**Data visualizations can be used with matplotlib.**

You will use matplotlib and Jupiter notebook and python script to compose analysis using visuals.

The plt.plot() function is composed of several elements. plt is used to pull all methods associated with the matplotlib import when done as plt. The plot method executes the command to plot values on a graph; the .show method plots the graph as a visual representation. **When no values are inputted, matplot will automatically code in the missing x axis values.**

**Placement matters in the plot method. The first value in the method is used to plot the x axis, the second value is for the y axis.**

Plot.xlabel() – this will inject a string value to label the chart data.

Plt.ylabel() – this is the label being assigned to the y axis of a chart.

**Create a line chart using the matlab methods\***

Add the %matplotlib inline code to the start of all matlab codes in junypter notebook. **The % sign is a magic command; when you run a cell with the % sign in front, it affects the back end processing procedure for the charts in matlabs. Using is in this way causes the values and chart representations to appear ‘inline’ with data displays contained in jupyer notebook.**

%matplotlib inline

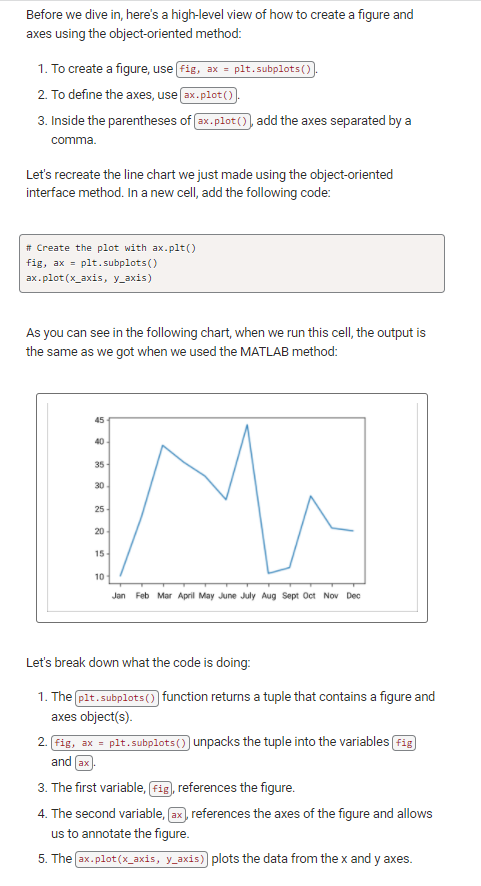
<https://matplotlib.org/3.1.1/api/matplotlib_configuration_api.html> - see more here

when you siwthc from using matlab ot the object oriented interface methods use the ax.plot() instead of plt.plot().

**Review the ax.plot method and why it is preferred to the plt.plot method** – section 5.1.3

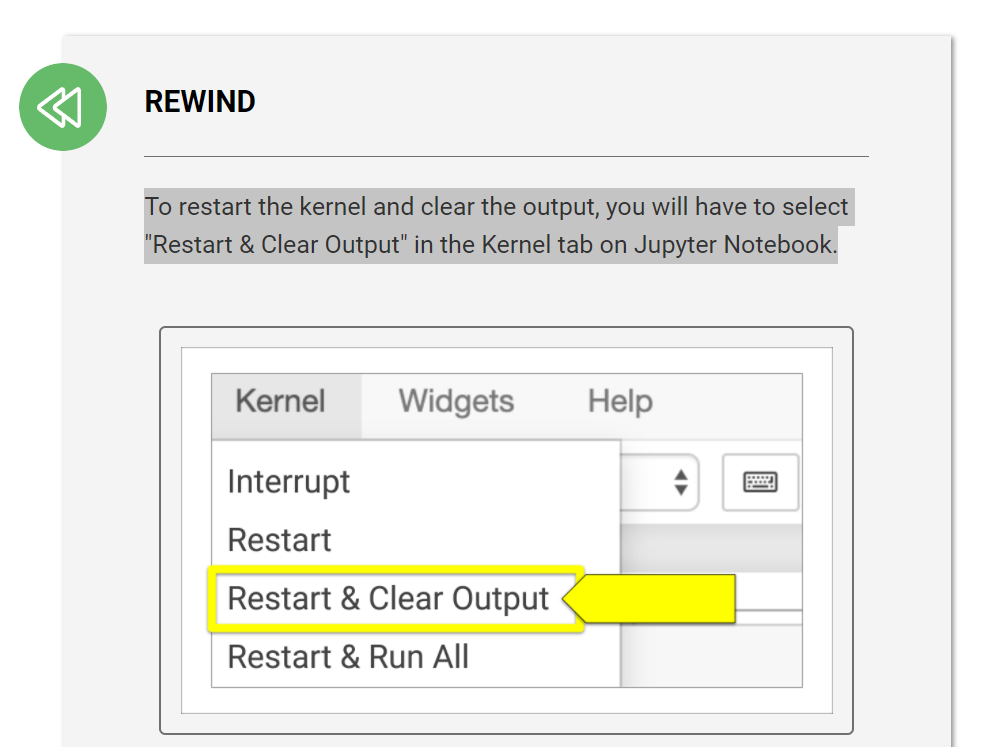
**Create a Line Chart Using the Object-Oriented Interface**

When we switch fomr using MATLABt to OOI (object oriented intereface methods); we use ax.plot() instead of plt.plot()



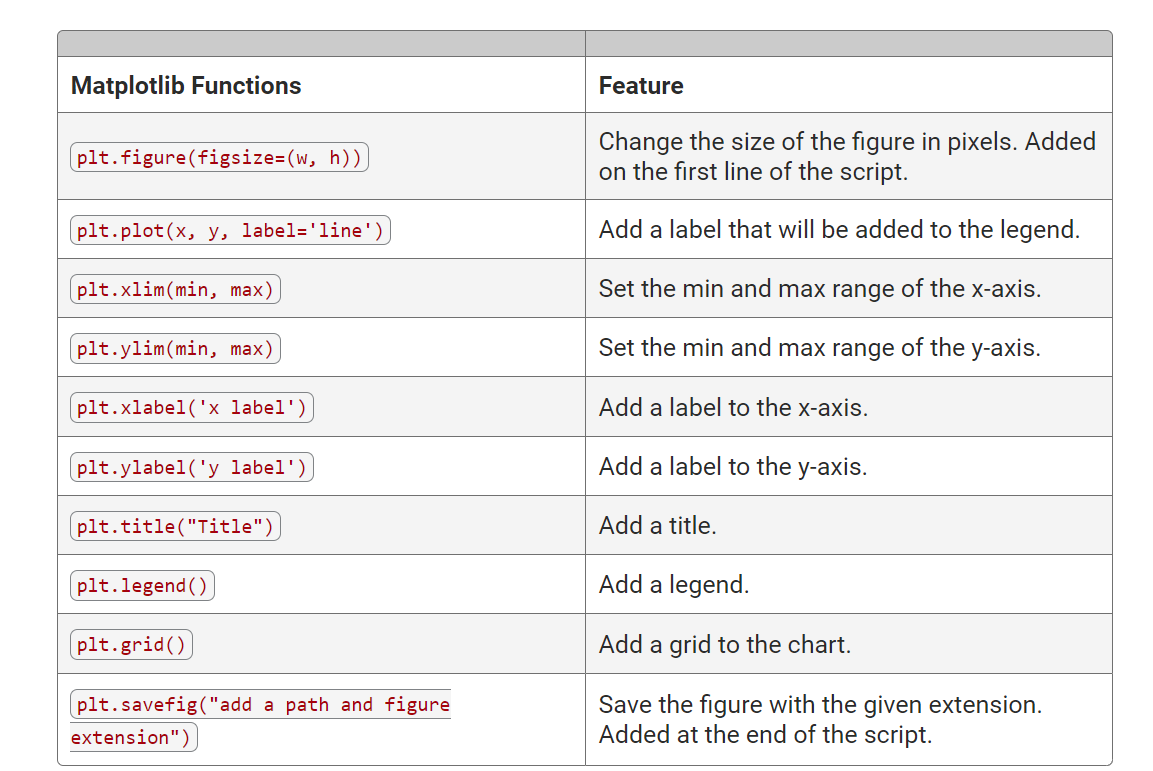
To change figure-level attributes (such as axis labels, a title, or a legend) or save the figure as an image, use fig = plt.figure().

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.



5.1.4 – annotating charts

**Annotating charts using MATLAB methods**



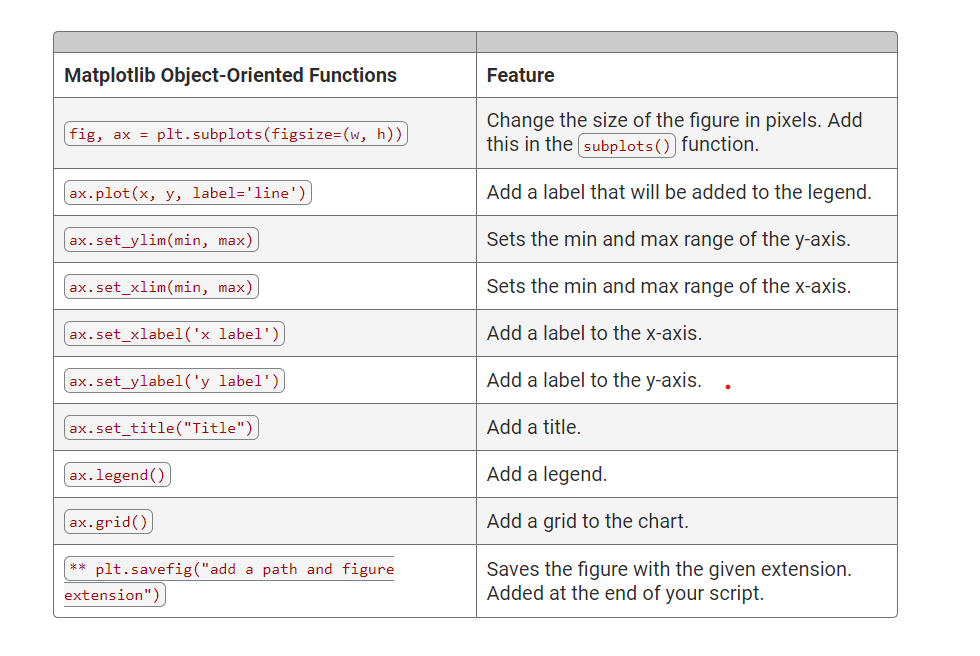
\*note: graph values have index values such that when the y or x axis is to be modified, you can call upon their values with index numbers.

https://matplotlib.org/stable/api/\_as\_gen/matplotlib.pyplot.plot.html

<https://matplotlib.org/stable/api/index.html>

<https://matplotlib.org/stable/tutorials/introductory/usage.html>

## Annotate Charts Using the Object-Oriented Interface

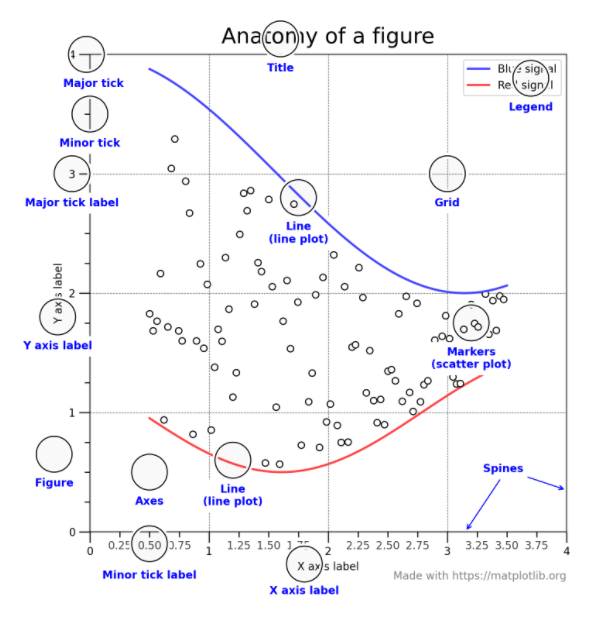


**Note from matplotlib documentations:**

Matplotlib graphs ddata on figures (jupyter wiedges, windows, etc) each of whcin contain one or more axes, a point that is specified in terms of x-y coordinates.

[fig](https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure), ax = [plt.subplots](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html#matplotlib.pyplot.subplots)() *# Create a figure containing a single axes.*

[ax.plot](https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html#matplotlib.axes.Axes.plot)([1, 2, 3, 4], [1, 4, 2, 3]); *# Plot some data on the axes.*



Create a figure by declaring it; fig = plt.figure() # an empty figure with no Axes

**The whole figure is the ‘fig’; the figure contains all other elements (children), special artists (the name given to chart/figure objects such as figure legends, titles, bars, etc) and even nested subfigures.**

* Axes: an artist attached to a fig containing plotted data. That typically includes multiple axis objects that provided tickes and tick labels. Axes classes have member functions (IE: ax.plot())
* Axis – objects seting the scale and limits and generate ticks and ticklabels.
* Artist – everything visible in a figure including text, lines, elements, and visual items.

You can use plot.xxx commands or ax.xx commands to execute the creation of a plot in matlibs.

**5.1.5 – create a bar chart using the matlab approach**

<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html>

<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.barh.html#matplotlib.pyplot.barh>

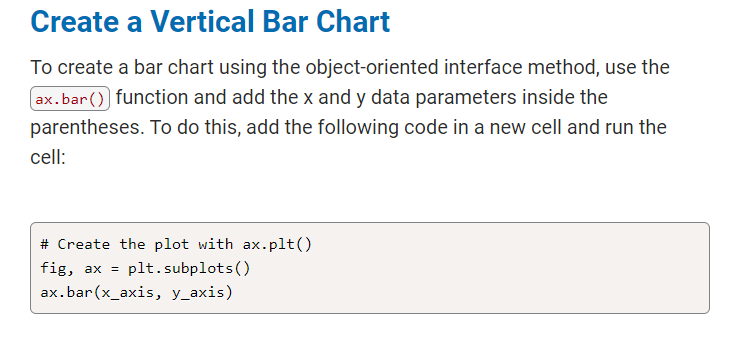
To create a horizontal chart, we use the plt.barh() function:

To create a vertical bar chart, use the plt.bar() function. In the matplotlib\_practice.ipynb file, add the following x and y data arguments inside parentheses:

We should invert the y-axis of the previous chart to have "January" at the top and "December" at the bottom. Can you think of the reason for this? Right—because we don't want to show the CEO ride-sharing data with the months in the wrong order!

To invert the y-axis to have the months in ascending order, use the gca() method. The gca() method means "get current axes." We can chain the gca() method to the invert\_yaxis() method by using gca().invert\_yaxis(), as shown here:

**5.1.6 – creating bar charts using object oriented approach**



\*\*plt.gca().invert\_yaxis() command works on object oriented and matlib method types\*.

<https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.bar.html>

**5.1.7 – creating scatter plots and bubble charts\*\*.**

You can generate scatter plots and create lines of best fit using matploblib libraries.

**NOTE: remember to refresh the kernel in the outputs; sometimes the software ‘stops up’ and it must be reopened fresh.**

If we were only to use the plt.plot() function with our x-axis and y-axis, it would create a line plot and that's not what we're looking for. So how do we make it a scatter plot?

When we use the plt.plot() function to create a scatter plot, we need to add a lowercase "o" as a parameter inside the parentheses. This switches the plot from a line chart to a scatter plot.

plt.plot(x\_axis, y\_axis, 'o') ; below is an alternative method:

plt.scatter(x\_axis, y\_axis)

## Create a Bubble Chart Using the MATLAB Method

The first two factors are the x- and y-axes data, which we have been using quite frequently. By changing the "dot" into a "bubble," we are adding a third factor: size. If there is more than one dataset that uses the same axes, we can change the color of each marker for each dataset, which will add a fourth factor: color.

plt.scatter(x\_axis, y\_axis, s=y\_axis)

We can adjust the size by multiplying the data in the y-axis by any value. Let's multiply each data point in the y-axis by 3 and see what happens. To do this, we **can iterate through the y-axis data and multiply each data point by 3 and add it to a new list**, like this:

\*IE: first create a new list, then create a for loop that iterates on data and add that elements to the newly created list.

y\_axis\_larger = []

for data in y\_axis:

y\_axis\_larger.append(data\*3)

Then we can use the new y-axis list for the s parameter:

plt.scatter(x\_axis, y\_axis, s=y\_axis\_larger)

We can refactor the code we used to create a scatter plot above, but instead of using a for loop to create the list y\_axis\_larger for the size parameter of the marker, we can use Python's list comprehension technique inside the plt.scatter() function. You can use **list comprehension** to replace many for and while loops. List comprehension is faster because it is optimized for Python to spot a predictable pattern during looping.

<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter>

<https://matplotlib.org/stable/gallery/shapes_and_collections/scatter.html#sphx-glr-gallery-shapes-and-collections-scatter-py>

<https://docs.python.org/3.7/tutorial/datastructures.html#list-comprehensions>

<https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.scatter.html>

<https://matplotlib.org/stable/gallery/lines_bars_and_markers/scatter_with_legend.html#sphx-glr-gallery-lines-bars-and-markers-scatter-with-legend-py>

<https://matplotlib.org/stable/gallery/color/named_colors.html>

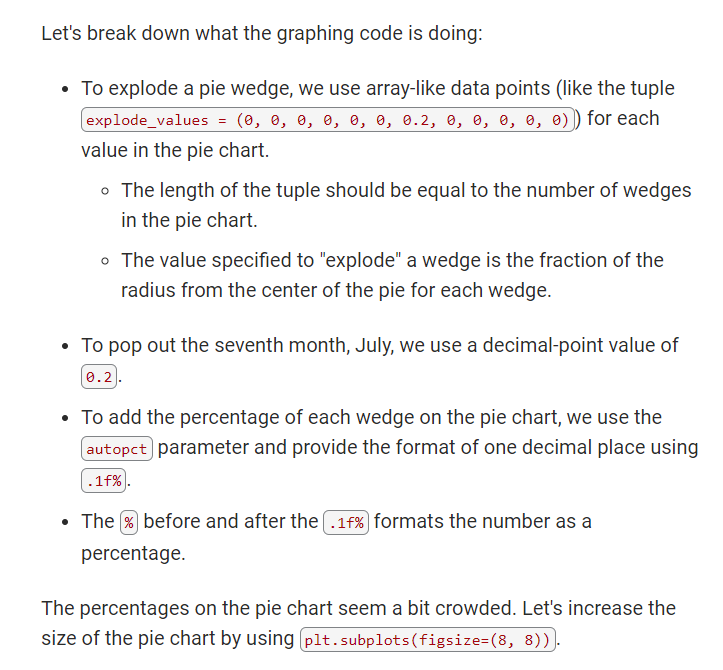
**5.1.8 – creating pie charts**

Let's add percentages for each month and "explode" the largest percentage, which is July, the seventh value in the x\_axis. The "explode" parameter will offset the indicated wedge by a fraction of the radius, where "0" is zero distance from the center of the pie, and "1" is completely outside the diameter of the pie.

explode\_values = (0, 0, 0, 0, 0, 0, 0.2, 0, 0, 0, 0, 0)

plt.pie(y\_axis, explode=explode\_values, labels=x\_axis, autopct='%.1f%%')

\*the above code will keep most elements in the pie chart at their original location on the pie chart; however, there is one element that will be ‘exploded’ away from their original position. \*NOTE: The explode\_values function requires tuples, not a list or series; the length of the tuple should be equal to the number of wedges in the pie charts.



<https://matplotlib.org/stable/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py>

**5.1.9 – chart extra**

Adding error bars can show either the standard deviation, standard error, confidence intervals, or minimum and maximum values of a dataset. When added to a chart, they can visually show the variability of the plotted data. By looking at the error bars, one can infer the significance of the data.

<https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.errorbar.html>

<https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.errorbar.html>

<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html>

<https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.bar.html>

<https://numpy.org/>

<https://numpy.org/doc/stable/reference/generated/numpy.arange.html>

<https://matplotlib.org/stable/gallery/ticks_and_spines/major_minor_demo.html>

5.1.10 – plotting dataframes and series

# Plot a Pandas DataFrame and Series

\*this is done using the ds.plot() or df.plot() functions to call either data series or data frames.

<https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html>

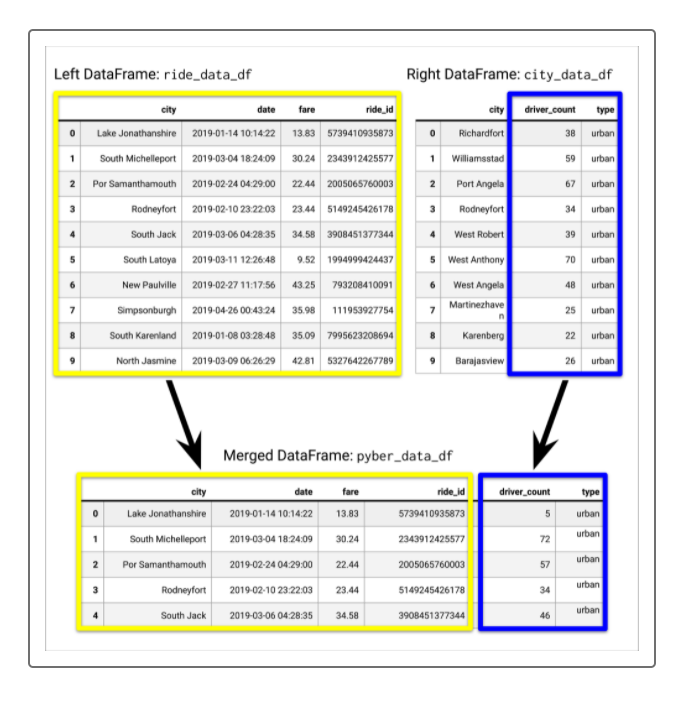
**5.2.4 – merging data frames:**

When we merge two DataFrames, we merge on a column with the same data, and the same column name, in both DataFrames. We use the following syntax to do that:

new\_df = pd.merge(leftdf, rightdf, on=["column\_leftdf", "column\_rightdf"])

We may have to merge the DataFrames using the how= parameter either left, right, inner, or outer depending how we want to merge the DataFrames. The default is inner.

In the pyber\_data\_df DataFrame, all the columns from the city\_data\_df are the first four columns after the index. The driver\_count and type columns from the ride\_data\_df are added at the end, as shown in the following image:



**3.5.2 – creating number of rides for each city type in the data files\*.**

To create a Data Series with one of the columns in a DataFrame, we can use the groupby() function and add the column inside the parentheses.

Using the groupby() function can be used to group large amounts of data when we want to compute mathematical operations on these groups.

3.5.6 – creating bubble charts of all city types:

Did you notice that we did not have to change the x-limit? That's because plotting all the data on one chart formats the x-axis automatically. We could change the y-limit from 0 to 40, but that might crowd the bubbles in the middle of the chart, making it harder to see any differences in the data.

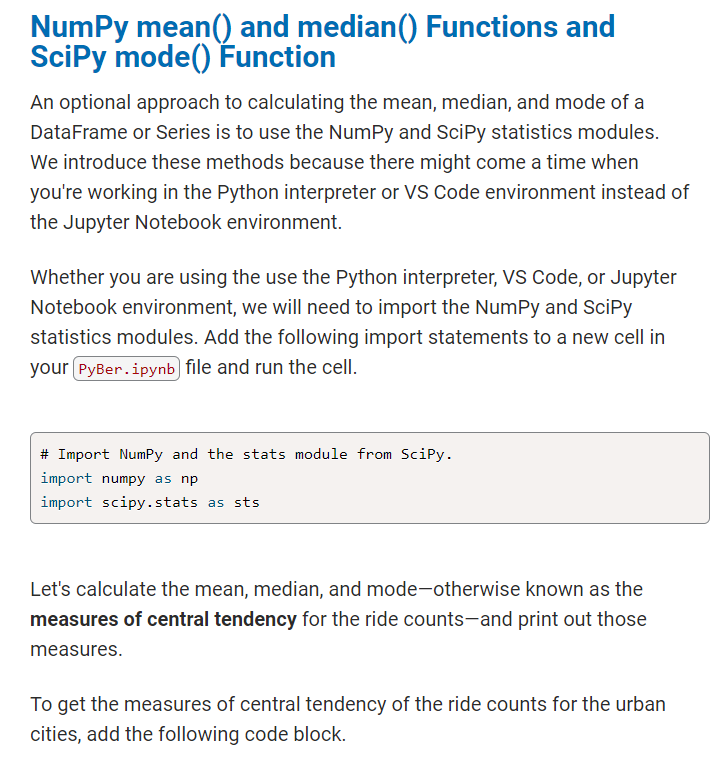
<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.legend.html>

<https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.legend.html>

<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.text.html>

5.4.1 – summary stats

* The Pandas describe() function on the DataFrame or Series.
* The Pandas mean(), median(), and mode() methods on a Series.
* The NumPy mean() and median() functions, and the SciPy stats mode() function on a Series.
* The describe() function is a convenient tool to get a high-level summary statistics on a DataFrame or Series. After running the function, the output will show the count, mean, standard deviation, minimum value, 25%, 50%, and 75% percentiles, and maximum value from a DataFrame column that has numeric values.



Unlike the Pandas mode() method, the sts.mode() method will return the number of times the mode appears in the dataset.

There are a few choices of methods to use to get the mean, median, and mode of a dataset. The method you choose is a matter of preference and depends on whether you're working with the Pandas, NumPy, or statistics modules.

5.4.4 – creating box and whisker plots