5.1 Lesson Summary - Introduction to Matplotlib

Presenting data in a chart or other visualization is a powerful way to quickly and intuitively extract meaning from that data. Charting your data is also an excellent way to illustrate patterns to others. The Python library **Matplotlib** allows you to create data visualizations directly within Python.

Concept: **NumPy** is a lower level Python library optimized for processing data in arrays. The array objects in NumPy can be created and processed significantly faster than Python's native array like objects including tuples and lists. NumPy's performance benefits make it an ideal foundation for dealing with large groups of numbers. Much of **Pandas** is built on top of NumPy. Matplotlib is also tightly integrated with NumPy. NumPy also offers useful functions for working with data. The foundational object in NumPy is the **ndarray**. It is a sequence of data similar to a Python list. A ndarray can be changed similar to Python lists and they can be multidimensional. To create a ndarray you can use the following code:

*import numpy as np*

*arr = np.array([1, 2, 3, 4])*

NumPy's *arange()* function allows you to create an ndarray according to a specified sequence. For example you could create the same ndarray in the code above by using the *arange()* function:

*import numpy as np*

*nums1thru4 = np.arange(1, 5, 1)*

* Activity: 01-Ins\_BasicLineGraphs
* Suppl link: <https://numpy.org/>

Concept: Once you have your data creating a **Matplotlib** chart can be fairly straightforward. After you have created two ndarrays named *x\_axis* and *y\_axis* you can display a basic line graph using the following commands:

*import matplotlib.pyplot as plt*

*plt.plot(x\_axis, y\_axis)*

*plt.show()*

* Activity: 01-Ins\_BasicLineGraphs, 02-Stu\_NJTemp

Concept: It is important to be able to label and **format** charts. To change the formatting of the previous example you could use the following code:

*plt.plot(x\_axis, y\_axis, marker ='x', color='blue', label="My Data")*

* Activity: 03-Ins\_ConfiguringLinePlots, 04-Stu\_LegendaryTemperature

Concept: Most charts benefit from having a **title**, **legend** and **axis** labels. To add these elements to your plot you can use the following code:

*plt.title("My Graph Title")*

*plt.xlabel("My Y Label")*

*plt.ylabel("My Y Label")*

*plt.legend(loc="best")*

If you want your chart to focus only on a small region of your data you can use the **xlim** and **ylim** methods. For example:

*plt.xlim(0,100)*

*plt.ylim(0,100)*

* Activity: 05-Ins\_Aesthetics, 06-Stu\_RollerCoaster

Concept: A **bar chart** can be useful when comparing the values of different items. To create a bar chart in Matplotlib you can use the following code:

*plt.bar(x\_axis, y\_axis, color='r', alpha=0.5, align="center")*

* Activity: 07-Ins\_BarCharts, 08-Stu\_PyBars, 13-Stu\_AvgRain

Concept: **Pie charts** can be a good indication of what percentage varying elements contribute to a whole. To create a pie chart in Matplotlib you can use the following code:

*plt.pie(datas, explode=explode, labels=labels, colors=colors, autopct="%1.1f%%", shadow=True, startangle=140)*

* Activity: 09-Ins\_PieCharts, 10-Stu\_PyPies

Concept: **Scatter plots** are useful for identifying distribution patterns in data that can have many overlapping values. To create a scatter plot with Matplotlib you can use the following code:

plt.scatter(x\_axis, data, marker="o", facecolors="red", edgecolors="black", s=x\_axis, alpha=0.75)

* Activity: 11-Ins\_ScatterPlots, 12-Stu\_ScatterPy