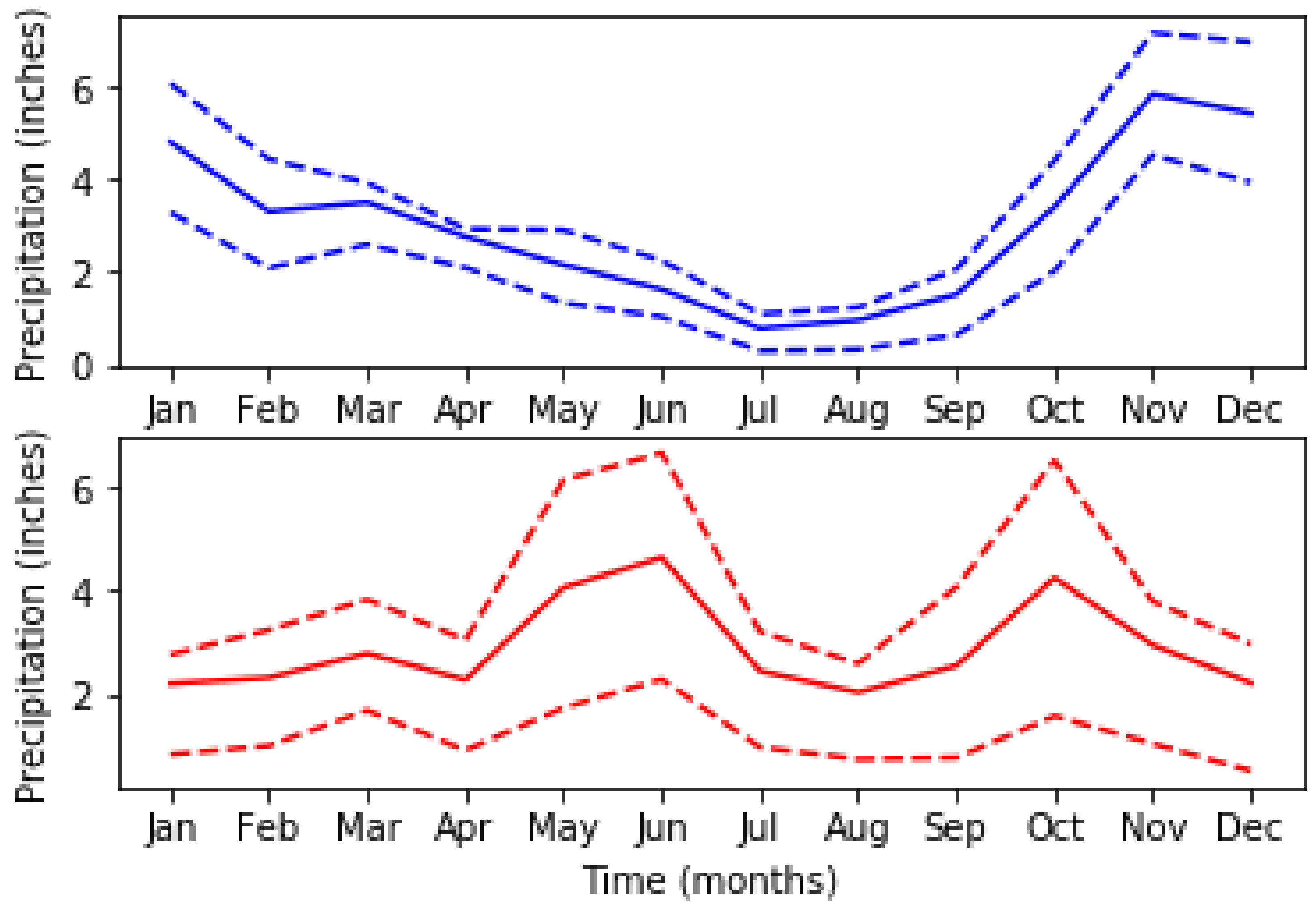
Plotting time-series data

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

Time-series data



Climate change time-series

```
date,co2,relative_temp
1958-03-06,315.71,0.1
1958-04-06,317.45,0.01
1958-05-06,317.5,0.08
1958-06-06,-99.99,-0.05
1958-07-06,315.86,0.06
1958-08-06,314.93,-0.06
...
2016-08-06,402.27,0.98
2016-09-06,401.05,0.87
2016-10-06,401.59,0.89
2016-11-06,403.55,0.93
2016-12-06,404.45,0.81
```

3. Climate change time-series

Let's look at a more complex dataset, that contains records of the change in climate in the last half a century or so. The data is in a CSV file with three columns. The "date" column indicates when the recording was made and is stored in the year-month-date format. A measurement was taken on the 6th day of every month from 1958 until 2016. The column "co2" contains measurements of the carbon dioxide in the atmosphere. The number shown in each row is parts-per-million of carbon dioxide. The column "relative-underscore-temp" denotes the temperature measured at this date, relative to a baseline which is the average temperature in the first ten years of measurements. If we want Pandas to recognize that this is a time-series, we'll need to tell it to parse the "date" column as a date. To use the full power of Pandas indexing facilities, **we'll also designate the date column as our index by using the index-underscore-col key-word argument.**

DateTimeIndex

```
climate_change.index
```

```
DatetimeIndex(['1958-03-06', '1958-04-06', '1958-05-06', '1958-06-06',
               '1958-07-06', '1958-08-06', '1958-09-06', '1958-10-06',
               '1958-11-06', '1958-12-06',
               ...
               '2016-03-06', '2016-04-06', '2016-05-06', '2016-06-06',
               '2016-07-06', '2016-08-06', '2016-09-06', '2016-10-06',
               '2016-11-06', '2016-12-06'],
              dtype='datetime64[ns]', name='date', length=706, freq=None)
```

4. DateTimeIndex
This is the index of our DataFrame. It's a DateTimeIndex object with 706 entries, one for each measurement. It has a DateTime datatype and Matplotlib will recognize that this is a variable that represents time. This will be important in a little bit.

Time-series data

```
climate_change['relative_temp']
```

```
0       0.10
1       0.01
2       0.08
3      -0.05
4       0.06
5      -0.06
6      -0.03
7       0.04
       ...
701     0.98
702     0.87
703     0.89
704     0.93
705     0.81
Name:co2, Length: 706, dtype: float64
```

```
climate_change['co2']
```

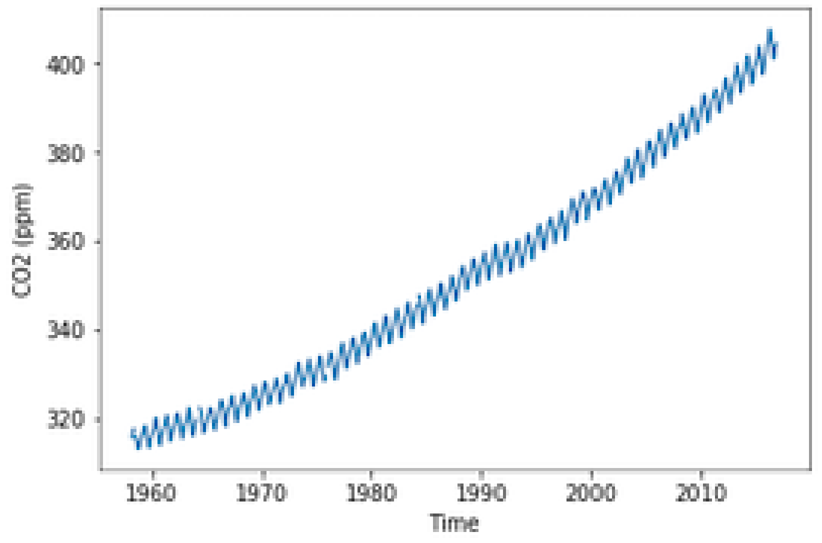
```
0       315.71
1       317.45
2       317.50
3          NaN
4       315.86
5       314.93
6       313.20
7          NaN
       ...
701     402.27
702     401.05
703     401.59
704     403.55
705     404.45
Name:co2, Length: 706, dtype: float64
```

5. Time-series data
The other two columns in the data are stored as regular columns of the DataFrame with a floating point data-type, which will allow us to calculate on them as continuous variables.
There are a few points in the CO2 data that are stored as NaNs or Not-a-Number. These are missing values where measurements were not taken.

Plotting time-series data

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
```

```
ax.plot(climate_change.index, climate_change['co2'])
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)')
plt.show()
```



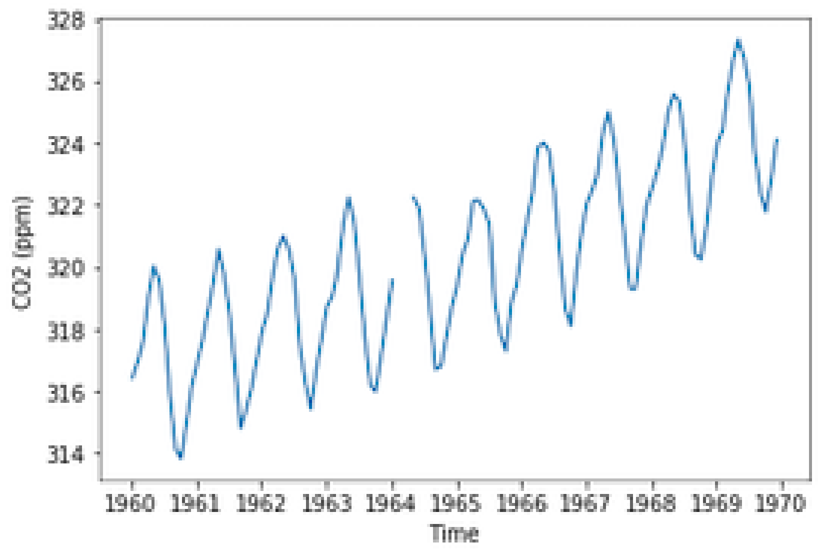
6. Plotting time-series data
To start plotting the data, we import Matplotlib and create a Figure and Axes. Next, we add the data to the plot. We add the index of our DataFrame for the x-axis and the "co2" column for the y-axis.

Zooming in on a decade

```
sixties = climate_change["1960-01-01":"1969-12-31"]
```

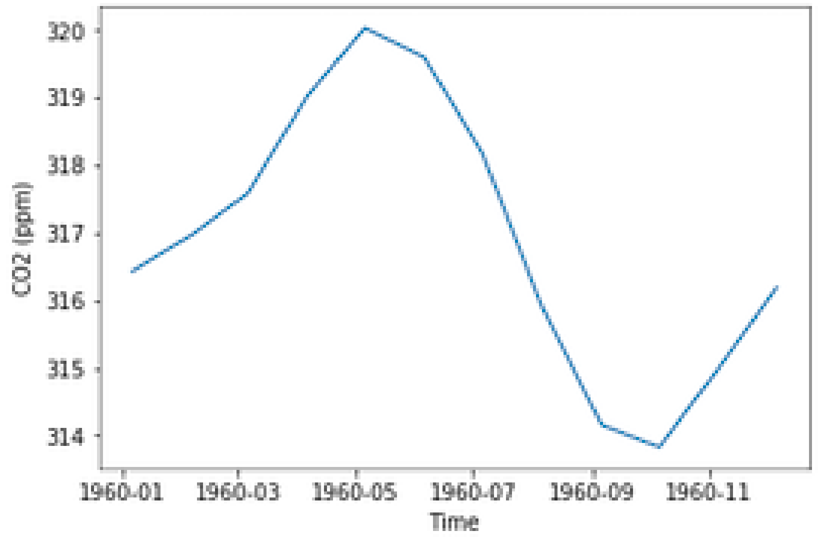
```
fig, ax = plt.subplots()
ax.plot(sixties.index, sixties['co2'])
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)')
plt.show()
```

7. Zooming in on a decade
We can select a decade of the data by slicing into the DataFrame with two strings that delimit the start date and end date of the period that we are interested in. When we do that, we get the plot of a part of the time-series encompassing only ten years worth of data. Matplotlib also now knows to label the x-axis ticks with years, with an interval of one year between ticks. Looking at this data, you'll also notice that the missing values in this time series are represented as breaks in the line plotted by Matplotlib.



Zooming in on one year

```
sixty_nine = climate_change["1969-01-01":"1969-12-31"]
fig, ax = plt.subplots()
ax.plot(sixty_nine.index, sixty_nine['co2'])
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)')
plt.show()
```



Let's practice time-series plotting!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Plotting time-series with different variables

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

Plotting two time-series together

```
import pandas as pd
climate_change = pd.read_csv('climate_change.csv',
                             parse_dates=["date"],
                             index_col="date")
```

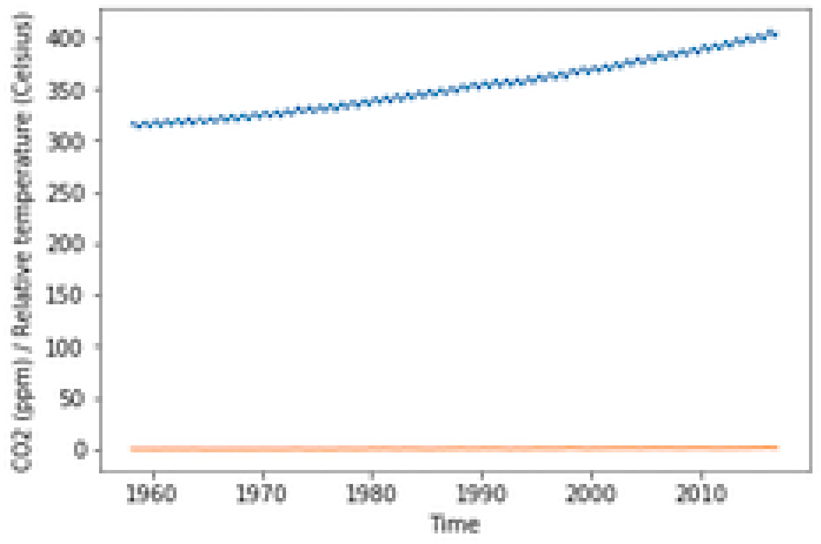
```
climate_change
```

```
              co2  relative_temp
date
1958-03-06  315.71           0.10
1958-04-06  317.45           0.01
1958-07-06  315.86           0.06
...            ...            ...
2016-11-06  403.55           0.93
2016-12-06  404.45           0.81

[706 rows x 2 columns]
```

Plotting two time-series together

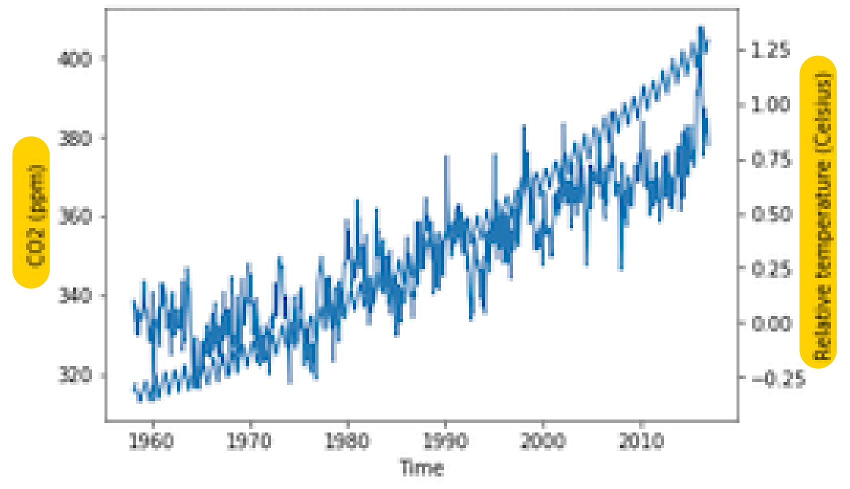
```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change["co2"])
ax.plot(climate_change.index, climate_change["relative_temp"])
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm) / Relative temperature')
plt.show()
```



3. Plotting two time-series together
As before, we can create a Figure and Axes and add the data from one variable to the plot. And we can add the data from the other variable to the plot. We also add axis labels and show the plot. But this doesn't look right. The line for carbon dioxide has shifted upwards, and the line for relative temperatures looks completely flat. The problem is that the scales for these two measurements are different.

Using twin axes

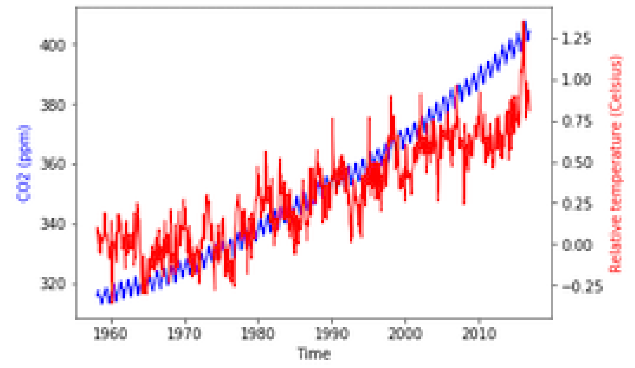
```
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change["co2"])
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)')
ax2 = ax.twinx()
ax2.plot(climate_change.index, climate_change["relative_temp"])
ax2.set_ylabel('Relative temperature (Celsius)')
plt.show()
```



4. Using twin axes
You've already seen how you could plot these time-series in separate sub-plots. Here, we're going to plot them in the same sub-plot, using two different y-axis scales. Again, we start by adding the first variable to our Axes. Then, we use the twinx method to create a twin of this Axes. This means that the two Axes share the same x-axis, but the y-axes are separate. We add the other variable to this second Axes object and show the figure. There is one y-axis scale on the left, for the carbon dioxide variable, and another y-axis scale to the right for the temperature variable. Now you can see the fluctuations in temperature more clearly. But this is still not quite right. The two lines have the same color. Let's take care of that.

Separating variables by color

```
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change["co2"], color='blue')
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)', color='blue')
ax2 = ax.twinx()
ax2.plot(climate_change.index, climate_change["relative_temp"],
         color='red')
ax2.set_ylabel('Relative temperature (Celsius)', color='red')
plt.show()
```



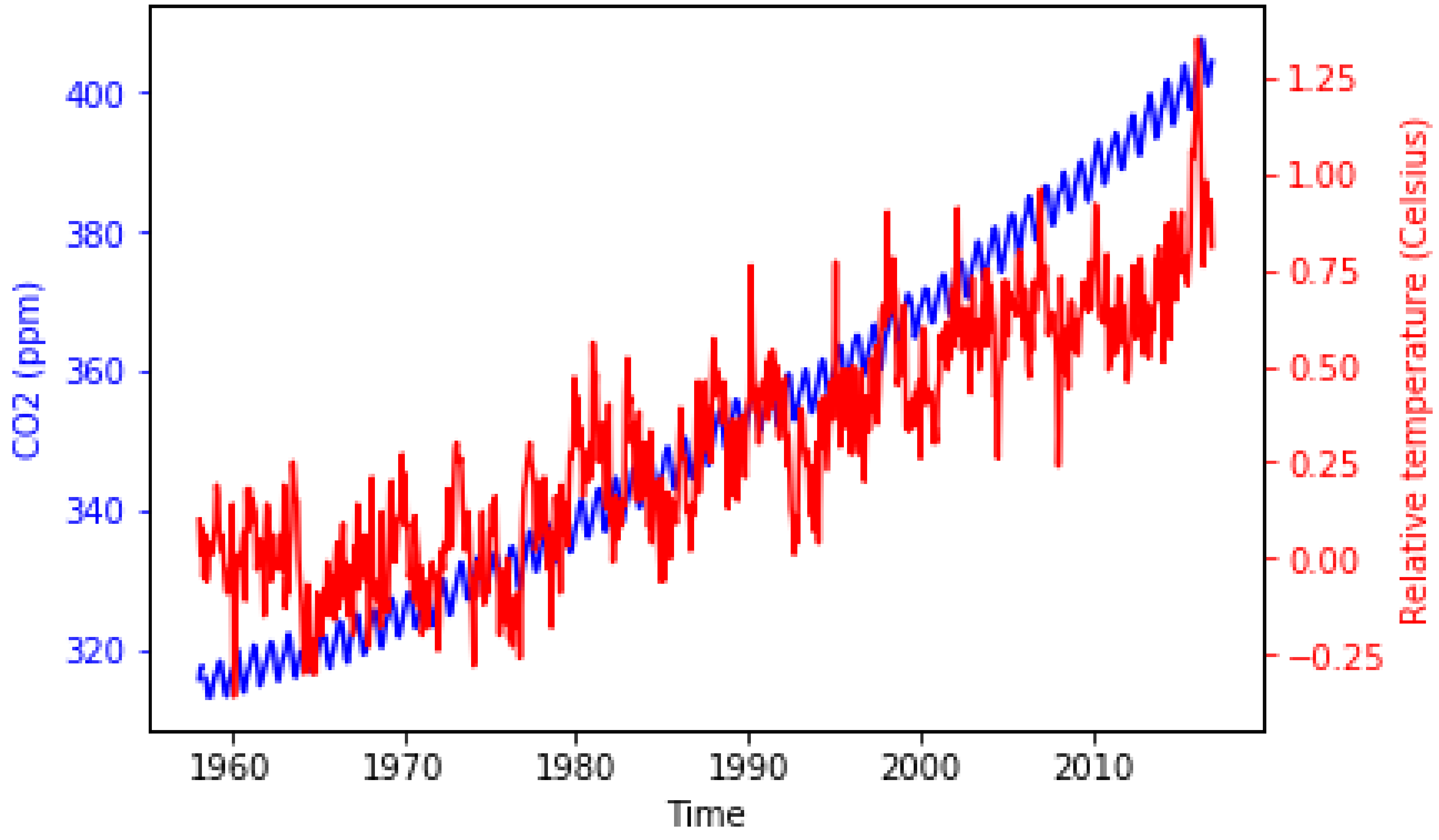
5. Separating variables by color
To separate the variables, we'll encode each one with a different color. We add color to the first variable, using the color key-word argument in the call to the plot function. We also set the color in our call to the set-underscore-ylabel function. We repeat this in our calls to plot and set-underscore-ylabel from the twin Axes object. In the resulting figure, each variable has its own color and the y-axis labels clearly tell us which scale belongs to which variable.

Coloring the ticks

```
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change["co2"],
        color='blue')
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)', color='blue')
ax.tick_params('y', colors='blue')
ax2 = ax.twinx()
ax2.plot(climate_change.index,
         climate_change["relative_temp"],
         color='red')
ax2.set_ylabel('Relative temperature (Celsius)',
color='red')
ax2.tick_params('y', colors='red')
plt.show()
```

6. Coloring the ticks
We can make encoding by color even more distinct by setting not only the color of the y-axis labels but also the y-axis ticks and the y-axis tick labels. This is done by adding a call to the tick-underscore-params method. This method takes either y or x as its first argument, pointing to the fact that we are modifying the parameters of the y-axis ticks and tick labels. To change their color, we use the colors key-word argument, setting it to blue. Similarly, we call the tick-underscore-params method from the twin Axes object, setting the colors for these ticks to red.

Coloring the ticks



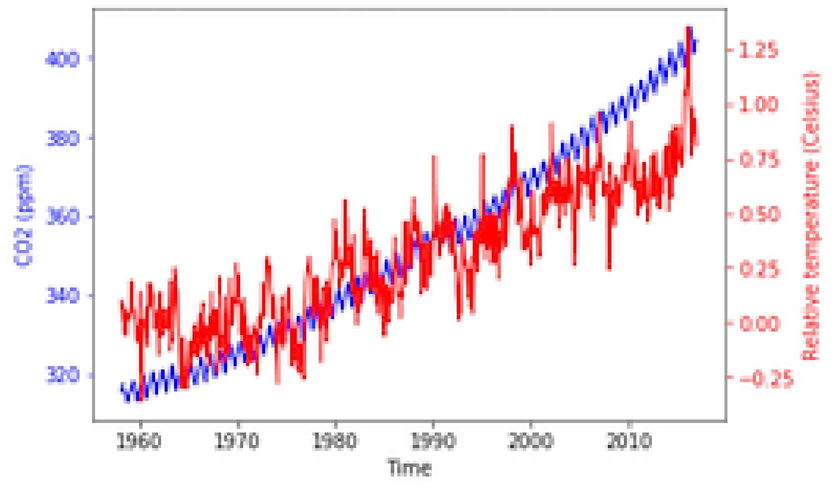
A function that plots time-series

```
def plot_timeseries(axes, x, y, color, xlabel, ylabel):
  axes.plot(x, y, color=color)
  axes.set_xlabel(xlabel)
  axes.set_ylabel(ylabel, color=color)
  axes.tick_params('y', colors=color)
```

Before we move on, let's implement this as a function that we can reuse.
8. A function that plots time-series
We use the def key-word to indicate that we are defining a function called plot-underscore-timeseries. This function takes as arguments an Axes object, x and y variables to plot, a color to associate with this variable, as well as x-axis and y-axis labels. The function calls the methods of the Axes object that we have seen before: plot, set-underscore-xlabel, set-underscore-ylabel, and tick-underscore-params.

Using our function

```
fig, ax = plt.subplots()
plot_timeseries(ax, climate_change.index, climate_change['co2'],
               'blue', 'Time', 'CO2 (ppm)')
ax2 = ax.twinx()
plot_timeseries(ax, climate_change.index,
                climate_change['relative_temp'],
               'red', 'Time', 'Relative temperature (Celsius)')
plt.show()
```



Create your own function!

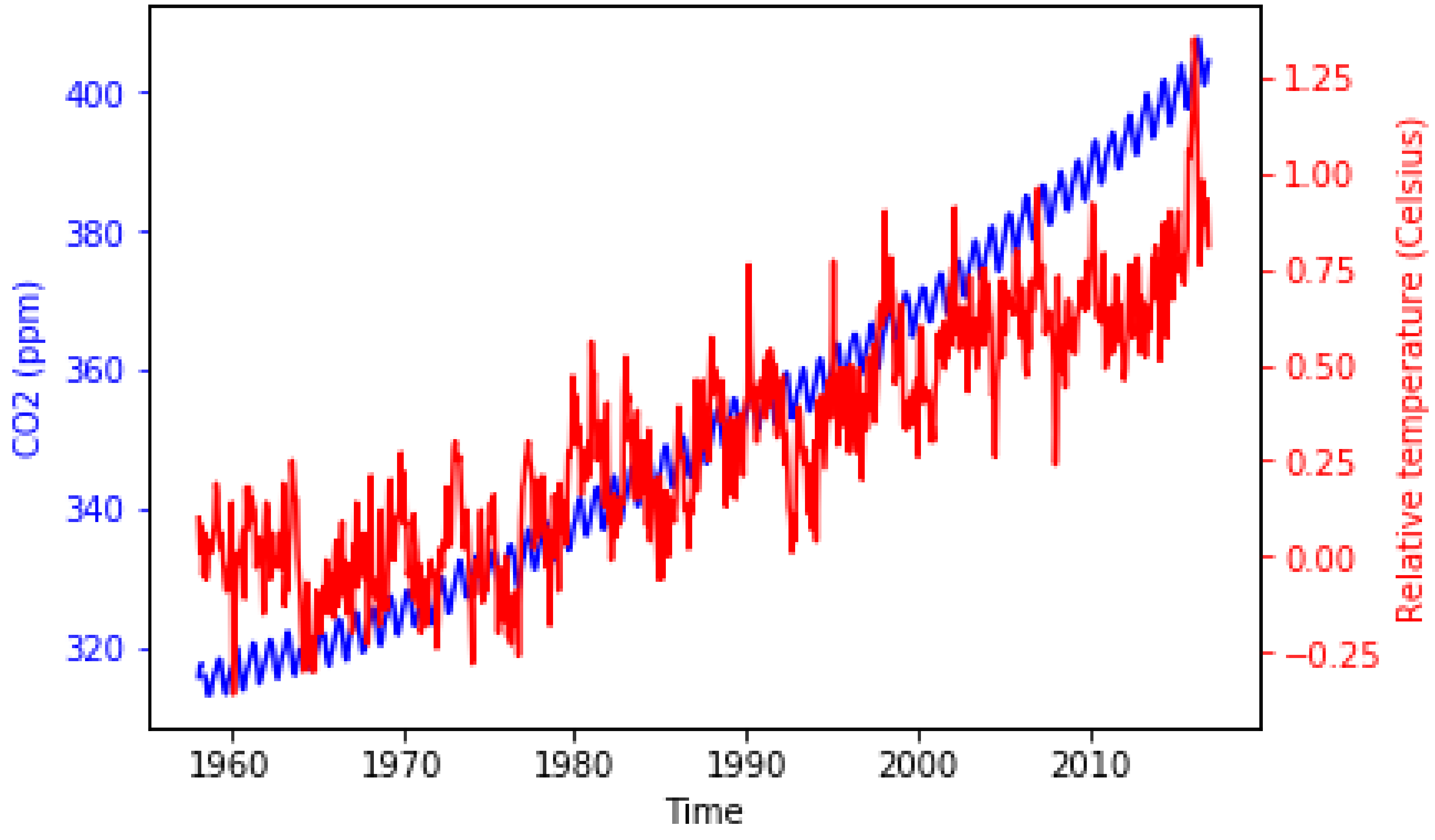
INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Annotating time-series data

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

Time-series data

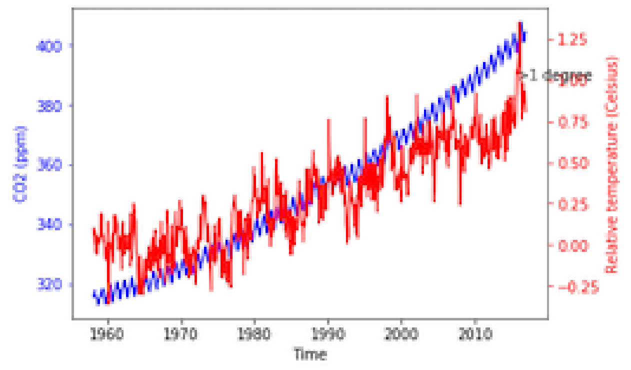


1. Annotating time-series data
One important way to enhance a visualization is to add annotations. Annotations are usually small pieces of text that refer to a particular part of the visualization, focusing our attention on some feature of the data and explaining this feature.

2. Time-series data
For example, consider the data that we saw in previous videos in this chapter. This data shows the levels of measured carbon dioxide in the atmosphere over a period of more than 50 years in blue and the relative temperature over the same period of time in red. That's a lot of data, and, when presenting it, you might want to focus attention on a particular aspect of this data.

Annotation

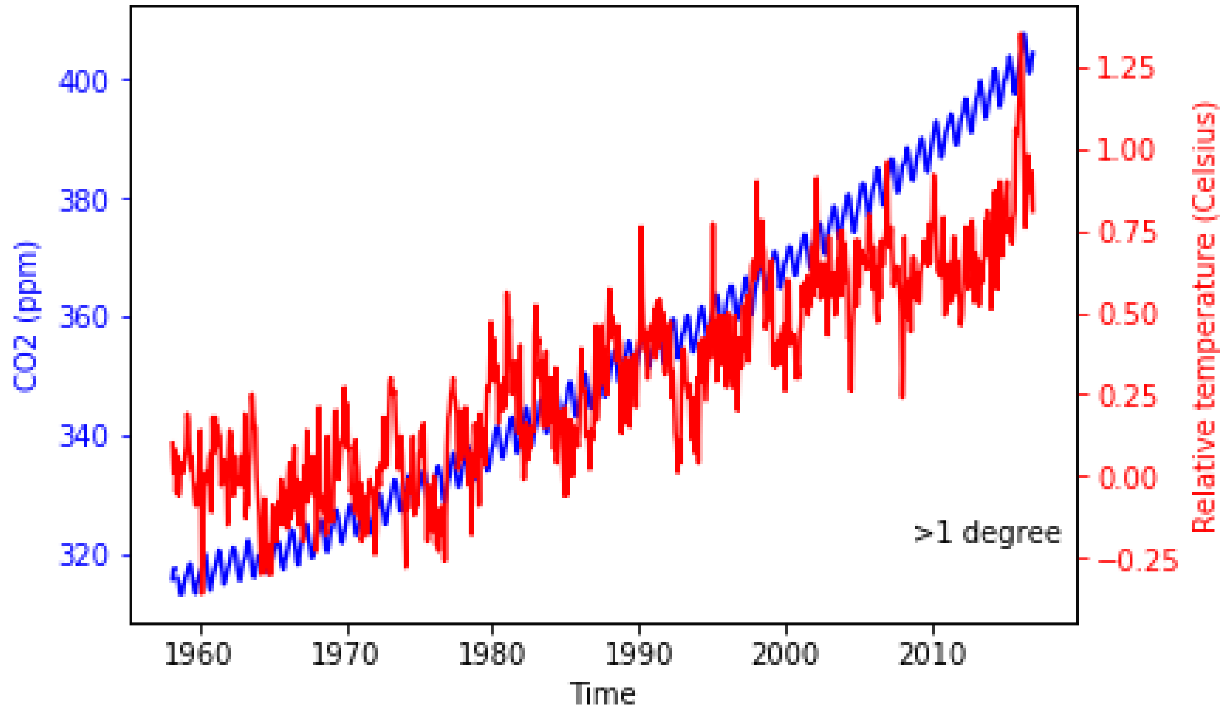
```
fig, ax = plt.subplots()
plot_timeseries(ax, climate_change.index, climate_change['co2'],
               'blue', 'Time', 'CO2 (ppm)')
ax2 = ax.twinx()
plot_timeseries(ax2, climate_change.index,
                climate_change['relative_temp'],
               'red', 'Time', 'Relative temperature (Celsius)')
ax2.annotate(">1 degree",  xy=[pd.TimeStamp("2015-10-06"), 1])
plt.show()
```



3. Annotation
One way to draw attention to part of a plot is by annotating it. This means drawing an arrow that points to part of the plot and being able to include text to explain it. For example, let's say that we noticed that the first date in which the relative temperature exceeded 1 degree Celsius was October 6th, 2015. We'd like to point this out in the plot. Here again is the code that generates the plot, using the function that we implemented previously. Next, we call a method of the Axes object called annotate. At the very least, this function takes the annotation text as input, in this case, the string ">1 degree", and the xy coordinate that we would like to annotate. Here, the value to annotate has the x position of the TimeStamp of that date. We use the Pandas time-stamp object to define that. The y position of the data is 1, which is the 1 degree Celsius value at that date. But this doesn't look great. The text appears on top of the axis tick labels. Maybe we can move it somewhere else?

Positioning the text

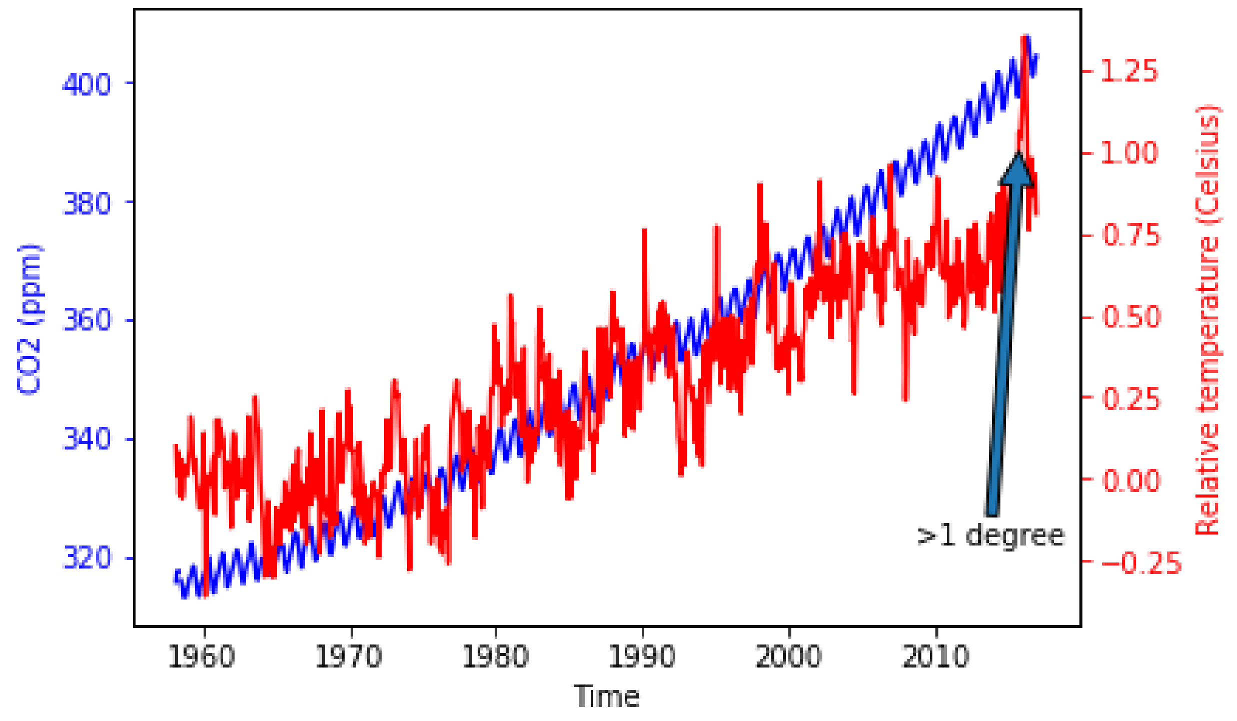
```
ax2.annotate(">1 degree",
             xy=(pd.Timestamp('2015-10-06'), 1),
             xytext=(pd.Timestamp('2008-10-06'), -0.2))
```



4. Positioning the text
The annotate method takes an optional xy text argument that selects the xy position of the text. After some experimentation, we've found that an x value of October 6th, 2008 and a y value of negative 0-point-2 degrees is a good place to put the text. The problem now is that there is no way to see which data point is the one that is being annotated. Let's add an arrow that connects the text to the data.

Adding arrows to annotation

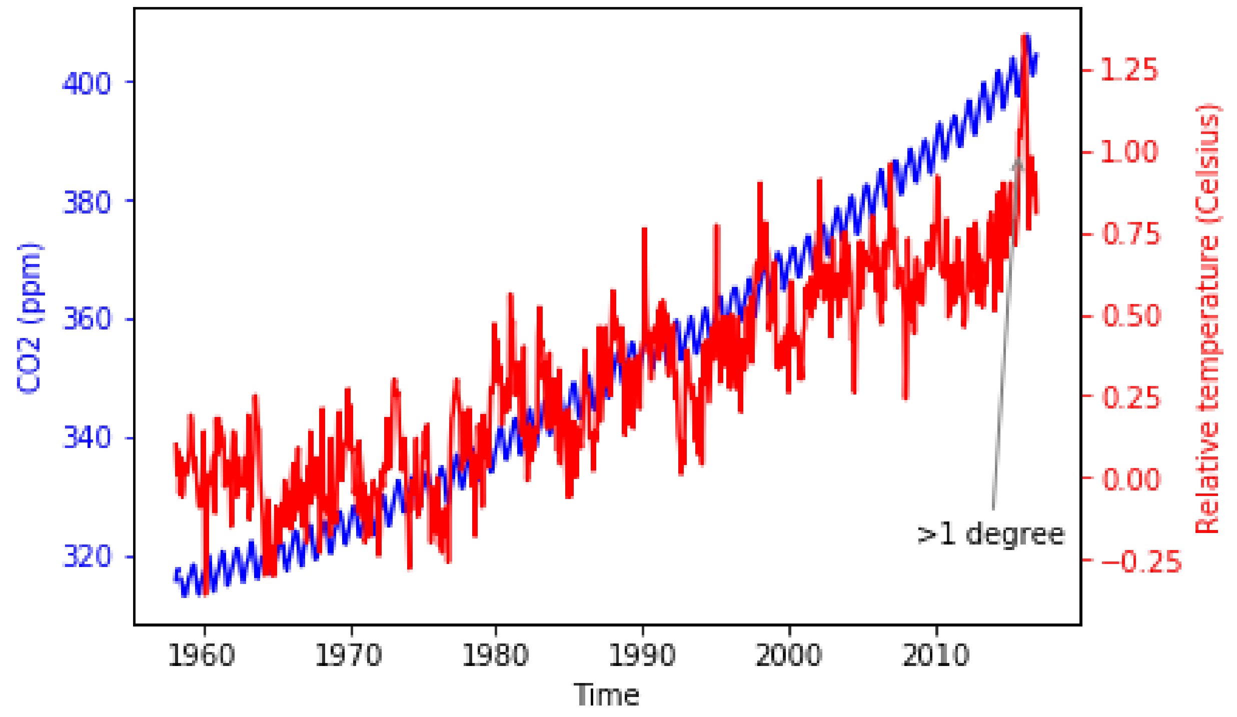
```
ax2.annotate(">1 degree",
             xy=(pd.Timestamp('2015-10-06'), 1),
             xytext=(pd.Timestamp('2008-10-06'), -0.2),
             arrowprops={})
```



5. Adding arrows to annotation
To connect between the annotation text and the annotated data, we can add an arrow. The key-word argument to do this is called arrowprops, which stands for arrow properties. This key-word argument takes as input a dictionary that defines the properties of the arrow that we would like to use. If we pass an empty dictionary into the key-word argument, the arrow will have the default properties, as shown here.

Customizing arrow properties

```
ax2.annotate(">1 degree",
             xy=(pd.Timestamp('2015-10-06'), 1),
             xytext=(pd.Timestamp('2008-10-06'), -0.2),
             arrowprops={"arrowstyle":"->", "color":"gray"})
```



6. Customizing arrow properties
We can also customize the appearance of the arrow. For example, here we set the style of the arrow to be a thin line with a wide head. That's what the string with a dash and a smaller than sign means. We also set the color to gray. This is a bit more subtle.

Customizing annotations

https://matplotlib.org/users/annotations.html

7. Customizing annotations
There are many more options for customizing the arrow properties and other properties of the annotation, which you can read about in the Matplotlib documentation here.

Practice annotating plots!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB