CHAPTER 10

An Example—Meteorological Data

One type of data that's easier to find on the Internet is meteorological data. Many sites provide historical data on many meteorological parameters, such as pressure, temperature, humidity, rain, and so on. You only need to specify the location and the date to get a file with datasets of measurements collected by weather stations. These data are a source of a wide range of information. As you read in the first chapter of this book, the purpose of data analysis is to transform the raw data into information and then convert it into knowledge.

In this chapter, you will see a simple example of how to use meteorological data. This example is useful for getting a general idea of how to apply many of the techniques seen in the previous chapters.

A Hypothesis to Be Tested: The Influence of the Proximity of the Sea

At the time of writing of this chapter, I find myself at the beginning of summer and temperatures rising. On the weekend, many inland people travel to mountain villages or cities close to the sea, in order to enjoy a little refreshment and get away from the sultry weather of the inland cities. This has always made me wonder what effect the proximity of the sea has on the climate.

This simple question can be a good starting point for data analysis. I don't want to pass this chapter off as something scientific; it's just a way for someone passionate about data analysis to put knowledge into practice in order to answer this question—what influence, if any, does the proximity of the sea have on local climate?

The System in the Study: The Adriatic Sea and the Po Valley

Now that the problem has been defined, it is necessary to look for a system that is well suited to the study of the data and to provide a setting suitable for this question.

First you need a sea. Well, I'm in Italy and I have many seas to choose from, since Italy is a peninsula surrounded by seas. Why limit myself to Italy? Well, the problem involves a behavior typical of the Italians, that is, they take refuge in places close to the sea during the summer to avoid the heat of the hinterland. Not knowing if this behavior is the same for people of other nations, I will only consider Italy as a system of study.

But what areas of Italy might we consider studying? Can we assess the effects of the sea at various distances? This creates a lot of problems. In fact, Italy is rich in mountainous areas and doesn't have a lot territory that uniformly extends for many kilometers inland. So, to assess the effects of the sea, I exclude the mountains, as they may introduce many other factors that also affect climate, such as altitude, for example.

© Fabio Nelli 2023
F. Nelli, *Python Data Analytics*, https://doi.org/10.1007/978-1-4842-9532-8_10

A part of Italy that is well suited to this assessment is the Po Valley. This plain starts from the Adriatic Sea and spreads inland for hundreds of kilometers (see Figure 10-1). It is surrounded by mountains, but the width of the valley mitigates any mountain effects. It also has many towns and so it is easy to choose a set of cities increasingly distant from the sea, to cover a distance of almost 400 km in this evaluation.

Figure 10-1. *An image of the Po Valley and the Adriatic Sea (Google Maps)*

The first step is to choose a set of ten cities that will serve as reference standards. These cities are selected in order to cover the entire range of the plain (see Figure 10-2).

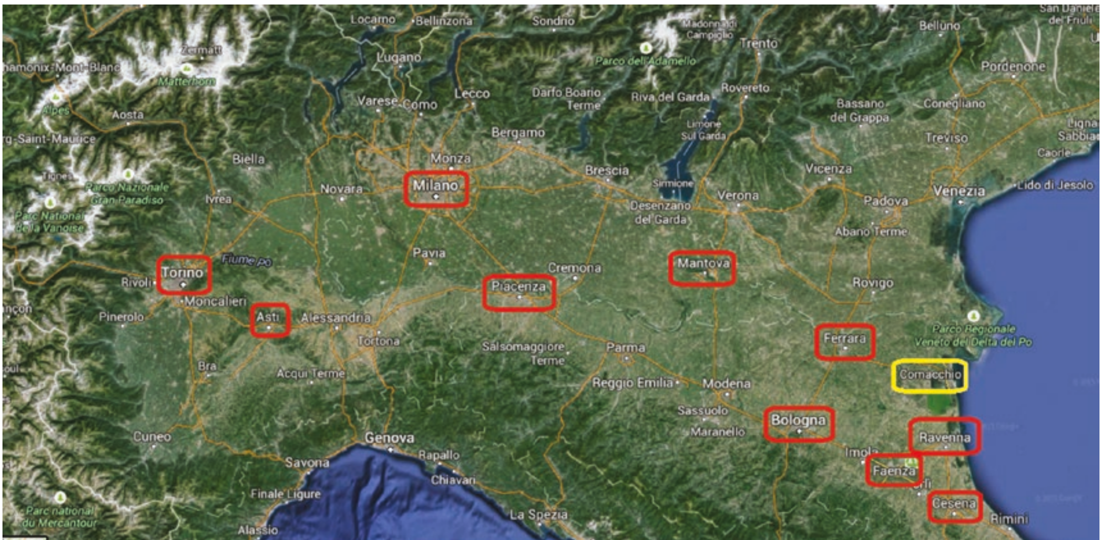

Figure 10-2. The ten cities chosen as samples (there is another one used as a reference for distances from the sea)

In Figure 10-2, you can see the ten cities that were chosen to analyze weather data: five cities within the first 100 km and the other five distributed in the remaining 300 km.

Here are the chosen cities:

- Ferrara
- Torino
- Mantova
- Milano
- Ravenna
- Asti
- Bologna
- Piacenza
- Cesena
- Faenza

Now you have to determine the distances of these cities from the sea. You can follow many procedures to obtain these values. In this case, you can use the service provided by the site TheDistanceNow (`www.thedistancenow.com/`), which is available in many languages (see Figure 10-3).

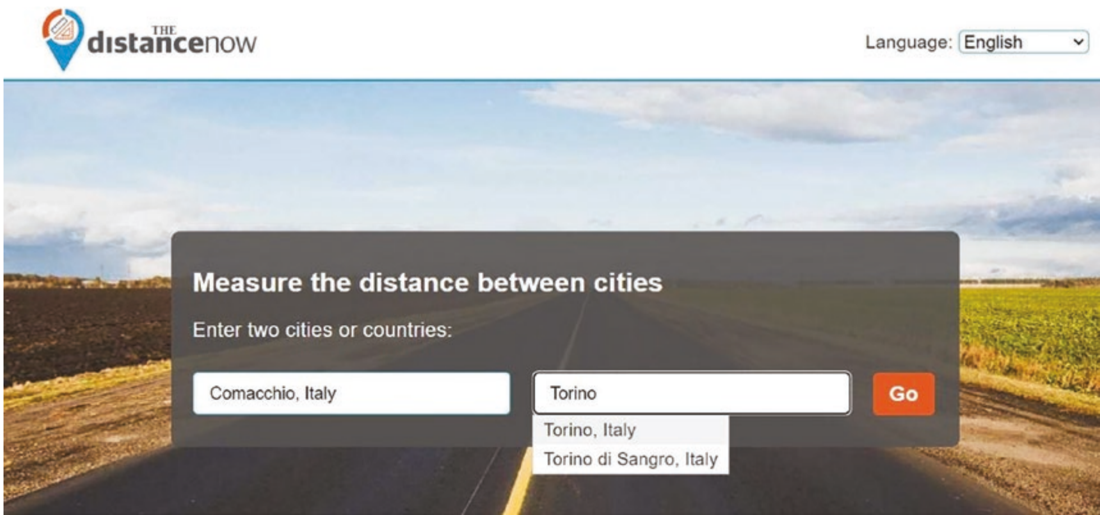

Figure 10-3. *TheDistanceNow website allows you to calculate distances between two cities*

Thanks to this service, it is possible to calculate the approximate distances of the cities from the sea. You can do this by selecting a city on the coast as the destination. For many of them, you can choose the city of Comacchio as a reference to calculate the distance from the sea (see Figure 10-2). Once you have determined the distances from the ten cities, you will get the values shown in Table 10-1.

Table 10-1. *The Distances from the Sea of the Ten Cities*

City	Distance (km)	Note
Ravenna	8	Measured with Google Earth
Cesena	14	Measured with Google Earth
Faenza	37	Distance Faenza-Ravenna +8km
Ferrara	47	Distance Ferrara-Comacchio
Bologna	71	Distance Bologna-Comacchio
Mantova	121	Distance Mantova-Comacchio
Piacenza	200	Distance Piacenza-Comacchio
Milano	250	Distance Milano-Comacchio
Asti	315	Distance Asti-Comacchio
Torino	357	Distance Torino-Comacchio

Finding the Data Source

Once you have defined the system under study, you need to establish a data source from which to obtain the needed data. By browsing the Internet, you can discover many sites that provide meteorological data measured from various locations around the world. One such site is OpenWeather, available at `openweathermap.org` (see Figure 10-4).

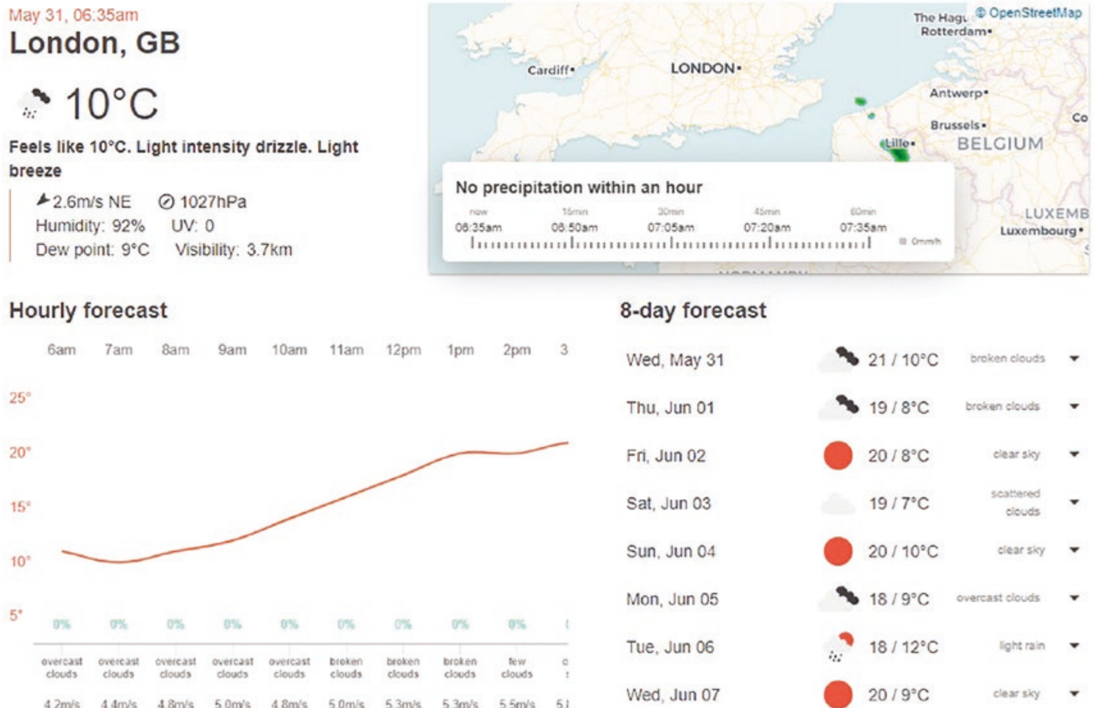

Figure 10-4. *The OpenWeather site*

After you've signed up for an account and received an app ID code, this site enables you to capture data by specifying the city through a request via an URL.

```
https://api.openweathermap.org/data/2.5/weather?q=Atlanta,US&appid=5807ad2a45eb6bf4e81d13
7dafe74e15
```

This request will return a JSON file containing all the information about the current weather situation in the city in question (see Figure 10-5). This JSON file will be submitted for data analysis using the Python pandas library.

Figure 10-5. *The JSON file containing the meteorological data on the city requested*

Data Analysis on Jupyter Notebook

This chapter addresses data analysis using Jupyter Notebook. It allows you to enter and study portions of code gradually.

After the service has started, create a new Notebook.

Start by importing the necessary libraries:

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

The first step is to study the structure of the data received from the site through a specific request.

Choose a city from those chosen for the study, for example Ferrara, and make a request for its current meteorological data, using the URL specified. Without a browser, you can get the text content of a page by using the `request.get()` text function. Because the content obtained is in JSON format, you can directly read the text received following this format with the `json.load()` function.

```
import json
import requests
ferrara = json.loads(requests.get('https://api.openweathermap.org/data/2.5/weather?q=Ferrara,
IT&appid=5807ad2a45eb6bf4e81d137dafe74e15').text)
```

Now you can see the contents of the JSON file with the meteorological data related to the city of Ferrara.

```
ferrara
Out [ ]:
{'coord': {'lon': 11.8333, 'lat': 44.8},
 'weather': [{'id': 804,
   'main': 'Clouds',
   'description': 'overcast clouds',
   'icon': '04d'}],
 'base': 'stations',
 'main': {'temp': 292.51,
  'feels_like': 292.22,
  'temp_min': 292.04,
  'temp_max': 293.74,
  'pressure': 1017,
  'humidity': 66,
  'sea_level': 1017,
  'grnd_level': 1017},
 'visibility': 10000,
```

```
 'wind': {'speed': 2.68, 'deg': 50, 'gust': 7.76},
 'clouds': {'all': 100},
 'dt': 1685511558,
 'sys': {'type': 2,
  'id': 2007888,
  'country': 'IT',
  'sunrise': 1685503859,
  'sunset': 1685558996},
 'timezone': 7200,
 'id': 3177088,
 'name': 'Provincia di Ferrara',
 'cod': 200}
```


When you want to analyze the structure of a JSON file, the following command is useful:

```
list(ferrara.keys())
Out [ ]:
['coord',
 'weather',
 'base',
 'main',
 'visibility',
 'wind',
 'clouds',
 'dt',
 'sys',
 'id',
 'name',
 'cod']
```

This way, you can have a list of all the keys that make up the internal structure of the JSON file. Once you know the name of these keys, you can easily access internal data.

```
print('Coordinates = ', ferrara['coord'])
print('Weather = ', ferrara['weather'])
print('base = ', ferrara['base'])
print('main = ', ferrara['main'])
print('visibility = ', ferrara['visibility'])
print('wind = ', ferrara['wind'])
print('clouds = ', ferrara['clouds'])
print('dt = ', ferrara['dt'])
print('sys = ', ferrara['sys'])
print('id = ', ferrara['id'])
print('name = ', ferrara['name'])
print('cod = ', ferrara['cod'])
Out [ ]:
Coordinates =  {'lon': 11.8333, 'lat': 44.8}
Weather =  [{'id': 804, 'main': 'Clouds', 'description': 'overcast clouds', 'icon': '04d'}]
base =  stations
main =  {'temp': 292.51, 'feels_like': 292.22, 'temp_min': 292.04, 'temp_max': 293.74,
'pressure': 1017, 'humidity': 66, 'sea_level': 1017, 'grnd_level': 1017}
visibility =  10000
```

```
wind =  {'speed': 2.68, 'deg': 50, 'gust': 7.76}
clouds =  {'all': 100}
dt =  1685511558
sys =  {'type': 2, 'id': 2007888, 'country': 'IT', 'sunrise': 1685503859, 'sunset': 
1685558996}
id =  3177088
name =  Provincia di Ferrara
cod =  200
```

Now choose the values that you consider most interesting or useful for this type of analysis. For example, an important value is temperature:

```
ferrara['main']['temp']
Out [ ]:
292.51
```

The purpose of this analysis of the initial structure is to identify the data that could be most important in the JSON structure. These data must be processed for analysis. That is, the data must be extracted from the structure, cleaned or modified according to your needs, and ordered in a dataframe. This way, you can apply all the data analysis techniques presented in this book.

A convenient way to avoid repeating the same code is to insert some extraction procedures into a function, such as the following:

```
def prepare(city,city_name):
    temp = [ ]
    humidity = [ ]
    pressure = [ ]
    description = [ ]
    dt = [ ]
    wind_speed = [ ]
    wind_deg = [ ]
    temp.append(city['main']['temp']-273.15)
    humidity.append(city['main']['humidity'])
    pressure.append(city['main']['pressure'])
    description.append(city['weather'][0]['description'])
    dt.append(city['dt'])
    wind_speed.append(city['wind']['speed'])
    wind_deg.append(city['wind']['deg'])
    headings = ['temp','humidity','pressure','description','dt','wind_speed','wind_deg']
    data = [temp,humidity,pressure,description,dt,wind_speed,wind_deg]
    df = pd.DataFrame(data,index=headings)
    city = df.T
    city['city'] = city_name
    city['day'] = city['dt'].apply(datetime.datetime.fromtimestamp)
    return city
```

This function does nothing more than take the meteorological data you are interested in from the JSON structure, and once they are cleaned or modified (for example, dates and times), that data are collected in a row of a dataframe (as shown in Figure 10-6).

```
t1 = prepare(ferrara,'ferrara')
t1
```

	temp	humidity	pressure	description	dt	wind_speed	wind_deg	city	day
0	19.36	66	1017	overcast clouds	1685511558	2.68	50	ferrara	2023-05-31 07:39:18

Figure 10-6. *The dataframe obtained with the data processed from JSON extraction*

Among all the parameters described in the JSON structure in the list column, these are the most appropriate for the study:

- Temperature
- Humidity
- Pressure
- Description
- Wind speed
- Wind degree

All these properties will be related to the time of acquisition expressed from the `dt` column, which contains a timestamp as the type of data. This value is difficult to read, so you can convert it into a datetime format that allows you to express the date and time in a manner more familiar to you. The new column will be called `day`.

```
city['day'] = city['dt'].apply(datetime.datetime.fromtimestamp)
```

Temperature is expressed in degrees Kelvin, and you can convert these values to Celsius by subtracting 273.15 from each value.

Finally, add the name of the city passed as a second argument of the `prepare()` function.

Data is collected at regular intervals, during different times of the day. For example, you could use a program that executes these requests every hour. Each acquisition will have a row of the dataframe structure that will be added to a general dataframe related to the city, called for example, `df_ferrara` (as shown in Figure 10-7).

```
df_ferrara = t1
t2 = prepare(ferrara,'ferrara')
df_ferrara = pd.concat([df_ferrara, t2])
df_ferrara
```

	temp	humidity	pressure	description	dt	wind_speed	wind_deg	city	day
0	19.36	66	1017	overcast clouds	1685511558	2.68	50	ferrara	2023-05-31 07:39:18
0	19.36	66	1017	overcast clouds	1685511558	2.68	50	ferrara	2023-05-31 07:39:18

Figure 10-7. *The dataframe structure corresponding to a city*

It often happens that data that's useful to this analysis is not present in the JSON source. In that case, you have to resort to other data sources and import the missing data into the structure. In this example, the distances of the cities to the sea are indispensable. You repeat the procedure just described for all the cities in the list that you want to analyze. Then you add the distance values to the dataframe you obtain.

```
.
df_ravenna['dist'] = 8
df_cesena['dist'] = 14
df_faenza['dist'] = 37
df_ferrara['dist'] = 47
df_bologna['dist'] = 71
df_mantova['dist'] = 121
df_piacenza['dist'] = 200
df_milano['dist'] = 250
df_asti['dist'] = 315
df_torino['dist'] = 357
.
```

Analysis of Processed Meteorological Data

For practical purposes, I have already collected data from all the cities involved in the analysis. I have already processed and collected them in a dataframe, which I saved as a CSV file.

If you want to refer to the data used in this chapter, you have to load the ten CSV files that I saved at the time of writing. These files contain data already processed to be used for this analysis.

```
df_ferrara=pd.read_csv('ferrara_270615.csv')
df_milano=pd.read_csv('milano_270615.csv')
df_mantova=pd.read_csv('mantova_270615.csv')
df_ravenna=pd.read_csv('ravenna_270615.csv')
df_torino=pd.read_csv('torino_270615.csv')
df_asti=pd.read_csv('asti_270615.csv')
df_bologna=pd.read_csv('bologna_270615.csv')
df_piacenza=pd.read_csv('piacenza_270615.csv')
df_cesena=pd.read_csv('cesena_270615.csv')
df_faenza=pd.read_csv('faenza_270615.csv')
```

Thanks to the `read_csv()` function of pandas, you can convert CSV files to the dataframe in just one step.

Once you have uploaded data for each city as a dataframe, you can easily see the content.

```
df_cesena
```

As you can see in Figure 10-8, Jupyter Notebook makes it much easier to read dataframes with the generation of graphical tables. Furthermore, you can see that each row shows the measured values for each hour of the day, covering a timeline of about 20 hours in the past.

	Unnamed: 0	temp	humidity	pressure	description	dt	wind_speed	wind_deg	city	day	dist
0	0	23.34	82	1017	very heavy rain	1435387623	1.91	175.511	Cesena	2015-06-27 08:47:03	14
1	1	24.95	69	1018	very heavy rain	1435390801	2.01	159.500	Cesena	2015-06-27 09:40:01	14
2	2	25.67	73	1017	very heavy rain	1435394204	2.10	100.000	Cesena	2015-06-27 10:36:44	14
3	3	26.17	69	1017	very heavy rain	1435398652	3.10	120.000	Cesena	2015-06-27 11:50:52	14
4	4	27.07	61	1016	very heavy rain	1435402083	3.10	110.000	Cesena	2015-06-27 12:48:03	14
5	5	27.41	69	1016	very heavy rain	1435405721	3.60	110.000	Cesena	2015-06-27 13:48:41	14
6	6	27.38	65	1015	very heavy rain	1435409381	5.70	110.000	Cesena	2015-06-27 14:49:41	14
7	7	26.59	65	1014	very heavy rain	1435416585	5.10	110.000	Cesena	2015-06-27 16:49:45	14
8	8	27.16	65	1014	very heavy rain	1435420195	6.20	120.000	Cesena	2015-06-27 17:49:55	14
9	9	27.10	65	1014	very heavy rain	1435423927	6.70	120.000	Cesena	2015-06-27 18:52:07	14
10	10	26.01	73	1013	very heavy rain	1435427556	6.20	120.000	Cesena	2015-06-27 19:52:36	14
11	11	23.37	94	1015	very heavy rain	1435438070	2.60	90.000	Cesena	2015-06-27 22:47:50	14
12	12	22.48	83	1016	very heavy rain	1435441857	5.70	90.000	Cesena	2015-06-27 23:50:57	14
13	13	21.94	83	1016	very heavy rain	1435445495	2.10	210.000	Cesena	2015-06-28 00:51:35	14
14	14	20.26	94	1016	very heavy rain	1435452847	2.01	107.004	Cesena	2015-06-28 02:54:07	14
15	15	19.65	93	1016	very heavy rain	1435456185	2.10	330.000	Cesena	2015-06-28 03:49:45	14
16	16	19.29	93	1016	very heavy rain	1435459689	1.50	320.000	Cesena	2015-06-28 04:48:09	14
17	17	18.41	93	1016	very heavy rain	1435463462	0.50	300.000	Cesena	2015-06-28 05:51:02	14
18	18	19.48	88	1016	very heavy rain	1435466850	0.50	270.000	Cesena	2015-06-28 06:47:30	14
19	19	22.00	88	1016	very heavy rain	1435470541	2.10	260.000	Cesena	2015-06-28 07:49:01	14

Figure 10-8. *The dataframe structure corresponding to a city*

In the case shown in Figure 10-8, note that there are only 19 rows. In fact, observing other cities, it looks like the meteorological measurement systems sometimes failed during the measuring process, leaving holes in the acquisition. If the data collected make up 19 rows, as in this case, they are sufficient to describe the trend of the meteorological properties during the day. However, it is good practice to check the size of all ten dataframes. If a city provides insufficient data to describe the daily trend, you need to replace it with another city.

There is an easy way to check the size, without having to put one table after another. Thanks to the `shape()` function, you can determine the number of data acquired (lines) for each city.

```
print(df_ferrara.shape)
print(df_milano.shape)
print(df_mantova.shape)
print(df_ravenna.shape)
print(df_torino.shape)
print(df_asti.shape)
print(df_bologna.shape)
print(df_piacenza.shape)
print(df_cesena.shape)
print(df_faenza.shape)
```

This will give the following result:

```
(20, 9)
(18, 9)
(20, 9)
```

```
(18, 9)
(20, 9)
(20, 9)
(20, 9)
(20, 9)
(20, 9)
(19, 9)
```

As you can see, the choice of ten cities is optimal, since the control units have provided enough data to continue with the data analysis.

A normal way to analyze the data you just collected is to use data visualization. You saw that the `matplotlib` library includes a set of tools to generate charts on which to display data. In fact, data visualization helps you during data analysis to discover features of the system you are studying.

Next, activate the necessary libraries:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

For example, there is a simple way to analyze the trend of the temperature during the day. Consider the city of Milan.

```
y1 = df_milano['temp']
df_milano['day'] = pd.to_datetime(df_milano['day'])
x1 = df_milano['day']
fig, ax = plt.subplots()
plt.xticks(rotation=70)
hours = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(hours)
ax.plot(x1,y1,'r')
```

Executing this code, you get the graph shown in Figure 10-9. As you can see, the temperature trend follows a nearly sinusoidal pattern characterized by a temperature that rises in the morning, to reach the maximum value during the heat of the afternoon (between 2:00 and 6:00 pm). Then the temperature decreases to a minimum value corresponding to just before dawn, that is, at 6:00 am.

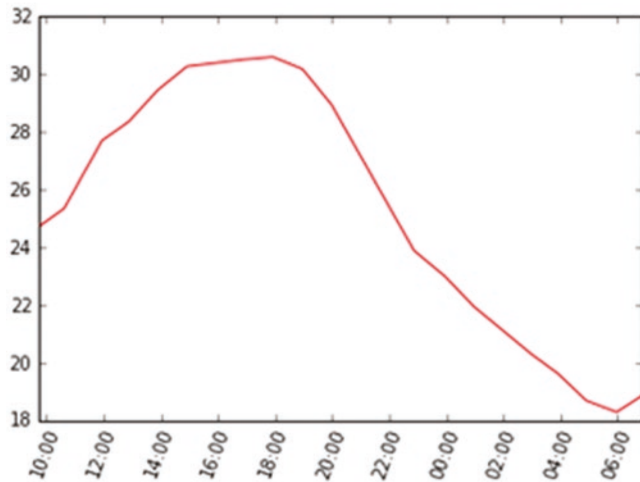

Figure 10-9. *Temperature trend of Milan during the day*

Because the purpose of your analysis is to try to interpret the weather, it is possible to assess how and if the sea influences this trend. This time, you evaluate the trends of different cities simultaneously. This is the only way to see if the analysis is going in the right direction. Thus, choose the three cities closest to the sea and the three cities farthest from it.

```
y1 = df_ravenna['temp']
df_ravenna['day'] = pd.to_datetime(df_ravenna['day'])
x1 = df_ravenna['day']
y2 = df_faenza['temp']
df_faenza['day'] = pd.to_datetime(df_faenza['day'])
x2 = df_faenza['day']
y3 = df_cesena['temp']
df_cesena['day'] = pd.to_datetime(df_cesena['day'])
x3 = df_cesena['day']
y4 = df_milano['temp']
df_milano['day'] = pd.to_datetime(df_milano['day'])
x4 = df_milano['day']
y5 = df_asti['temp']
df_asti['day'] = pd.to_datetime(df_asti['day'])
x5 = df_asti['day']
y6 = df_torino['temp']
df_torino['day'] = pd.to_datetime(df_torino['day'])
x6 = df_torino['day']
fig, ax = plt.subplots()
plt.xticks(rotation=70)
hours = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(hours)
plt.plot(x1,y1,'r',x2,y2,'r',x3,y3,'r')
plt.plot(x4,y4,'g',x5,y5,'g',x6,y6,'g')
```

This code will produce the chart shown in Figure 10-10. The temperatures of the three cities closest to the sea are shown in red, while the temperatures of the three cities farthest away are in green.

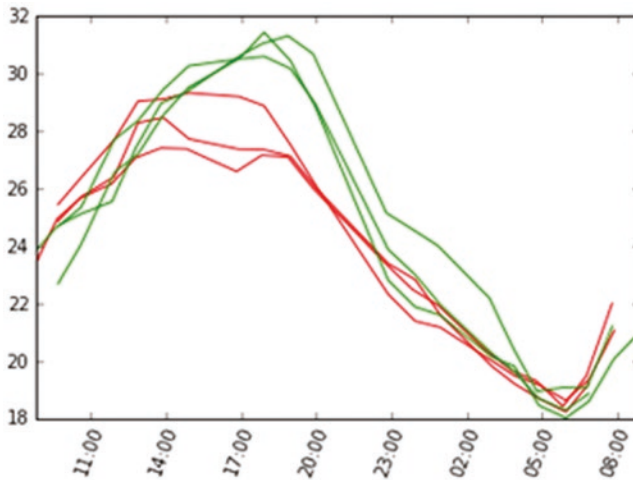

Figure 10-10. *The trend of the temperatures of six different cities (red is the closest to the sea; green is the farthest)*

Looking at Figure 10-10, the results seem promising. In fact, the three closest cities have maximum temperatures much lower than those farthest away, whereas there seems to be little difference in the minimum temperatures.

In order to go deep into this aspect, you can collect the maximum and minimum temperatures of all ten cities and display a line chart that charts these temperatures compared to their distances from the sea.

```
dist = [df_ravenna['dist'][0],
     df_cesena['dist'][0],
     df_faenza['dist'][0],
     df_ferrara['dist'][0],
     df_bologna['dist'][0],
     df_mantova['dist'][0],
     df_piacenza['dist'][0],
     df_milano['dist'][0],
     df_asti['dist'][0],
     df_torino['dist'][0]
]temp_max = [df_ravenna['temp'].max(),
     df_cesena['temp'].max(),
     df_faenza['temp'].max(),
     df_ferrara['temp'].max(),
     df_bologna['temp'].max(),
     df_mantova['temp'].max(),
     df_piacenza['temp'].max(),
     df_milano['temp'].max(),
     df_asti['temp'].max(),
     df_torino['temp'].max()
]
temp_min = [df_ravenna['temp'].min(),
     df_cesena['temp'].min(),
     df_faenza['temp'].min(),
     df_ferrara['temp'].min(),
```

```
     df_bologna['temp'].min(),
     df_mantova['temp'].min(),
     df_piacenza['temp'].min(),
     df_milano['temp'].min(),
     df_asti['temp'].min(),
     df_torino['temp'].min()
]
```

Start by representing the maximum temperatures.

```
plt.plot(dist,temp_max,'ro')
```

The result is shown in Figure 10-11.

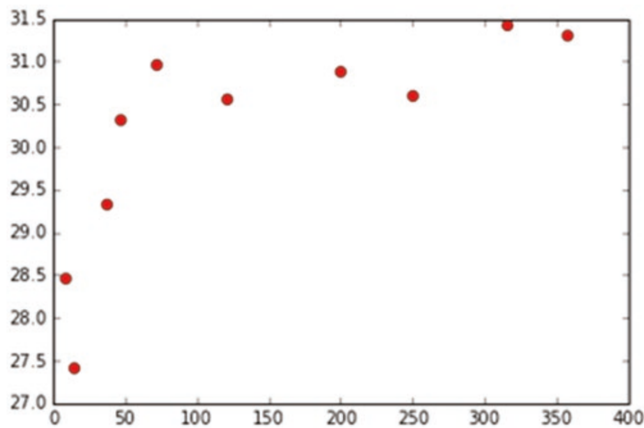

Figure 10-11. *Trend of maximum temperature in relation to distance from the sea*

As shown in Figure 10-11, you can affirm the hypothesis that the presence of the sea somehow influences meteorological parameters is true (at least in the day today ☺).

Furthermore, you can see that the effect of the sea decreases rapidly, and after about 60-70 km, the maximum temperatures reach a plateau.

An interesting idea would be to represent the two different trends with two straight lines obtained by linear regression. To do this, you can use the SVR method provided by the `scikit-learn` library.

If you haven't installed the `scikit-learn` library yet, do so now. If you are working with Anaconda, you can install it via Anaconda Navigator by selecting it from the packages available in your virtual environment (see Figure 10-12), or from the CMD.exe Prompt console, by entering this command:

```
conda install scikit-learn
```

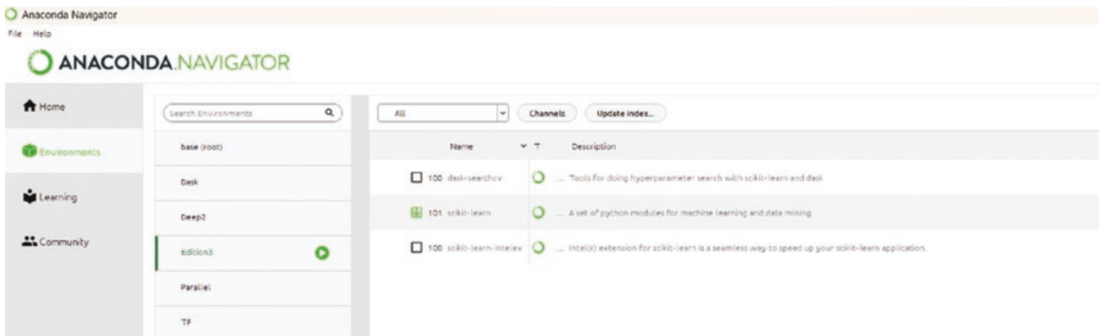

Figure 10-12. Installation of the scikit-learn library with Anaconda Navigator

At this point, you can continue with the example by inserting the following code into a cell in the Notebook:

```
x = np.array(dist)
y = np.array(temp_max)
x1 = x[x<100]
x1 = x1.reshape((x1.size,1))
y1 = y[x<100]
x2 = x[x>50]
x2 = x2.reshape((x2.size,1))
y2 = y[x>50]
from sklearn.svm import SVR
svr_lin1 = SVR(kernel='linear', C=1e3)
svr_lin2 = SVR(kernel='linear', C=1e3)
svr_lin1.fit(x1, y1)
svr_lin2.fit(x2, y2)
xp1 = np.arange(10,100,10).reshape((9,1))
xp2 = np.arange(50,400,50).reshape((7,1))
yp1 = svr_lin1.predict(xp1)
yp2 = svr_lin2.predict(xp2)
plt.plot(xp1, yp1, c='r', label='Strong sea effect')
plt.plot(xp2, yp2, c='b', label='Light sea effect')
plt.axis((0,400,27,32))
plt.scatter(x, y, c='k', label='data')
```

This code will produce the chart shown in Figure 10-13.

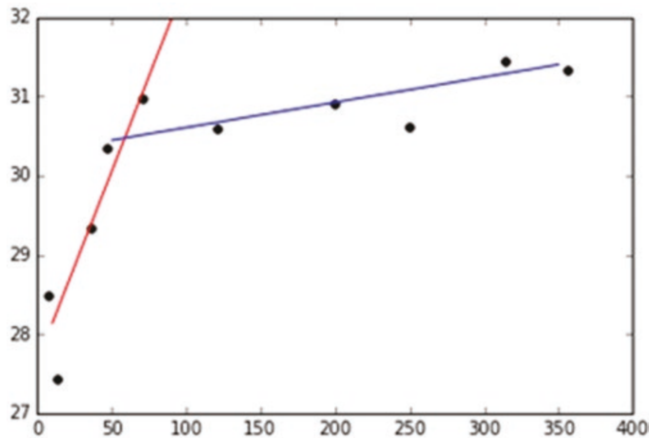

Figure 10-13. *The two trends described by the maximum temperatures in relation to distance*

As you can see, temperature increase in the first 60 km is very rapid, rising from 28 to 31 degrees. It then increases very mildly (if at all) over longer distances. The two trends are described by two straight lines that have the following expression

$$x = ax + b$$

where *a* is the slope and the *b* is the intercept.

```
print( svr_lin1.coef_)
print( svr_lin1.intercept_)
print( svr_lin2.coef_)
print( svr_lin2.intercept_)
Out [ ]:
[[-0.04794118]]
[ 27.65617647]
[[-0.00317797]]
[ 30.2854661]
```

You might consider the intersection point of the two lines as the point between the area where the sea exerts its influence and the area where it doesn't, or at least not as strongly.

```
from scipy.optimize import fsolve
def line1(x):
    a1 = svr_lin1.coef_[0][0]
    b1 = svr_lin1.intercept_[0]
    return a1*x + b1
def line2(x):
    a2 = svr_lin2.coef_[0][0]
    b2 = svr_lin2.intercept_[0]
    return a2*x + b2
def findIntersection(fun1,fun2,x0):
 return fsolve(lambda x : fun1(x) - fun2(x),x0)
result = findIntersection(line1,line2,0.0)
print("[x,y] = [ %d , %d ]" % (result,line1(result)))
```

```
x = np.linspace(0,300,31)
plt.plot(x,line1(x),x,line2(x),result,line1(result),'ro')
```

Executing the code, you can find the point of intersection as follows:

```
Out [ ]:
[x,y] = [ 58, 30 ]
```

This point is represented in the chart shown in Figure 10-14.

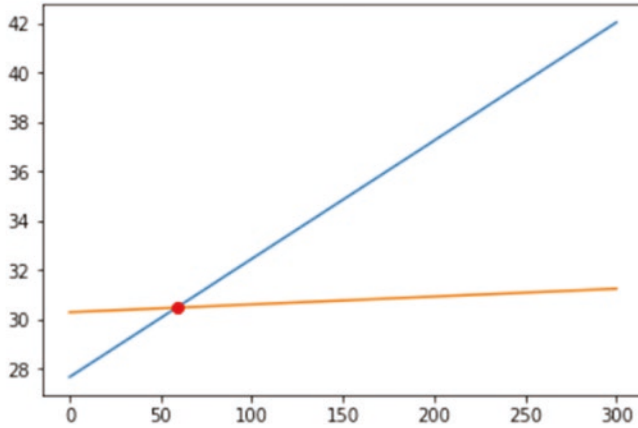

Figure 10-14. *The point of intersection between two straight lines obtained by linear regression*

You can say that the average distance in which the effects of the sea vanish is 58 km.
Now you can analyze the minimum temperatures.

```
plt.axis((0,400,15,25))
plt.plot(dist,temp_min,'bo')
```

Doing this, you'll obtain the chart shown in Figure 10-15.

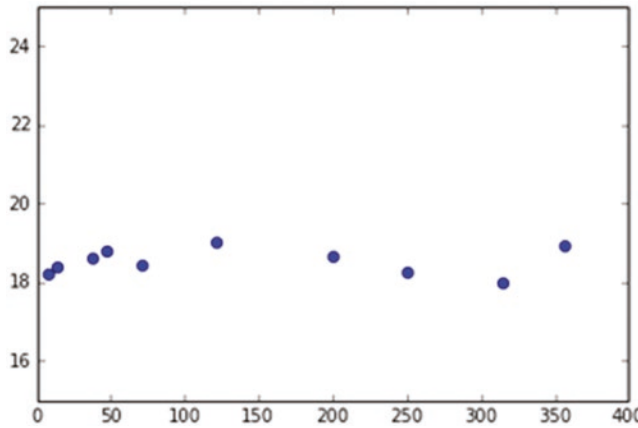

Figure 10-15. *The minimum temperatures appear to be independent of the distance from the sea*

In this case, it appears very clear that the sea has no effect on minimum temperatures recorded during the night, or rather, around six in the morning. If I remember well, when I was a child I was taught that the sea mitigated the cold temperatures, or that the sea released the heat absorbed during the day. This does not seem to be the case. This case tracks summer in Italy; it would be interesting to see if this hypothesis is true in the winter or somewhere else.

Another meteorological measure contained in the ten dataframes is the humidity. Even for this measure, you can see the trend of the humidity during the day for the three cities closest to the sea and for the three farthest away.

```
y1 = df_ravenna['humidity']
x1 = df_ravenna['day']
y2 = df_faenza['humidity']
x2 = df_faenza['day']
y3 = df_cesena['humidity']
x3 = df_cesena['day']
y4 = df_milano['humidity']
x4 = df_milano['day']
y5 = df_asti['humidity']
x5 = df_asti['day']
y6 = df_torino['humidity']
x6 = df_torino['day']
fig, ax = plt.subplots()
plt.xticks(rotation=70)
hours = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(hours)
plt.plot(x1,y1,'r',x2,y2,'r',x3,y3,'r')
plt.plot(x4,y4,'g',x5,y5,'g',x6,y6,'g')
```

This code will create the chart shown in Figure 10-16.

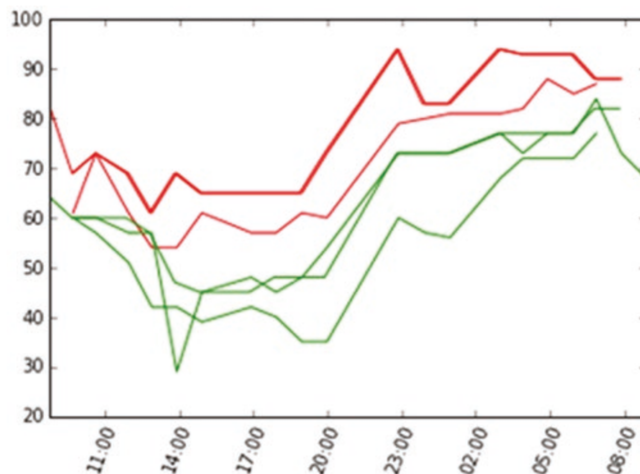

Figure 10-16. *The trend of the humidity during the day for three cities nearest the sea (shown in red) and three cities farthest away (indicated in green)*

At first glance, it would seem that the cities closest to the sea experience more humidity than those farthest away and that this difference in moisture (about 20 percent) extends throughout the day. You can see if this remains true when you report the maximum and minimum humidity with respect to the distances from the sea.

```
hum_max = [df_ravenna['humidity'].max(),
     df_cesena['humidity'].max(),
     df_faenza['humidity'].max(),
     df_ferrara['humidity'].max(),
     df_bologna['humidity'].max(),
     df_mantova['humidity'].max(),
     df_piacenza['humidity'].max(),
     df_milano['humidity'].max(),
     df_asti['humidity'].max(),
     df_torino['humidity'].max()
]
plt.plot(dist,hum_max,'bo')
```

The maximum humidity of ten cities according to their distance from the sea is represented in the chart in Figure 10-17.

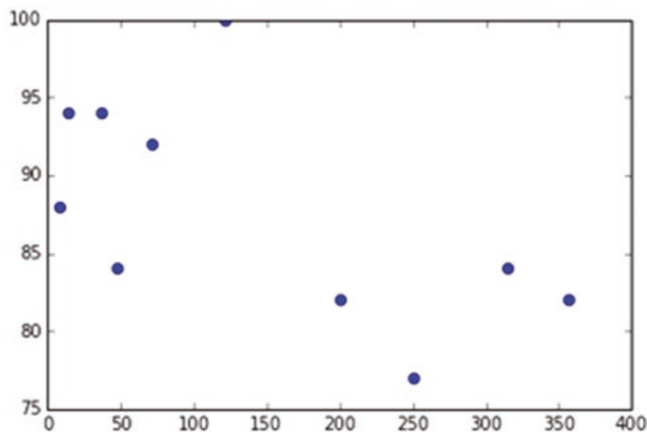

Figure 10-17. *The trend of the maximum humidity function with respect to the distance from the sea*

```
hum_min = [df_ravenna['humidity'].min(),
     df_cesena['humidity'].min(),
     df_faenza['humidity'].min(),
     df_ferrara['humidity'].min(),
     df_bologna['humidity'].min(),
     df_mantova['humidity'].min(),
     df_piacenza['humidity'].min(),
     df_milano['humidity'].min(),
     df_asti['humidity'].min(),
     df_torino['humidity'].min()
]
plt.plot(dist,hum_min,'bo')
```

The minimum humidity of ten cities according to their distance from the sea is represented in the chart in Figure 10-18.

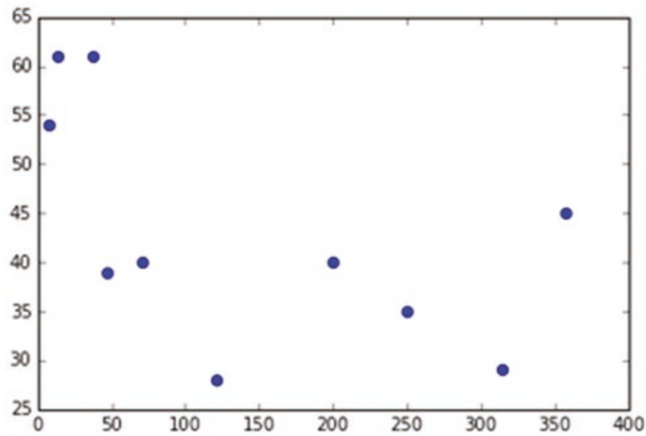

Figure 10-18. *The trend of the minimum humidity as a function of distance from the sea*

Looking at Figures 10-17 and 10-18, you can certainly see that the humidity, both the minimum and maximum, is greater in the cities closest to the sea. However, in my opinion, it is not possible to say that there is a linear relationship or some other kind of relationship to draw a curve. The collected points (ten) are too few to highlight a trend in this case.

The RoseWind

Among the various meteorological data that were collected for each city are those related to the wind:

- Wind degree (direction)
- Wind speed

If you analyze the dataframe, you will notice that the wind speed is relative to the direction it blows and the time of day. For instance, each measurement shows the direction in which the wind blows (see Figure 10-19).

	wind_deg	wind_speed	day
0	159.5000	2.01	2015-06-27 09:42:05
1	100.0000	2.10	2015-06-27 10:37:24
2	80.0000	4.60	2015-06-27 11:57:01
3	90.0000	4.60	2015-06-27 12:53:43
4	80.0000	6.20	2015-06-27 13:54:20
5	80.0000	6.70	2015-06-27 14:55:06
6	90.0000	6.70	2015-06-27 16:55:00
7	90.0000	5.70	2015-06-27 17:55:43
8	90.0000	4.60	2015-06-27 18:58:17
9	97.0000	2.06	2015-06-27 19:58:58
10	89.0000	2.06	2015-06-27 22:52:39
11	88.0147	2.86	2015-06-27 23:57:25
12	107.0040	2.01	2015-06-28 00:57:46
13	107.0040	2.01	2015-06-28 03:00:34
14	132.5030	1.06	2015-06-28 03:54:49
15	132.5030	1.06	2015-06-28 04:54:04
16	132.5030	1.06	2015-06-28 05:58:15
17	251.0000	1.54	2015-06-28 06:52:59

Figure 10-19. *The wind data contained in the dataframe*

To better analyze this kind of data, it is necessary to visualize them. In this case, a linear chart in Cartesian coordinates is not the most optimal approach.

If you use the classic scatterplot with the points contained in a single dataframe:

```
plt.plot(df_ravenna['wind_deg'],df_ravenna['wind_speed'],'ro')
```

You get a chart like the one shown in Figure 10-20, which certainly is not very educational.

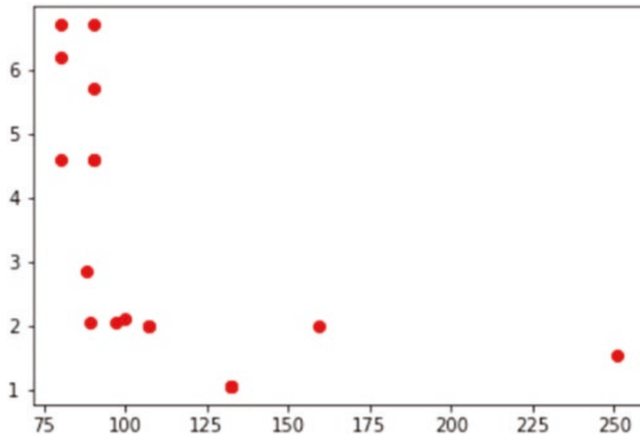

Figure 10-20. *A scatterplot representing a distribution of 360 degrees*

To represent a distribution of points in 360 degrees, it's best to use another type of visualization: the *polar chart*. You have already seen this kind of chart in Chapter 8.

First you need to create a histogram, whereby the data are distributed over the interval of 360 degrees divided into eight bins, each of which is 45 degrees.

```
hist, bins = np.histogram(df_ravenna['wind_deg'],8,[0,360])
print(hist)
print(bins)
```

The values returned are occurrences within each bin expressed by an array called `hist`:

```
Out [ ]:
[ 0  5 11  1  0  1  0  0]
```

and an array called `bins`, which defines the edges of each bin within the range of 360 degrees.

```
Out [ ]:
 [   0.   45.   90.  135.  180.  225.  270.  315.  360.]
```

These arrays will be useful to correctly define the polar chart to be drawn. For this purpose, you have to create a function in part by using the code contained in Chapter 8. This function is called `showRoseWind()`, and it will need three different arguments: `values` is the array containing the values to be displayed, which in this case is the `hist` array; `city_name` is a string containing the name of the city to be shown as the chart title; and `max_value` is an integer that establishes the maximum value for presenting the blue color.

Defining a function of this kind helps you avoid rewriting the same code many times, and it produces more modular code, which allows you to focus on the concepts related to a particular operation within a function.

```
def showRoseWind(values,city_name,max_value):
   N = 8
   theta = np.arange(0.,2 * np.pi, 2 * np.pi / N)
   radii = np.array(values)
   plt.axes([0.025, 0.025, 0.95, 0.95], polar=True)
```

```
    colors = [(1-x/max_value, 1-x/max_value, 0.75) for x in radii]
    plt.bar(theta +np.pi/8, radii, width=(2*np.pi/N), bottom=0.0, color=colors)
    plt.title(city_name,x=0.2, fontsize=20)
```

One thing that changed is the color map. In this case, the closer to blue the slice is, the greater the value it represents.

Once you define a function, you can use it:

```
showRoseWind(hist,'Ravenna',max(hist))
```

Executing this code, you will obtain a polar chart like the one shown in Figure 10-21.

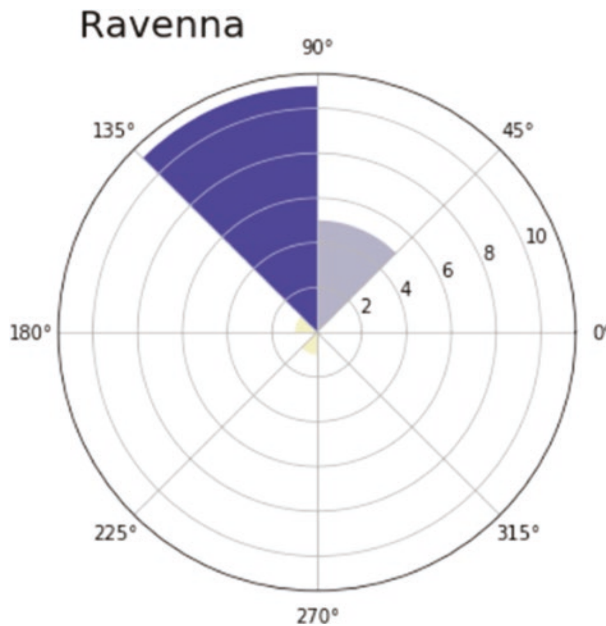

Figure 10-21. *The polar chart represents the distribution of values within a range of 360 degrees*

As you can see in Figure 10-20, you have a range of 360 degrees divided into eight areas of 45 degrees each (bin), in which a scale of values is represented radially. In each of the eight areas, a slice is represented with a variable length that corresponds precisely to the corresponding value. The more radially extended the slice is, the greater the value represented. In order to increase the readability of the chart, a color scale has been entered that corresponds to the extension of its slice. The wider the slice is, the more the color tends to a deep blue.

This polar chart provides you with information about how the wind direction will be distributed radially. In this case, the wind has blown purely toward the southwest/west most of the day.

Once you have defined the `showRoseWind` function, it is very easy to observe the winds with respect to any of the ten sample cities.

```
hist, bin = np.histogram(df_ferrara['wind_deg'],8,[0,360])
print(hist)
showRoseWind(hist,'Ferrara', 15.0)
Out [ ]:
[7 2 3 3 3 2 0 0]
```

Figure 10-22 shows the polar charts of the ten cities.

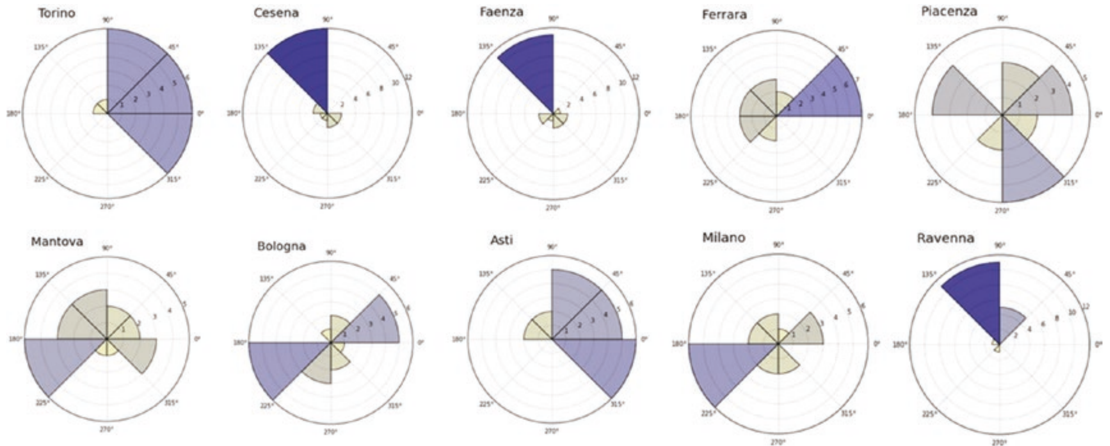

Figure 10-22. *The polar charts display the distribution of the wind direction*

Calculating the Mean Distribution of the Wind Speed

Even the other quantity that relates the speed of the winds can be represented as a distribution on 360 degrees.

Now define a feature called RoseWind_Speed that will allow you to calculate the mean wind speeds for each of the eight bins into which 360 degrees are divided.

```
def RoseWind_Speed(df_city):
   degs = np.arange(45,361,45)
   tmp =  []
   for deg in degs:
      tmp.append(df_city[(df_city['wind_deg']>(deg-46)) & (df_city['wind_deg']<deg)]['wind_
      speed'].mean())
   return np.nan_to_num(tmp)
```

This function returns a NumPy array containing the eight mean wind speeds. This array will be used as the first argument of the ShowRoseWind_Speed() function, which is an improved version of the previous ShowRoseWind() function used to represent the polar chart.

```
def showRoseWind_Speed(speeds,city_name):
    N = 8
    theta = np.arange(0,2 * np.pi, 2 * np.pi / N)
    radii = np.array(speeds)
    plt.axes([0.025, 0.025, 0.95, 0.95], polar=True)
    colors = [(1-x/10.0, 1-x/10.0, 0.75) for x in radii]
    bars = plt.bar(theta+np.pi/8, radii, width=(2*np.pi/N), bottom=0.0, color=colors)
    plt.title(city_name,x=0.2, fontsize=20)
showRoseWind_Speed(RoseWind_Speed(df_ravenna),'Ravenna')
```

Figure 10-23 represents the RoseWind corresponding to the wind speeds distributed around 360 degrees.

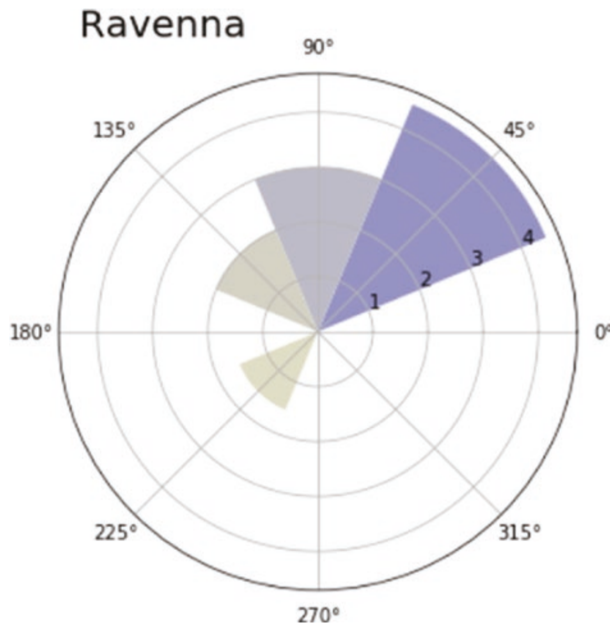

Figure 10-23. *This polar chart represents the distribution of wind speeds within 360 degrees*

At the end of all this work, you can save the dataframe as a CSV file, thanks to the `to_csv ()` function of the pandas library.

```
df_ferrara.to_csv('ferrara.csv')
df_milano.to_csv('milano.csv')
df_mantova.to_csv('mantova.csv')
df_ravenna.to_csv('ravenna.csv')
df_torino.to_csv('torino.csv')
df_asti.to_csv('asti.csv')
df_bologna.to_csv('bologna.csv')
df_piacenza.to_csv('piacenza.csv')
df_cesena.to_csv('cesena.csv')
df_faenza.to_csv('faenza.csv')
```

Conclusions

The purpose of this chapter was mainly to show how you can get information from raw data. Some of this information will not lead to important conclusions, while other information will lead to the confirmation of a hypothesis, thus increasing your state of knowledge. These are the cases in which data analysis has led to a success.

In the next chapter, you see another case related to real data obtained from an open data source. You also see how you can further enhance the graphical representation of the data using the D3 JavaScript library. This library, although not Python, can be easily integrated into Python.