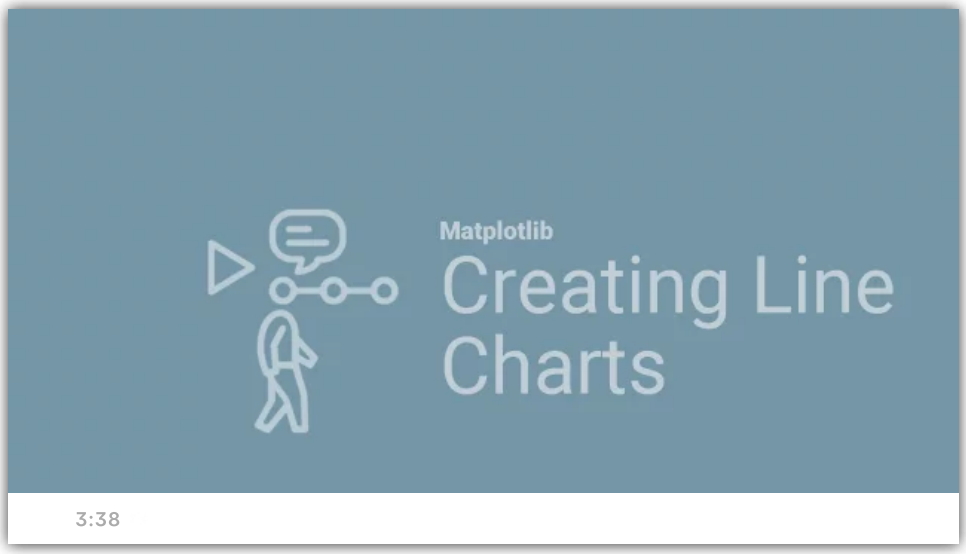
5.1.3 Create Line Charts

With Matplotlib set up and ready to go, it's time to explore what this program can do! Omar has suggested that you start with one of the most common charts: a line chart.

Omar has also reminded you to comment on your code. This way, if he or the CEO wants to take a look at your repo, they'll know exactly what you're doing and whether you're heading in the right direction.



The line chart is one of the most common charts used to graph data. This type of chart is pretty straightforward: it plots continuous data as points and joins them with a line. Line charts allow you to compare multiple datasets on the same chart rather easily.

Different visualization types serve different purposes. The purpose of a line chart is to display data over time. In this way, a line chart is useful for determining the relationship between time and another value. In contrast, a pie chart, which we'll learn to create later, is not used to show changes over time but to compare parts of a whole.

We'll walk through how to create line charts with Matplotlib using both the MATLAB method and the object-oriented interface method. We'll follow this general approach for every chart we create in this module.

First, launch Jupyter Notebook in the PythonData environment inside the PyBer_Analysis folder.

Next, create a new Jupyter Notebook file and name it `matplotlib_practice`.

Create a Line Chart Using the MATLAB Method

To make sure our plots are displayed inline within the Jupyter Notebook directly below the code cell that produced it, we need to add `%matplotlib inline` to the first cell, as shown here:

```
%matplotlib inline
```

The % sign is sometimes called the **magic command**. When you run a cell that starts with `%`, "magic" happens behind the scenes and sets the back-end processor used for charts. Using it with the `inline` value displays the chart within the cell for a Jupyter Notebook file.

NOTE

> Other values for the magic command are:
>
> - 'GTK3Agg'
> - 'GTK3Cairo'
> - 'MacOSX'
> - 'nbAgg'
> - 'Qt4Agg'
> - 'Qt4Cairo'
> - 'Qt5Agg'

- 'Qt5Cairo'
- 'TkAgg'
- 'TkCairo'
- 'WebAgg'
- 'WX'
- 'WXAgg'
- 'WXCairo'
- 'Agg'
- 'Cairo'
- 'Pdf'
- 'Pgf'
- 'Ps'
- 'Svg'
- 'template'

For more experienced users, if you're embedding Matplotlib in a web or other application, you might need to use a back-end processor, such as GTK, Qt, Tk, or Wx, that matches the toolkit you're using to build your application.

For more information, see the **Matplotlib documentation on back ends (https://matplotlib.org/3.1.1/api/matplotlib_configuration_api.html)**.

In the next cell, we'll import the `matplotlib` dependency. Then we'll chain the `pyplot` module to it using `matplotlib.pyplot`. This is the same as typing `from matplotlib import pyplot`. Finally, we'll rename `matplotlib.pyplot` as the alias `plt`.

To do this, type the following into the next cell:

```
# Import dependencies.
import matplotlib.pyplot as plt
```

The `pyplot` module will allow us to create all the necessary charts; create a plotting area in a figure; plot lines in a plotting area; and annotate the chart with labels, colors, and other decorative features.

NOTE

> Pyplot provides a MATLAB-like interface for making plots.

In the next cell, we'll add some ride-sharing data so we have something to graph. The data we'll use at first is just a small portion of the data we'll work with later.

The data is in the form of arrays, or lists. One array will contain the months of the year, and the second will contain the total fares in US dollars for each month.

Add the following code to the new cell and run it:

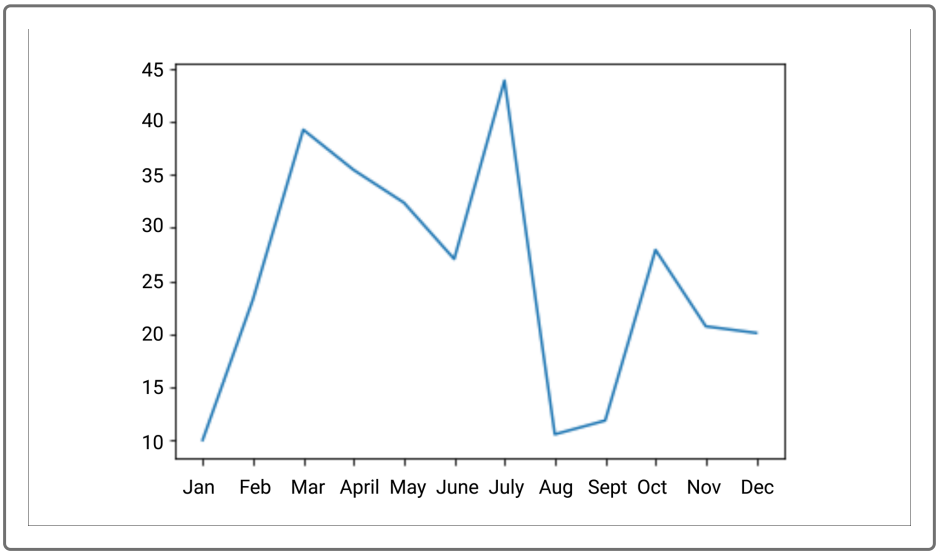
```
# Set the x-axis to a list of strings for each month.
x_axis = ["Jan", "Feb", "Mar", "April", "May", "June", "July", "Aug", "Sept"

# Set the y-axis to a list of floats as the total fare in US dollars accumul
y_axis = [10.02, 23.24, 39.20, 35.42, 32.34, 27.04, 43.82, 10.56, 11.85, 27.
```

In the next cell, chain the `plot()` function to `plt`. Inside the parentheses of the `plot()` function, add the `x_axis` and `y_axis` parameters:

```
# Create the plot
plt.plot(x_axis, y_axis)
```

The purpose of `plt.plot()` is to create the figure with the current axes. When we execute this cell, the first variable, `x_axis`, is the data for the x-axis, and the second variable, `y_axis`, is the data for the y-axis. We have labeled our arrays to make it more convenient for this and other examples. The output in the next cell is a figure with the months on the x-axis and the fare in U.S. dollars on the y-axis, as shown here:



Create a Line Chart Using the Object-Oriented Interface

When we switch from using MATLAB to the object-oriented interface method, we use `ax.plot()` instead of `plt.plot()`. Let's create a simple line graph so we can see the differences between the approaches.

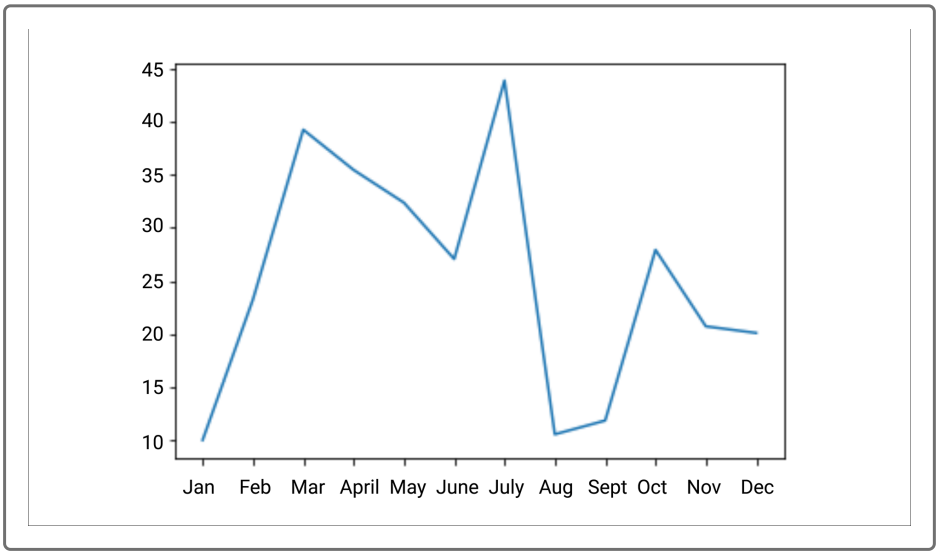
Before we dive in, here's a high-level view of how to create a figure and axes using the object-oriented method:

1. To create a figure, use `fig, ax = plt.subplots()`.
2. To define the axes, use `ax.plot()`.
3. Inside the parentheses of `ax.plot()`, add the axes separated by a comma.

Let's recreate the line chart we just made using the object-oriented interface method. In a new cell, add the following code:

```
# Create the plot with ax.plt()
fig, ax = plt.subplots()
ax.plot(x_axis, y_axis)
```

As you can see in the following chart, when we run this cell, the output is the same as we got when we used the MATLAB method:



Let's break down what the code is doing:

1. The `plt.subplots()` function returns a tuple that contains a figure and axes object(s).
2. `fig, ax = plt.subplots()` unpacks the tuple into the variables `fig` and `ax`.
3. The first variable, `fig`, references the figure.
4. The second variable, `ax`, references the axes of the figure and allows us to annotate the figure.
5. The `ax.plot(x_axis, y_axis)` plots the data from the x and y axes.

The `fig, ax = plt.subplots()` line of code can also be written like this.

```
# Create the plot with ax.plt()
fig = plt.figure()
ax = fig.add_subplot()
```

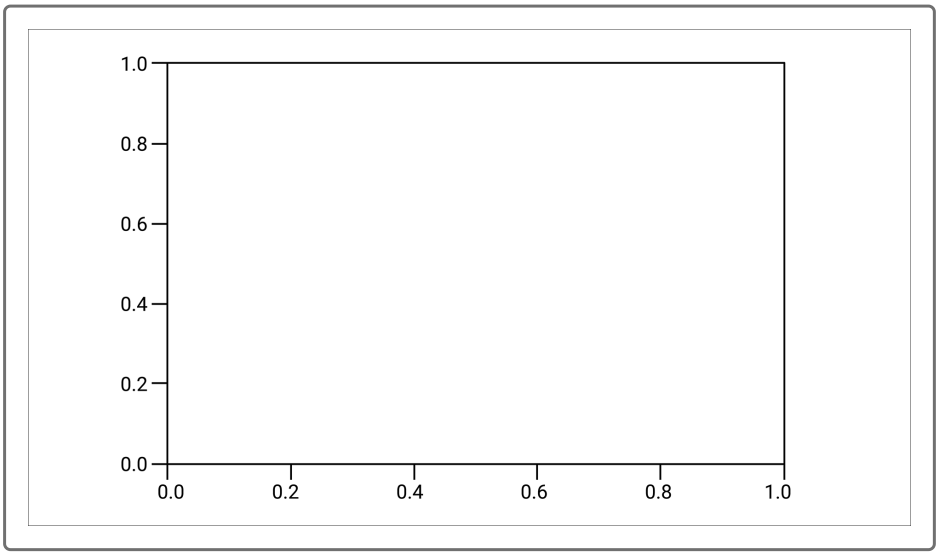
Where `fig` references the figure, and `ax` references the axes of the figure and allows us to annotate the figure.

In a new cell, replace `fig, ax = plt.subplots()` with `fig = plt.figure()` and `ax = fig.add_subplot()`.

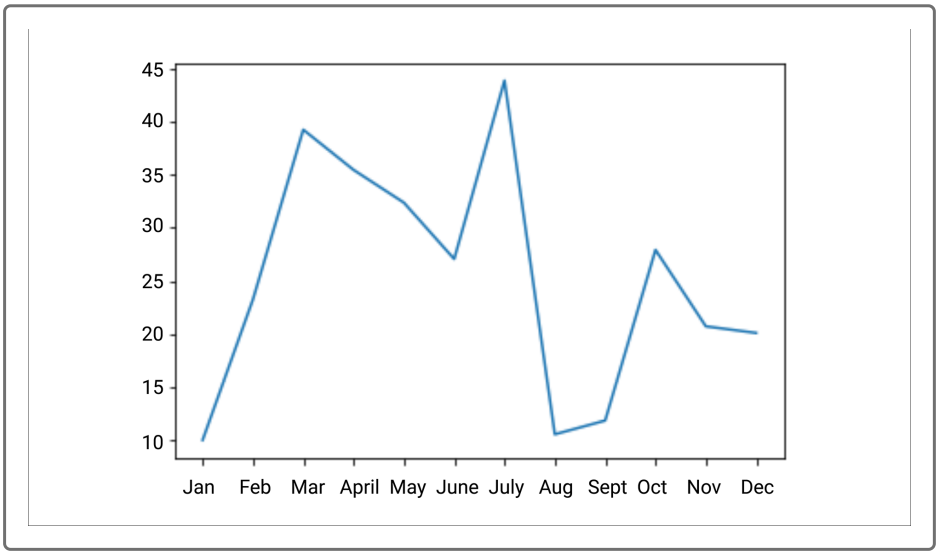
Your cell should look like this:

```
# Create the plot with ax.plt()
fig = plt.figure()
ax = fig.add_subplot()
```

When we execute this cell, we'll see that the result is an empty chart, as shown here:



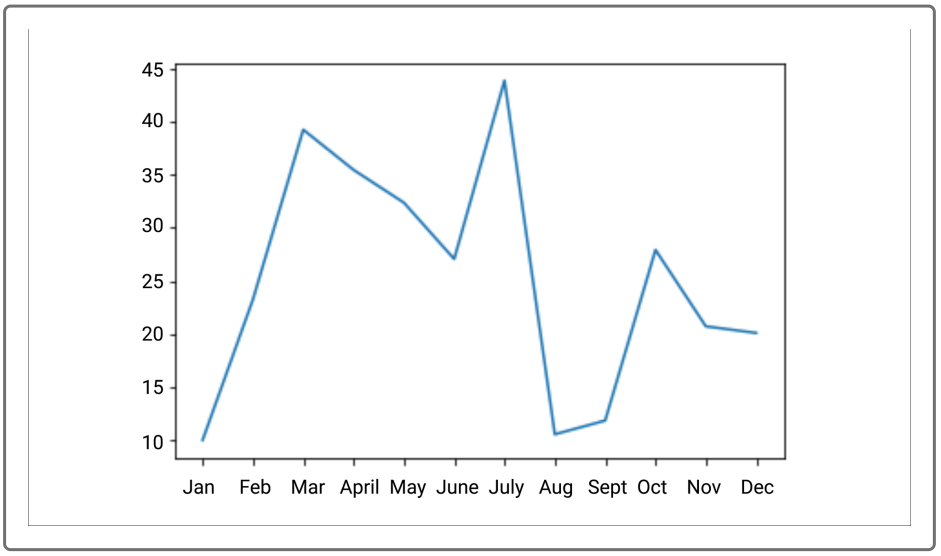
In the same cell, add `ax.plot(x_axis, y_axis)` to the cell and run the cell. The result is the same as using the MATLAB method, as you can see below:



To create a quick graph using the object-oriented interface method, use the following two lines:

```
# Create the plot with ax.plt()
ax = plt.axes()
ax.plot(x_axis, y_axis)
```

When we execute the cell, the result is the same as before, as you can see here:



IMPORTANT

> To change figure-level attributes (such as axis labels, a title, or a legend) or save the figure as an image, use `fig = plt.figure()`.

Use plt.show()

If you search for examples of Matplotlib graphs on the internet, you'll typically find examples that have `plt.show()` at the end of the graphing script.

The `plt.show()` function looks for all the active figure objects and opens a window that displays them if you have two or more sets of data to plot.

This is how we use the `plt.show()` function:

```
# Create the plot.
plt.plot(x_axis, y_axis)
plt.show()
```

IMPORTANT

> `plt.show()` should only be used once per session because it can cause the graph to display improperly. If you have problems with data not being displayed properly, you may have to restart your kernel and clear your output.

REWIND

To restart the kernel and clear the output, you will have to select "Restart & Clear Output" in the Kernel tab on Jupyter Notebook.