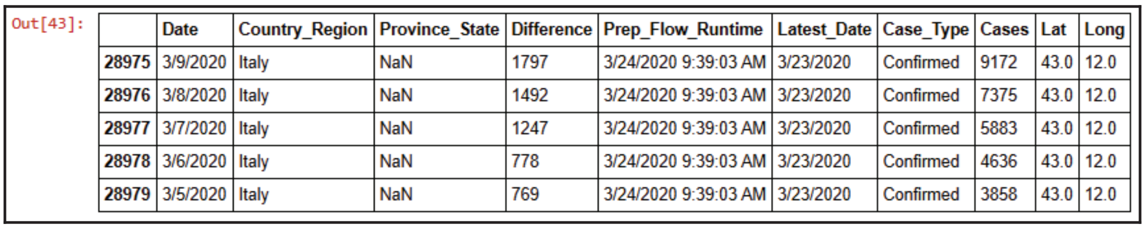
5. We will load the second DataFrame that we will use to compare with the first using the following commands:

   ```
   In[]: df_results_2 = covid_df[(covid_df.Case_Type == 'Confirmed') &
   (covid_df.Country_Region == 'Italy')]
   ```

6. Run the `head()` function to validate the results:

   ```
   In[]: df_results_2.head()
   ```

   The output would look like the following table where a new `df_results_2` DataFrame is displayed:



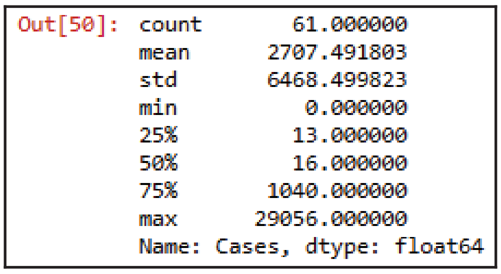
Out[43]:

	Date	Country_Region	Province_State	Difference	Prep_Flow_Runtime	Latest_Date	Case_Type	Cases	Lat	Long
28975	3/9/2020	Italy	NaN	1797	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	9172	43.0	12.0
28976	3/8/2020	Italy	NaN	1492	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	7375	43.0	12.0
28977	3/7/2020	Italy	NaN	1247	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	5883	43.0	12.0
28978	3/6/2020	Italy	NaN	778	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	4636	43.0	12.0
28979	3/5/2020	Italy	NaN	769	3/24/2020 9:39:03 AM	3/23/2020	Confirmed	3858	43.0	12.0

7. Let's profile the data in each DataFrame to better understand it. We use the `describe()` function to better identify key statistics and how the data is distributed. First, we look at the contents of the first DataFrame:

```
In[]: df_results_1["Cases"].describe()
```

The output will look like the following screenshot where results are displayed:

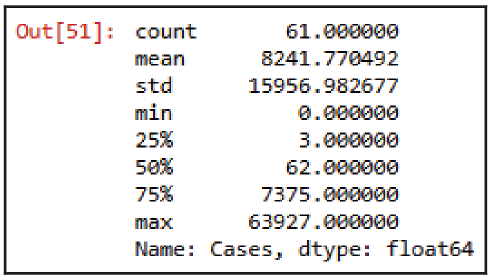


```
Out[50]: count        61.000000
         mean       2707.491803
         std        6468.499823
         min           0.000000
         25%          13.000000
         50%          16.000000
         75%        1040.000000
         max       29056.000000
         Name: Cases, dtype: float64
```

8. Then, we look at the contents of the second DataFrame:

```
In[]: df_results_2["Cases"].describe()
```

The output will look like the following screenshot where results are displayed:



```
Out[51]: count        61.000000
         mean       8241.770492
         std       15956.982677
         min           0.000000
         25%           3.000000
         50%          62.000000
         75%        7375.000000
         max       63927.000000
         Name: Cases, dtype: float64
```

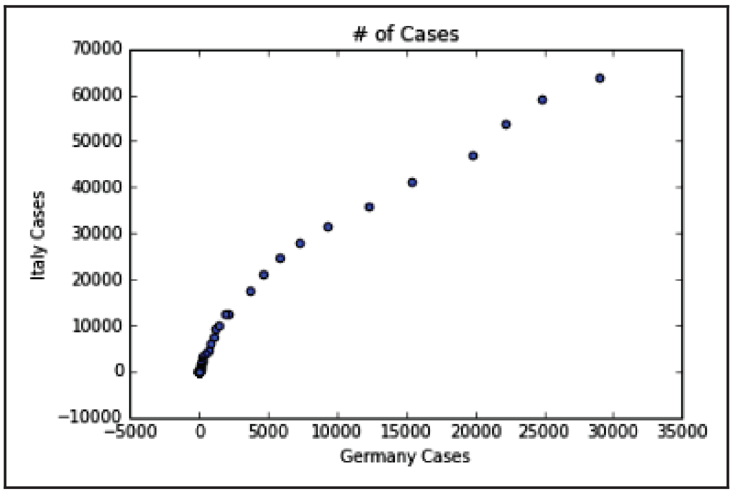
Based on the results of the `describe()` function run against each DataFrame, we have a basis for comparison. First, we have the count, which is the same value of `61` values. This is important when creating a scatter plot since the size of the data is required to be the same. Another common value between the two data series is the minimum, which is at `0`. However, the maximum values are different, which are slightly larger than double (29,056 versus 63,927). Next, we have the mean, which is vastly different. The first results have a rounded mean value of `2,707.49` and the second is `8,241.77`. Finally, the standard deviation (**std**) is different as well so we know the shape and size of the distribution curves will be different.

The question we want to answer is: are these values correlated? To confirm this visually, we continue by creating a scatter plot with a few simple commands. The scatter plot will have *x* and *y* axes and plot the values in a grid with dots representing where the values align to the axis.

9. We use the `plt.scatter()` function to create the visualization. It requires two parameters, which are the *x* and *y* axes values separated by a comma. We are passing the common series of values found in the `Cases` column from each DataFrame. We also include labels and a title to help the audience to understand the chart:

```
In[]: plt.scatter(df_results_1["Cases"], df_results_2["Cases"]);
plt.title("# of Cases")
plt.xlabel("Germany Cases")
plt.ylabel("Italy Cases");
```

The output would look like the following graph where results are displayed:



The result of our scatter plot does have a correlation where the values closer to 0 on either axis cluster together, which are shown with overlapping blue dots on the chart. You can also observe a natural straight-line alignment between the values as the **# of Cases** increase.

With every correlation comes the inevitable expectation that one variable is dependent on the other and is the cause. Causation is when one variable directly impacts the direction of another. Cause and effect or root cause analysis are common analysis techniques. Identifying causation between only two variables is rare and requires deeper analysis to understand all of the factors that directly and indirectly impact the change. The first point to understand is that *correlation doesn't always equal causation*. A good example is our ice-cream store sales and the weather. We know that eating more ice-cream would never improve the weather but if you look purely at the data, you might accidentally come to that kind of conclusion if the data is highly correlated. Another point when analyzing data to determine whether correlation and causation are related is based on the sample size of the data. I would recommend being a **data skeptic** and question a causative conclusion if the population of data is incomplete or covers a small window of time. Finally, we have walked through the value of joining data for data analysis but that invites the opportunity to come to conclusions about data that didn't exist independently. Be sure to add assumptions and be transparent with the methods of how you joined the data so the correlations can be peer-reviewed.

Summary

Congratulations, we have now learned some essential skills for making various plots that visualize the distribution of data. We discussed key statistics related to the central tendency of data by calculating the standard deviation, mean, median, and mode of a series of data values. We looked at normal distributions and how data values can be skewed positively or negatively. When data has symmetry, it becomes easier to work with some algorithms found in predictive analytics. We reviewed patterns and outliers that are common when working with datasets, along with how to use a box plot chart to visualize outliers.

We discussed best practices and tips for working with geospatial data, along with how it can be used to help to tell a story with data. Finally, we discussed the difference between correlation versus causation along with the importance of the correlation coefficient, so you can understand the relationships between two variables/series of data values.

In our next chapter, we will be switching to working with unstructured data sources and best practices when working with free text data.

Further reading

For more information on the relative topics of this chapter, you can visit the following links:

- Authoritative sources for COVID-19 data: `https://github.com/CSSEGISandData/COVID-19`
- Centers for Disease Control and Prevention COVID-19 data: `https://www.cdc.gov/coronavirus/2019-ncov/`
- Cheatsheets to help to create data visuals using Python: `https://python-graph-gallery.com/cheat-sheets/`
- The World Bank ISO country data: `https://datahelpdesk.worldbank.org/knowledgebase/articles/898590-country-api-queries`
- Open source mapping software: `https://www.openstreetmap.org/`
- HealthMap geoanalytics example of COVID-19: `https://www.healthmap.org/covid-19/`