Saved



55 Points

Let’s dive right into working with two really important tools: Matplotlib and Seaborn! In this exercise, you will learn how to use Matplotlib, a powerful Python data visualization library. Matplotlib provides the building blocks to create rich visualizations of many different kinds of datasets. You will learn how to create visualizations for different kinds of data and how to customize, automate, and share these visualizations.

**Course Description**



Visualizing data in plots and figures exposes the underlying patterns in the data and provides insights. Good visualizations also help you communicate your data to others, and are useful to data analysts and other consumers of the data. In this course, you will learn how to use Matplotlib, a powerful Python data visualization library. Matplotlib provides the building blocks to create rich visualizations of many different kinds of datasets. You will learn how to create visualizations for different kinds of data and how to customize, automate, and share these visualizations.

**Daily XP0**

# Introduction to data visualization with Matplotlib

**50 XP**

## 1. Introduction to Data Visualization with Matplotlib

Hello and welcome to this course on data visualization with Matplotlib! A picture is worth a thousand words. Data visualizations let you derive insights from data and let you communicate about the data with others.

## 2. Data visualization

For example, this visualization shows an animated history of an outbreak of Ebola in West Africa. The amount of information in this complex visualization is simply staggering! This visualization was created using Matplotlib, a Python library that is widely used to visualize data. There are many software libraries that visualize data. One of the main advantages of Matplotlib is that it gives you complete control over the properties of your plot. This allows you to customize and control the precise properties of your visualizations. At the end of this course, you will know not only how to control your visualizations, but also how to create programs that automatically create visualizations based on your data.

## 3. Introducing the pyplot interface

There are many different ways to use Matplotlib. In this course, we will use the main object-oriented interface. This interface is provided through the pyplot submodule. Here, we import this submodule and name it plt. While using the name plt is not necessary for the program to work, this is a very strongly-followed convention, and we will follow it here as well. The plt-dot-subplots command, when called without any inputs, creates two different objects: a Figure object and an Axes object. The Figure object is a container that holds everything that you see on the page. Meanwhile, the Axes is the part of the page that holds the data. It is the canvas on which we will draw with our data, to visualize it. Here, you can see a Figure with empty Axes. No data has been added yet.

## 4. Adding data to axes

Let's add some data to our figure. Here is some data. This is a DataFrame that contains information about the weather in the city of Seattle in the different months of the year. The "MONTH" column contains the three-letter names of the months of the year. The "monthly average normal temperature" column contains the temperatures in these months, in Fahrenheit degrees, averaged over a ten-year period.

## 5. Adding data to axes

To add the data to the Axes, we call a plotting command. The plotting commands are methods of the Axes object. For example, here we call the method called plot with the month column as the first argument and the temperature column as the second argument. Finally, we call the plt-dot-show function to show the effect of the plotting command. This adds a line to the plot. The horizontal dimension of the plot represents the months according to their order and the height of the line at each month represents the average temperature. The trends in the data are now much clearer than they were just by reading off the temperatures from the table.

## 6. Adding more data

If you want, you can add more data to the plot. For example, we also have a table that stores data about the average temperatures in the city of Austin, Texas. We add these data to the axes by calling the plot method again.

## 7. Putting it all together

Here is what all of the code to create this figure would then look like. First, we create the Figure and the Axes objects. We call the Axes method plot to add first the Seattle temperatures, and then the Austin temperatures to the Axes. Finally, we ask Matplotlib to show us the figure.

## 8. Practice making a figure!

Now it's your turn. In the exercises, you will practice making a figure and axes and adding data into them.

# Using the matplotlib.pyplot interface

There are many ways to use Matplotlib. In this course, we will focus on the pyplot interface, which provides the most flexibility in creating and customizing data visualizations.

Initially, we will use the pyplot interface to create two kinds of objects: Figure objects and Axes objects.

This course introduces a lot of new concepts, so if you ever need a quick refresher, download the [*Matplotlib Cheat Sheet*](https://res.cloudinary.com/dyd911kmh/image/upload/v1676360378/Marketing/Blog/Matplotlib_Cheat_Sheet.pdf) and keep it handy!

##### Instructions

**100 XP**

* Import the matplotlib.pyplot API, using the conventional name plt.
* Create Figure and Axes objects using the plt.subplots function.
* Show the results, an empty set of axes, using the plt.show function.
* # Import the matplotlib.pyplot submodule and name it plt
* import \_\_\_\_ as \_\_\_\_
* # Create a Figure and an Axes with plt.subplots
* fig, ax = \_\_\_\_
* # Call the show function to show the result
* \_\_\_\_

# Import the matplotlib.pyplot submodule and name it plt

import matplotlib.pyplot as plt

# Create a Figure and an Axes with plt.subplots

fig, ax = plt.subplots()

# Call the show function to show the result

plt.show()

# Import the matplotlib.pyplot submodule and name it plt

import matplotlib.pyplot as plt

# Create a Figure and an Axes with plt.subplots

fig, ax = plt.subplots()

# Call the show function to show the result

plt.show

<function matplotlib.pyplot.show()>

# Import the matplotlib.pyplot submodule and name it plt

import matplotlib.pyplot as plt

# Create a Figure and an Axes with plt.subplots

fig, ax = plt.subplots()

# Call the show function to show the result

plt.show()

Nicely done! This script provides the basis for everything we'll do in this course.

# Adding data to an Axes object

Adding data to a figure is done by calling methods of the Axes object. In this exercise, we will use the plot method to add data about rainfall in two American cities: Seattle, WA and Austin, TX.

The data are stored in two pandas DataFrame objects that are already loaded into memory: seattle\_weather stores information about the weather in Seattle, and austin\_weather stores information about the weather in Austin. Each of the DataFrames has a "MONTH" column that stores the three-letter name of the months. Each also has a column named "MLY-PRCP-NORMAL" that stores the average rainfall in each month during a ten-year period.

In this exercise, you will create a visualization that will allow you to compare the rainfall in these two cities.

##### Instructions

**100 XP**

* Import the matplotlib.pyplot submodule as plt.
* Create a Figure and an Axes object by calling plt.subplots.
* Add data from the seattle\_weather DataFrame by calling the Axes plot method.
* Add data from the austin\_weather DataFrame in a similar manner and call plt.show to show the results.
* # Import the matplotlib.pyplot submodule and name it plt
* \_\_\_\_
* # Create a Figure and an Axes with plt.subplots
* fig, ax = \_\_\_\_
* # Plot MLY-PRCP-NORMAL from seattle\_weather against the MONTH
* ax.\_\_\_\_(seattle\_weather["MONTH"], \_\_\_\_)
* # Plot MLY-PRCP-NORMAL from austin\_weather against MONTH
* ax.\_\_\_\_(\_\_\_\_, \_\_\_\_)
* # Call the show function
* \_\_\_\_

Great! Next you will learn how to label the axes on this plot.

import matplotlib.pyplot as plt

# Create a Figure and an Axes with plt.subplots

fig, ax = plt.subplots()

# Plot MLY-PRCP-NORMAL from seattle\_weather against the MONTH

ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"])

# Plot MLY-PRCP-NORMAL from austin\_weather against MONTH

ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"])

# Call the show function

# Import the matplotlib.pyplot submodule and name it plt import matplotlib.pyplot as plt # Create a Figure and an Axes with plt.subplots fig, ax = plt.subplots() # Plot MLY-PRCP-NORMAL from seattle\_weather against the MONTH ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"]) # Plot MLY-PRCP-NORMAL from austin\_weather against MONTH ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"]) # Call the show function plt.show()

# Customizing data appearance

We can customize the appearance of data in our plots, while adding the data to the plot, using key-word arguments to the plot command.

In this exercise, you will customize the appearance of the markers, the linestyle that is used, and the color of the lines and markers for your data.

As before, the data is already provided in pandas DataFrame objects loaded into memory: seattle\_weather and austin\_weather. These each have a "MONTHS" column and a "MLY-PRCP-NORMAL" that you will plot against each other.

In addition, a Figure object named fig and an Axes object named ax have already been created for you.

##### Instructions

**100 XP**

* Call ax.plot to plot "MLY-PRCP-NORMAL" against "MONTHS" in both DataFrames.
* Pass the color key-word arguments to these commands to set the color of the Seattle data to blue ('b') and the Austin data to red ('r').
* Pass the marker key-word arguments to these commands to set the Seattle data to circle markers ('o') and the Austin markers to triangles pointing downwards ('v').
* Pass the linestyle key-word argument to use dashed lines for the data from both cities ('--').
* # Plot Seattle data, setting data appearance
* ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"], \_\_\_\_)
* # Plot Austin data, setting data appearance
* ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"], \_\_\_\_)
* # Call show to display the resulting plot
* plt.show()

# Plot Seattle data, setting data appearance ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"], marker='o', linestyle='--', color='b') # Plot Austin data, setting data appearance ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"], marker='v', linestyle='--', color='r') # Call show to display the resulting plot plt.show()

# Plot Seattle data, setting data appearance

ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"], marker='o', linestyle='--', color='b')

# Plot Austin data, setting data appearance

ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"], marker='v', linestyle='--', color='r')

# Call show to display the resulting plot

plt.show()

Beautiful! Now you can create plots that look exactly as you'd like them to appear.

# Customizing axis labels and adding titles

Customizing the axis labels requires using the set\_xlabel and set\_ylabel methods of the Axes object. Adding a title uses the set\_title method.

In this exercise, you will customize the content of the axis labels and add a title to a plot.

As before, the data is already provided in pandas DataFrame objects loaded into memory: seattle\_weather and austin\_weather. These each have a "MONTH" column and a "MLY-PRCP-NORMAL" column. These data are plotted against each other in the first two lines of the sample code provided.

In addition, a Figure object named fig and an Axes object named ax have already been created for you.

##### Instructions

**100 XP**

* Use the set\_xlabel method to add the label: "Time (months)".
* Use the set\_ylabel method to add the label: "Precipitation (inches)".
* Use the set\_title method to add the title: "Weather patterns in Austin and Seattle".
* ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"])
* ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"])
* # Customize the x-axis label
* \_\_\_\_
* # Customize the y-axis label
* \_\_\_\_
* # Add the title
* \_\_\_\_
* # Display the figure
* plt.show()

ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"]) ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"]) # Customize the x-axis label ax.set\_xlabel("Time (months)") # Customize the y-axis label ax.set\_ylabel("Precipitation (inches)") # Add the title ax.set\_title("Weather patterns in Austin and Seattle") # Display the figure plt.show()

ax.plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"])

ax.plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"])

# Customize the x-axis label

ax.set\_xlabel("Time (months)")

# Customize the y-axis label

ax.set\_ylabel("Precipitation (inches)")

# Add the title

ax.set\_title("Weather patterns in Austin and Seattle")

# Display the figure

plt.show()

Great! We'll learn more about annotating figures later in the course.

**Daily XP500**

# Small multiples

**50 XP**

## 1. Small multiples

In some cases, adding more data to a plot can make the plot too busy, obscuring patterns rather than revealing them.

## 2. Adding data

For example, let's explore the data we have about weather in Seattle. Here we plot average precipitation in Seattle during the course of the year. But let's say that we are also interested in the range of values.

## 3. Adding more data

We add the 25th percentile and the 75th percentile of the precipitation in dashed lines above and below the average. What would happen if we compared this to Austin?

## 4. And more data

This code adds the data from Austin to the plot. When we display the plot,

## 5. Too much data!

it's a bit of a mess. There's too much data in this plot. One way to overcome this kind of mess is to use what are called small multiples. These are multiple small plots that show similar data across different conditions. For example, precipitation data across different cities.

## 6. Small multiples with plt.subplots

In Matplotlib, small multiples are called sub-plots. That is also the reason that the function that creates these is called subplots. Previously, we called this function with no inputs. This creates one subplot. Now, we'll give it some inputs. Small multiples are typically arranged on the page as a grid with rows and columns. Here, we are creating a Figure object with three rows of subplots, and two columns. This is what this would look like before we add any data to it. In this case, the variable ax is no longer only one Axes object.

## 7. Adding data to subplots

Instead, it is an array of Axes objects with a shape of 3 by 2. To add data, we would now have to index into this object and call the plot method on an element of the array.

## 8. Subplots with data

There is a special case for situations where you have only one row or only one column of plots. In this case, the resulting array will be one-dimensional and you will only have to provide one index to access the elements of this array. For example, consider what we might do with the rainfall data that we were plotting before. We create a figure and an array of Axes objects with two rows and one column. We address the first element in this array, which is the top sub-plot, and add the data for Seattle to this plot. Then, we address the second element in the array, which is the bottom plot, and add the data from Austin to it. We can add a y-axis label to each one of these. Because they are one on top of the other, we only add an x-axis label to the bottom plot, by addressing only the second element in the array of Axes objects. When we show this,

## 9. Subplots with data

we see that the data are now cleanly presented in a way that facilitates the direct comparison between the two cities. One thing we still need to take care of is the range of the y-axis in the two plots, which is not exactly the same. This is because the highest and lowest values in the two datasets are not identical.

## 10. Sharing the y-axis range

To make sure that all the subplots have the same range of y-axis values, we initialize the figure and its subplots with the key-word argument sharey set to True. This means that both subplots will have the same range of y-axis values, based on the data from both datasets. Now the comparison across datasets is more straightforward.

## 11. Practice making subplots!

Next, go ahead and practice creating visualizations with small multiples.

# Creating a grid of subplots

How would you create a Figure with 6 Axes objects organized in 3 rows and 2 columns?

##### Possible Answers

* fig, ax = plt.subplots((3, 2))
* fig, ax = plt.axes((2, 3))
* **fig, ax = plt.subplots(3, 2)**
* fig, ax = plt.subplots((2, 3))
* That's right! This gives you the output you need.

# Creating small multiples with plt.subplots

Small multiples are used to plot several datasets side-by-side. In Matplotlib, small multiples can be created using the plt.subplots() function. The first argument is the number of rows in the array of Axes objects generate and the second argument is the number of columns. In this exercise, you will use the Austin and Seattle data to practice creating and populating an array of subplots.

The data is given to you in DataFrames: seattle\_weather and austin\_weather. These each have a "MONTH" column and "MLY-PRCP-NORMAL" (for average precipitation), as well as "MLY-TAVG-NORMAL" (for average temperature) columns. In this exercise, you will plot in a separate subplot the monthly average precipitation and average temperatures in each city.

* Create a Figure and an array of subplots with 2 rows and 2 columns.
* Addressing the top left Axes as index 0, 0, plot the Seattle precipitation.
* In the top right (index 0,1), plot Seattle temperatures.
* In the bottom left (1, 0) and bottom right (1, 1) plot Austin precipitations and temperatures.
* # Create a Figure and an array of subplots with 2 rows and 2 columns
* fig, ax = plt.subplots(\_\_\_\_, \_\_\_\_)
* # Addressing the top left Axes as index 0, 0, plot month and Seattle precipitation
* ax[0, 0].plot(\_\_\_\_, \_\_\_\_)
* # In the top right (index 0,1), plot month and Seattle temperatures
* ax[0, 1].plot(\_\_\_\_, \_\_\_\_)
* # In the bottom left (1, 0) plot month and Austin precipitations
* ax[\_\_\_\_].plot(\_\_\_\_, \_\_\_\_)
* # In the bottom right (1, 1) plot month and Austin temperatures
* ax[\_\_\_\_].plot(\_\_\_\_, \_\_\_\_)
* plt.show()

That's great! Next, you will put together all the things you've learned so far.

# Create a Figure and an array of subplots with 2 rows and 2 columns fig, ax = plt.subplots(2, 2) # Addressing the top left Axes as index 0, 0, plot month and Seattle precipitation ax[0, 0].plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"]) # In the top right (index 0,1), plot month and Seattle temperatures ax[0, 1].plot(seattle\_weather["MONTH"], seattle\_weather["MLY-TAVG-NORMAL"]) # In the bottom left (1, 0) plot month and Austin precipitations ax[1,0].plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"]) # In the bottom right (1, 1) plot month and Austin temperatures ax[1,1].plot(austin\_weather["MONTH"], austin\_weather["MLY-TAVG-NORMAL"]) plt.show()

# Create a Figure and an array of subplots with 2 rows and 2 columns

fig, ax = plt.subplots(2, 2)

# Addressing the top left Axes as index 0, 0, plot month and Seattle precipitation

ax[0, 0].plot(seattle\_weather["MONTH"], seattle\_weather["MLY-PRCP-NORMAL"])

# In the top right (index 0,1), plot month and Seattle temperatures

ax[0, 1].plot(seattle\_weather["MONTH"], seattle\_weather["MLY-TAVG-NORMAL"])

# In the bottom left (1, 0) plot month and Austin precipitations

ax[1,0].plot(austin\_weather["MONTH"], austin\_weather["MLY-PRCP-NORMAL"])

# In the bottom right (1, 1) plot month and Austin temperatures

ax[1,1].plot(austin\_weather["MONTH"], austin\_weather["MLY-TAVG-NORMAL"])

plt.show()

# Small multiples with shared y axis

When creating small multiples, it is often preferable to make sure that the different plots are displayed with the same scale used on the y-axis. This can be configured by setting the sharey key-word to True.

In this exercise, you will create a Figure with two Axes objects that share their y-axis. As before, the data is provided in seattle\_weather and austin\_weather DataFrames.

##### Instructions

**100 XP**

* Create a Figure with an array of two Axes objects that share their y-axis range.
* Plot Seattle's "MLY-PRCP-NORMAL" in a solid blue line in the top Axes.
* Add Seattle's "MLY-PRCP-25PCTL" and "MLY-PRCP-75PCTL" in dashed blue lines to the top Axes.
* Plot Austin's "MLY-PRCP-NORMAL" in a solid red line in the bottom Axes and the "MLY-PRCP-25PCTL" and "MLY-PRCP-75PCTL" in dashed red lines.
* # Create a figure and an array of axes: 2 rows, 1 column with shared y axis
* fig, ax = plt.subplots(2, 1, sharey=True)
* # Plot Seattle precipitation data in the top axes
* \_\_\_\_.plot(\_\_\_\_, \_\_\_\_, color = \_\_\_\_)
* \_\_\_\_.plot(\_\_\_\_, \_\_\_\_, color = \_\_\_\_, linestyle = \_\_\_\_)
* \_\_\_\_.plot(\_\_\_\_, \_\_\_\_, color = \_\_\_\_, linestyle = \_\_\_\_)
* # Plot Austin precipitation data in the bottom axes
* \_\_\_\_.plot(\_\_\_\_, \_\_\_\_, color = \_\_\_\_)
* \_\_\_\_.plot(\_\_\_\_, \_\_\_\_, color = \_\_\_\_, linestyle = \_\_\_\_)
* \_\_\_\_.plot(\_\_\_\_, \_\_\_\_, color = \_\_\_\_, linestyle = \_\_\_\_)
* plt.show()

Good job! Next, you will learn how to plot more complex time-series data.

# Create a figure and an array of axes: 2 rows, 1 column with shared y axis

fig, ax = plt.subplots(2, 1, sharey=True)

# Plot Seattle precipitation data in the top axes

ax[0].plot(seattle\_weather['MLY-PRCP-NORMAL'], color = 'b')

ax[0].plot(seattle\_weather['MLY-PRCP-NORMAL'], seattle\_weather['MLY-PRCP-25PCTL'], color = 'b', linestyle = '--')

ax[0].plot(seattle\_weather['MLY-PRCP-NORMAL'],seattle\_weather['MLY-PRCP-75PCTL'] , color = 'b', linestyle = '--')

# Plot Austin precipitation data in the bottom axes

ax[1].plot(austin\_weather['MLY-PRCP-NORMAL'], color = 'r')

ax[1].plot(austin\_weather['MLY-PRCP-NORMAL'], austin\_weather['MLY-PRCP-25PCTL'], color = 'r', linestyle = '--')

ax[1].plot(austin\_weather['MLY-PRCP-NORMAL'], austin\_weather['MLY-PRCP-75PCTL'], color = 'r', linestyle = '--')

plt.show()

# Create a figure and an array of axes: 2 rows, 1 column with shared y axis fig, ax = plt.subplots(2, 1, sharey=True) # Plot Seattle precipitation data in the top axes ax[0].plot(seattle\_weather['MLY-PRCP-NORMAL'], color = 'b') ax[0].plot(seattle\_weather['MLY-PRCP-NORMAL'], seattle\_weather['MLY-PRCP-25PCTL'], color = 'b', linestyle = '--') ax[0].plot(seattle\_weather['MLY-PRCP-NORMAL'],seattle\_weather['MLY-PRCP-75PCTL'] , color = 'b', linestyle = '--') # Plot Austin precipitation data in the bottom axes ax[1].plot(austin\_weather['MLY-PRCP-NORMAL'], color = 'r') ax[1].plot(austin\_weather['MLY-PRCP-NORMAL'], austin\_weather['MLY-PRCP-25PCTL'], color = 'r', linestyle = '--') ax[1].plot(austin\_weather['MLY-PRCP-NORMAL'], austin\_weather['MLY-PRCP-75PCTL'], color = 'r', linestyle = '--') plt.show()