Assignment 1

This assignment is meant to help you familiarize yourself with JupyterLab, review Python and NumPy, and introduce you to matplotlib, a Python visualization library.

To receive credit for this assignment, answer all questions correctly in Jupyter Notebooks, save the Notebooks as PDF, and submit before the deadline.

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the labs, we ask that you **write your solutions individually**. If you do discuss the assignments with others, please include their names when submitting the assignment. (That's a good way to learn your classmates' names.).

Scoring Breakdown

Question	**Points**
Part 1: Using Jupyter	25
Part 2: Python	25
Part 3: Numpy	25
Part 4: Matlipplot	25
Total	**100**

Part 1: Using JupyterLab

JupyterLab is a new version (for the lack of better words) of Jupyter Notebook. It is notebook, text editor and python console together with a file explorer.

You can find information on installing and using JupyterLab in this tutorial.

OR

Recommended: You can use JupyterLab by downloading Anaconda found here.

1.1. Running Cells and Displaying Output

Run the following cell. If you are unfamiliar with Jupyter Notebooks, skim this tutorial. In Jupyter notebooks, all print statements are displayed below the cell. Furthermore, the output of the last line is displayed following the cell upon execution.

```
"Will this line be displayed?"

print("Hello" + ",", "world!")

5 + 3
```

1.2 Viewing Documentation

To output the documentation for a function, use the help function.

```
help(print)
```

salready be defined in the kernel for this to work. Below, click your mouse anywhere on print() and use Shift + Tab to view the function's documentation.

```
print('Welcome.')
```

1.3 Importing Libraries

In CS577, we will be using common Python libraries to help us process data. By convention, we import all libraries at the very top of the notebook. There are also a set of standard aliases that are used to shorten the library names. Try importing some of the libraries that you may encounter throughout the course, along with their respective aliases.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

A note on Keyboard Shortcuts

Even if you are familiar with Jupyter, we strongly encourage you to become proficient with keyboard shortcuts (this will save you time in the future). You can find and customize the current list of keyboard shortcuts by selecting the Advanced Settings Editor item in the Settings menu, then selecting Keyboard Shortcuts in the Settings tab.

Here are a few that we like:

1. Ctrl + Return: Evaluate the current cell 1. Shift + Return: Evaluate the current cell and move to the next 1. ESC: command mode (may need to press before using any of the commands below) 1. a: create a cell above 1. b: create a cell below 1. dd: delete a cell 1. z: undo the last cell operation 1. m: convert a cell to markdown 1. y: convert a cell to code.

Part 2: Python

Python is the main programming language we'll use in the course.

If any of the below exercises are challenging (or if you would like to refresh your Python knowledge), please review one or more of the following materials.

• Python Tutorial: Introduction to Python from the creators of Python.

• Composing Programs Chapter 1: This is more of a introduction to programming with Python.

• Advanced Crash Course: A fast crash course which assumes some programming background.

2.1. Write a function summation that evaluates the following summation for n ≥ 1:

$$\sum_{i=1}^{n} i^3 + 3i^2$$

2.2. Write a function list_sum that computes the square of each value in list_1, the cube of each value in list_2, then returns a list containing the element-wise sum of these results. Assume that list_1 and list_2 have the same number of elements.

Part 3: NumPy

NumPy is the numerical computing module, which is a prerequisite for this course.

For a quick recap, read the following materials.

• NumPy Quick Start Tutorial

• Stanford CS231n NumPy Tutorial

The core of NumPy is the array. Like Python lists, arrays store data; however, they store data in a more efficient manner. In many cases, this allows for faster computation and data manipulation. In addition to values in the array, we can access attributes such as shape and data type. A full list of attributes can be found here. Arrays, unlike Python lists, cannot store items of different data types.

Arrays are also useful in performing vectorized operations. Given two or more arrays of equal length, arithmetic will perform element-wise computations across the arrays.

3.1. Create array arr1 containing the values 1, 2, and 3 (in that order) and arr2 containing the values 4,5, and 6.

3.2. Add arr1 and arr2 element-wise.

3.3. Use NumPy to recreate your answer to Question 2.2. This time call the method as array_sum. The input parameters will both be lists, so you will need to convert the lists into arrays before performing your operations.

Hint: Use the NumPy documentation. If you're stuck, try a search engine! Searching the web for examples of how to use modules is very common in data science.

3.4. Show the time difference between list_sum and array_sum methods 2.2. and 3.3. with the following inputs using %%time command.

```
sample_list  = list(range(100000))
sample_array  = np.arange(100000)
```

Hint: %%time is a magic command, which times the execution of a cell. You can use this by writing it as the first line of a cell. (Note that %% is used for cell magic commands that apply to the entire cell).

With the large dataset, we see that using NumPy results in code that executes much faster! Throughout this course (and in the real world), you will find that writing efficient code will be important; arrays and vectorized operations are the most common way of making Python programs run quickly.

A note on np.arange and np.linspace

np.arange returns an array that steps from a to b with a fixed step size s.

np.linspace(a, b, N) divides the interval [a, b] into N equally spaced points. For example, np.linspace always includes both end points while np.arange will not include the second end point b. For this reason, when we are plotting ranges of values we tend to prefer np.linspace. Notice how the following two statements have different parameters but return the same result.

```
np.arange(-5, 6, 1.0)
array([-5., -4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.,  5.])
np.linspace(-5, 5, 11)
array([-5., -4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.,  5.])
```

Part 4: Matplotlib

4.1 Visualize the function $f(t) = 3\sin(2\pi t)$. Set the x limit of all figures to $[0, \pi]$ and the y limit to $[-10, 10]$. Plot the sine function using plt.plot with 30 red plus signs. Additionally, make sure the x ticks are labeled $[0, \pi/2, \pi]$, and that your axes are labeled as well. You can use the matplotlib documentation for reference.

Hint 1: You can set axis bounds with plt.axis.

Hint 2: You can set xticks and labels with plt.xticks.

Hint 3: Make sure you add plt.xlabel, plt.ylabel, plt.title.