Lab 01 - Linear Algebra and Numpy

Introduction:

This notebook's purpose is to refresh lessons taught in prerequisite courses for CS3400. This includes connecting to a remote resouce, which in this lab's case is MSOE's ROSIE super computer. In addition, this lab makes students recall computational science tools and their usage, such as Jupyter Notebooks and the Numpy library. Specifically, the Numpy library is used heavily in this notebook in order to remind students how to properly manipulate arrays of all dimensions, and also to recall linear alegra operations and rules that were learned prior to CS3400.

Author: Nigel Nelson

Experiment 1

In this experiment you will be making sure that you can connect to ROSIE and run an interactive session (jupyter notebook session). You should have a username on ROSIE by the start of class, but you might have to reset your password. All local user accounts signed up for this class will have a default password of 'g0-raid3rS'. At the current time, to do this you will have to access the terminal on ROSIE - meaning you will have to ssh in. Once you have reset your password, you will be able to access ROSIE's web portal and initiate interactive session from there. The following steps and sections will give you what you need to start.

Accessing ROSIE

An objective of this class is to give you some more experinece using remote resources and ROSIE is a great resource to have. Our current (iterim) admin is Dr. Taylor. While Dr. Taylor is assisting us, please refer to ROSIE's [webpage (https://msoe.dev/#/)](https://msoe.dev/#/) as a first step in finding solutions to issues you may be having. Your instructor is also a good resource if troubleshooting is required.

SSH Client

If you are on windows, you will have to download and install an ssh client. A commonly used and free client is [Putty (https://www.putty.org/)](https://www.putty.org/). Please follow the link and install Putty on your machine.

On network or off

If you are doing these steps off-campus, you will need to use a VPN to access the network that ROSIE is on. To do this you can follow the written instruction on [msoe.dev - Activiation (https://msoe.dev/#/activate)](https://msoe.dev/#/activate).

Starting an Interactive session

Once you have access to ROSIE's network (VPN) and you have a username and current password (done through the SSH client), you can now complete the steps for starting an interactive session. You should access [ROSIE's web portal (http://dh-ood.hpc.msoe.edu/)](http://dh-ood.hpc.msoe.edu/) and start a jupyter notebook session to run (and complete) this notebook.

Experiment 2 - Structuing your Data and Feature Matrices / Slicing

In this experiment you will refamiliarize yourself with python/numpy and use some of the common data manipulation techniques that you will need for the rest of the class.

What is Numpy?

- Matrix library
- Memory-efficient data structures -- arrays
  - Used in scikit-learn, matplotlib, and others
- Expressive API for indexing and operations
- Time-efficient algorithms
  - Calls C and Fortran libraries where possible

How Do I Import Libraries into my Jupyter Notebook working kernel?

- The following bit of code can be used to import libraries. The world is your oyster!

In [1]:

```
import numpy as np
import scipy
import scipy.stats as stats
import matplotlib.pyplot as plt
from IPython.display import Image
picturename = '/data/cs3400/misc/mb.gif'
datapath = '/data/cs3400/datasets/IRIS.csv'
```

How to read in files, organize data, and plot some features!

In the first step you will read the IRIS.csv file that you are given (which is also on our class's datashare on ROSIE) and put the features into a matrix. In machine learning the standard for organizing matrices is always observations in rows, and features that describe the observations as columns. Read in the data file and assign the data to a numpy matrix.

1. Use the function numpy.loadtxt.
   - You will want to use the proper delimiter for the file you have.
   - Make sure that you skip any text rows, numpy matrices can only be a single datatype.
   - Depending on the dataset you may need to specify what columns you want to use.
   - If you get stuck and don't want to head to the web, you can always use the the help command for more information e.g. `help(np.loadtext)`

With your data matrix you should explore the data a bit.

2. Use `data.shape` to find your dimensions
3. Plot the first two features your data using matplotlib. Label all of your axes and use legends!
    A. Make a figure with a line plot
    B. Make another figure with a scatter plot
    C. Make a third figure displaying both the same line and scatter plots.
4. Print all of the feature values for the 150th observation in your dataset.
5. Select observations 49-52 from your dataset and print them to the notebook.
6. Select all of the entries in your dataset that have their first feature <= 5 and print the first 5 results. (hint: do this in multiple steps. First make a boolean mask of your matrix)
7. Calculate the median, standard deviation, and mode of the entries selected in the previous step. (Hint 1: these should be done column by column. Hint 2: Don't forget about other packages like scipy!)

1) Load the IRIS.csv file into a numpy matrix

In [2]:
```
data = np.loadtxt(datapath, delimiter=",", usecols=(0,1,2,3), skiprows=1)
```

2) Display its dimensions (`data.shape`)

In [3]:
```
data.shape
```

Out[3]:
```
(150, 4)
```

Plot the first two features of your data using matplotlib. Label all of your axes and use legends!

In [4]:
```
species = np.loadtxt(datapath, delimiter=",", dtype='S', usecols=(4), skiprow

setosa_mask = species == b'Iris-setosa'
versicolor_mask = species == b'Iris-versicolor'
virginica_mask = species == b'Iris-virginica'
```

In [5]:
```
sepal_length = data[:, 0]
sepal_width = data[:, 1]

setosa_sepal_length = sepal_length[setosa_mask]
setosa_sepal_width = sepal_width[setosa_mask]

versicolor_sepal_length = sepal_length[versicolor_mask]
versicolor_sepal_width = sepal_width[versicolor_mask]

virginica_sepal_length = sepal_length[virginica_mask]
virginica_sepal_width = sepal_width[virginica_mask]
```

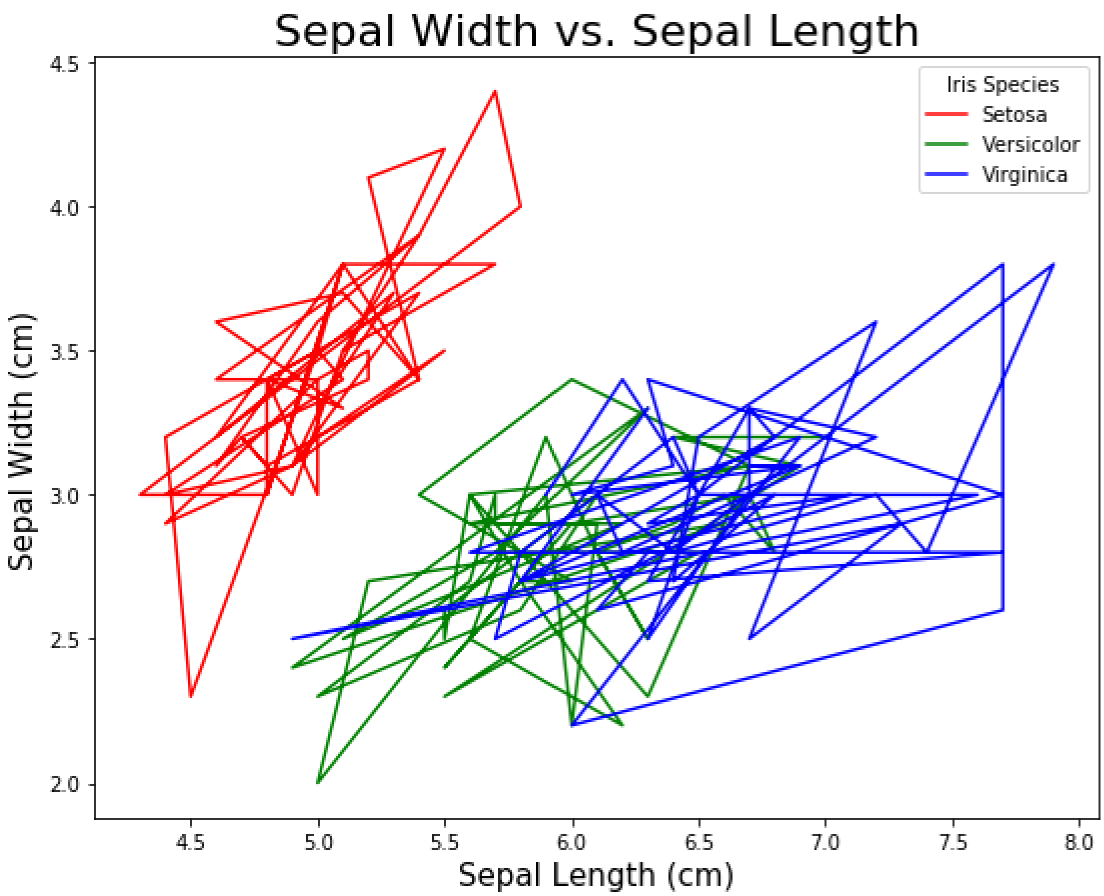
3-A) Make a line plot of the first two dimensions using matplotlib

In [6]:
```
figure, axis = plt.subplots(figsize=(9,7))

axis.plot(setosa_sepal_length, setosa_sepal_width, label="Setosa", c='r')
axis.plot(versicolor_sepal_length, versicolor_sepal_width, label="Versicolor'
axis.plot(virginica_sepal_length, virginica_sepal_width, label="Virginica", c

axis.set_xlabel("Sepal Length (cm)", fontsize=15)
axis.set_ylabel("Sepal Width (cm)", fontsize=15)
axis.set_title("Sepal Width vs. Sepal Length", fontsize=22)
plt.legend(title="Iris Species")
```

Out[6]: <matplotlib.legend.Legend at 0x7f1783003828>



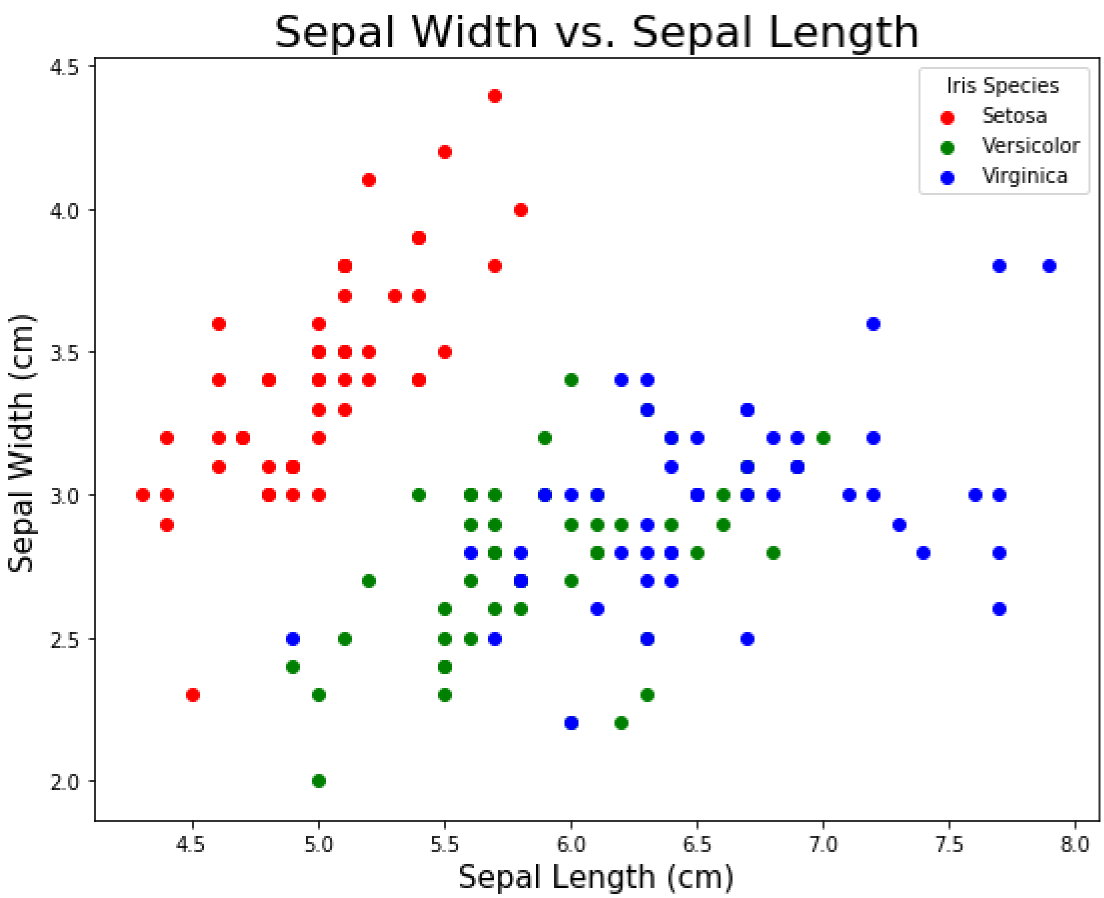
3-B) Make a scatter plot of the first two dimensions using matplot lib

In [7]:
```
figure, axis = plt.subplots(figsize=(9,7))

axis.scatter(setosa_sepal_length, setosa_sepal_width, label="Setosa", c='r')
axis.scatter(versicolor_sepal_length, versicolor_sepal_width, label="Versicol
axis.scatter(virginica_sepal_length, virginica_sepal_width, label="Virginica'

axis.set_xlabel("Sepal Length (cm)", fontsize=15)
axis.set_ylabel("Sepal Width (cm)", fontsize=15)
axis.set_title("Sepal Width vs. Sepal Length", fontsize=22)
plt.legend(title="Iris Species")
```

Out[7]: <matplotlib.legend.Legend at 0x7f1781736b70>



3-C) Make a third figure displaying both the same line and scatter plots

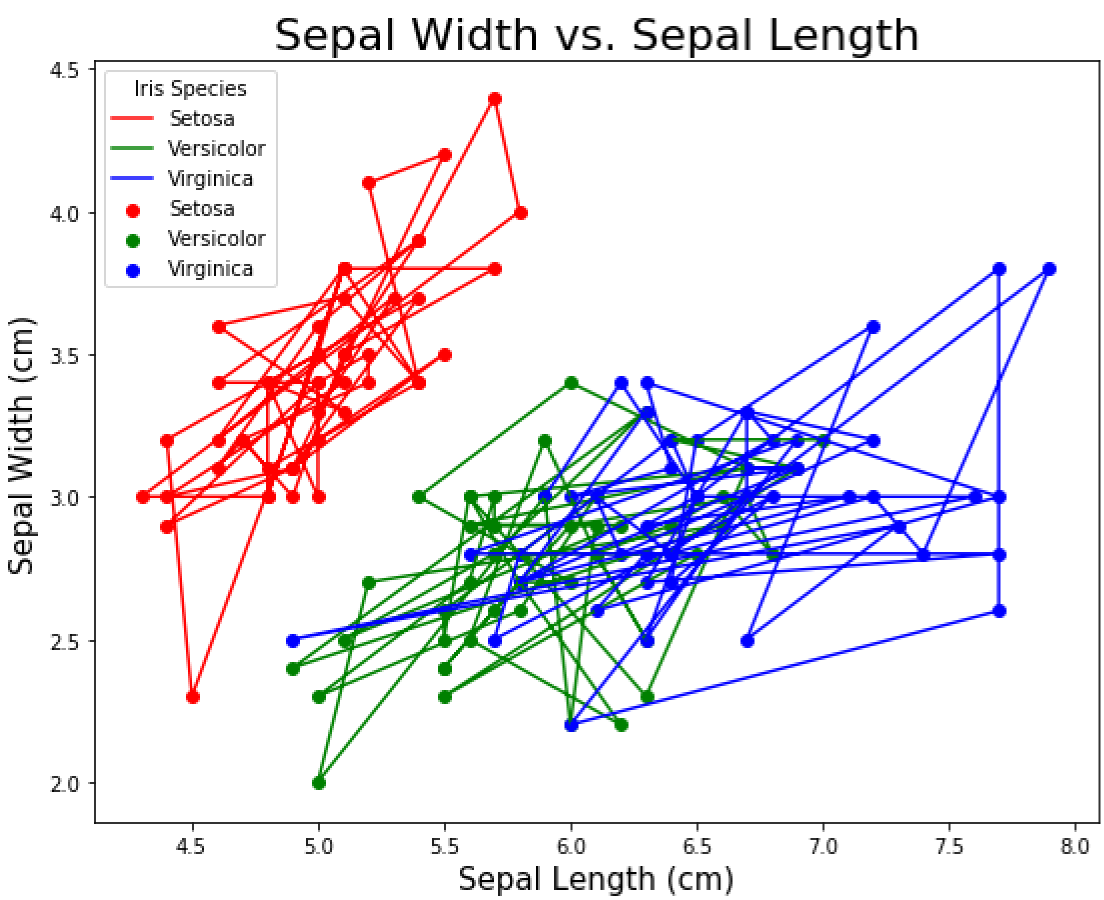
In [8]:
```
figure, axis = plt.subplots(figsize=(9,7))

axis.scatter(setosa_sepal_length, setosa_sepal_width, label="Setosa", c='r')
axis.scatter(versicolor_sepal_length, versicolor_sepal_width, label="Versicol
axis.scatter(virginica_sepal_length, virginica_sepal_width, label="Virginica'

axis.plot(setosa_sepal_length, setosa_sepal_width, label="Setosa", c='r')
axis.plot(versicolor_sepal_length, versicolor_sepal_width, label="Versicolor'
axis.plot(virginica_sepal_length, virginica_sepal_width, label="Virginica", c

axis.set_xlabel("Sepal Length (cm)", fontsize=15)
axis.set_ylabel("Sepal Width (cm)", fontsize=15)
axis.set_title("Sepal Width vs. Sepal Length", fontsize=22)
plt.legend(title="Iris Species")
```

Out[8]: <matplotlib.legend.Legend at 0x7f17816b0a58>



4) Print all of the feature values for the 150th observation in your dataset.

In [9]:
```
def print_observation(iris_observation):
    print('Sepal Length: ' + str(iris_observation[0]))
    print('Sepal Width: ' + str(iris_observation[1]))
    print('Petal Length: ' + str(iris_observation[2]))
    print('Petal Width: ' + str(iris_observation[3]))
```

In [10]:

```
print("Observation #150 in the Iris Data Set:")
print_observation(data[149])
```

```
Observation #150 in the Iris Data Set:
Sepal Length: 5.9
Sepal Width: 3.0
Petal Length: 5.1
Petal Width: 1.8
```

5) Select observations 49-52 from your dataset and print them to the notebook.

In [11]:

```
for n in range(48,52):
    print("Observation #" + str(n + 1) + " in the Iris Data Set:")
    print_observation(data[n])
```

```
Observation #49 in the Iris Data Set:
Sepal Length: 5.3
Sepal Width: 3.7
Petal Length: 1.5
Petal Width: 0.2
Observation #50 in the Iris Data Set:
Sepal Length: 5.0
Sepal Width: 3.3
Petal Length: 1.4
Petal Width: 0.2
Observation #51 in the Iris Data Set:
Sepal Length: 7.0
Sepal Width: 3.2
Petal Length: 4.7
Petal Width: 1.4
Observation #52 in the Iris Data Set:
Sepal Length: 6.4
Sepal Width: 3.2
Petal Length: 4.5
Petal Width: 1.5
```

6) Select all of the entries in your dataset that have their first feature <= 5 and print the first 5 results (Hint: Do this in multiple steps. First make a boolean mask of your matrix)

In [12]:

```
mask = data[:, 0] <= 5
maskdata = data[mask]
for n in range(0,5):
    print("Observation #" + str(n + 1) + " in the masked Iris Data Set:")
    print_observation(maskdata[n])
```

```
Observation #1 in the masked Iris Data Set:
Sepal Length: 4.9
Sepal Width: 3.0
Petal Length: 1.4
Petal Width: 0.2
Observation #2 in the masked Iris Data Set:
Sepal Length: 4.7
Sepal Width: 3.2
Petal Length: 1.3
Petal Width: 0.2
Observation #3 in the masked Iris Data Set:
Sepal Length: 4.6
Sepal Width: 3.1
Petal Length: 1.5
Petal Width: 0.2
Observation #4 in the masked Iris Data Set:
Sepal Length: 5.0
Sepal Width: 3.6
Petal Length: 1.4
Petal Width: 0.2
Observation #5 in the masked Iris Data Set:
Sepal Length: 4.6
Sepal Width: 3.4
Petal Length: 1.4
Petal Width: 0.3
```

7) Calculate the median, standard deviation, and mode of the entries selected in the previous step. (Hint 1: these should be done column by column. Hint 2: Don't forget about other packages like scipy!)

In [13]:
```
print("Stats on the collection of Iris whose sepal length <= 5.0 cm:")
print()
print("Median Sepal Length: " + str(np.median(maskdata[:,0])))
print("Median Sepal Width: " + str(np.median(maskdata[:,1])))
print("Median Petal Length: " + str(np.median(maskdata[:,2])))
print("Median Petal Width: " + str(np.median(maskdata[:,3])))
print()
print("Standard Deviation Sepal Length: " + str(np.std(maskdata[:,0])))
print("Standard Deviation Sepal Width: " + str(np.std(maskdata[:,1])))
print("Standard Deviation Petal Length: " + str(np.std(maskdata[:,2])))
print("Standard Deviation Petal Width: " + str(np.std(maskdata[:,3])))
print()
print("Mode Sepal Length: " + str(stats.mode(maskdata[:,0])[0][0]))
print("Mode Sepal Width: " + str(stats.mode(maskdata[:,1])[0][0]))
print("Mode Petal Length: " + str(stats.mode(maskdata[:,2])[0][0]))
print("Mode Petal Width: " + str(stats.mode(maskdata[:,3])[0][0]))
```

```
Stats on the collection of Iris whose sepal length <= 5.0 cm:

Median Sepal Length: 4.85
Median Sepal Width: 3.1
Median Petal Length: 1.45
Median Petal Width: 0.2

Standard Deviation Sepal Length: 0.21323402636539976
Standard Deviation Sepal Width: 0.38649062084350766
Standard Deviation Petal Length: 0.7729559738432714
Standard Deviation Petal Width: 0.3470860844228705

Mode Sepal Length: 5.0
Mode Sepal Width: 3.0
Mode Petal Length: 1.4
Mode Petal Width: 0.2
```

Experiment 3 - Linear Algebra in Numpy

In this experiment you will be performing a number of linear algebra operations in your jupyter notebook. Check out the linalg module of numpy!

We have started by creating a few vectors and matrices for you.

In [14]:
```
array_1 = np.array([1, 2, 3, 4, 5], dtype=np.float32)
print(array_1)
array_2 = np.zeros(4, dtype=np.int32)
print(array_2)
matrix_1 = np.ones((4,5), dtype=np.float64)
print(matrix_1)
matrix_2 = np.eye(5,5)
print(matrix_2)
```

```
[1. 2. 3. 4. 5.]
[0 0 0 0]
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

You will:

1. Create a few more numpy vectors and matrices
2. Print the number of dimensions each of your numpy vectors and matrices
3. Print the shape (length and dimension) of each of your numpy vectors and matrices
4. Print the datatype used in each of your numpy vectors and matrices
5. Try to compute a dot product on two matrices of with disagreeable dimensions
6. Compute a dot product on two matrices with agreeable dimensions
7. Try to compute element-wise addition on two matrices with disagreeable dimensions
8. Compute an element-wise addition on two matrices with agreeable dimensions
9. Compute the norm (distance) between a vector and itself
10. Compute the norm (distance) between two different vectors
11. Apply a set of linear coefficients to a matrix of observations.

1) Create numpy vectors and matrices (we have done a few for you)

In [15]:
```
matrix_3 = np.identity(5, dtype=np.int8)
print("matrix_3:")
print(matrix_3)
print()
matrix_4 = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
print("matrix_4:")
print(matrix_4)
print()
array_3 = np.full(7, 9, dtype=np.float32)
print("array_3:")
print(array_3)
print()
array_4 = np.ones_like(array_1)
print("array_4:")
print(array_4)
print()
```

```
matrix_3:
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]

matrix_4:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

array_3:
[9. 9. 9. 9. 9. 9. 9.]

array_4:
[1. 1. 1. 1. 1.]
```

In [16]:
```
arrays = [matrix_3, matrix_4, array_3, array_4]
array_names = ["matrix_3", "matrix_4", "array_3", "array_4"]
```

2) Print the number of dimensions each of your numpy vectors and matrices

In [17]:
```
for n in range(0,4):
    print(array_names[n] + "'s number of dimensions: " + str(arrays[n].ndim))
```

```
matrix_3's number of dimensions: 2
matrix_4's number of dimensions: 2
array_3's number of dimensions: 1
array_4's number of dimensions: 1
```

3. Print the shape (length and dimension) of each of your numpy

vectors and matrices

In [18]:
```
for n in range(0,4):
    print(array_names[n] + "'s shape: " + str(arrays[n].shape))
```

```
matrix_3's shape: (5, 5)
matrix_4's shape: (4, 3)
array_3's shape: (7,)
array_4's shape: (5,)
```

4) Print the datatype used in each of your numpy vectors and matrices

In [19]:
```
for n in range(0,4):
    print(array_names[n] + "'s datatype: " + str(arrays[n].dtype))
```

```
matrix_3's datatype: int8
matrix_4's datatype: int64
array_3's datatype: float32
array_4's datatype: float32
```

5) Try to compute a dot product on two matrices of with disagreeable dimensions

In [20]:
```
print("Dot product of matrix_4 and matrix_2:")
np.dot(matrix_4, matrix_2)
```

```
Dot product of matrix_4 and matrix_2:

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-78822cdc9599> in <module>
      1 print("Dot product of matrix_4 and matrix_2:")
----> 2 np.dot(matrix_4, matrix_2)

ValueError: shapes (4,3) and (5,5) not aligned: 3 (dim 1) != 5 (dim 0)
```

6) Compute a dot product on two matrices with agreeable dimensions

In [21]:
```
print("Dot product of matrix_1 and matrix_3:")
np.dot(matrix_1, matrix_3)
```

```
Dot product of matrix_1 and matrix_3:
```

Out[21]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

7) Try to compute element-wise addition on two matrices with disagreeable dimensions

In [22]:
```
print("Element-wise addition between matrix_3 and matrix_1:")
np.add(matrix_3, matrix_1)
```

```
Element-wise addition between matrix_3 and matrix_1:

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-ffba9c728f8b> in <module>
      1 print("Element-wise addition between matrix_3 and matrix_1:")
----> 2 np.add(matrix_3, matrix_1)

ValueError: operands could not be broadcast together with shapes (5,5) (4,
5)
```

8) Compute an element-wise addition on two matrices with agreeable dimensions

In [23]:
```
print("Element-wise addition between matrix_3 and matrix_2:")
np.add(matrix_3, matrix_2)
```

```
Element-wise addition between matrix_3 and matrix_2:
```

Out[23]:
```
array([[2., 0., 0., 0., 0.],
       [0., 2., 0., 0., 0.],
       [0., 0., 2., 0., 0.],
       [0., 0., 0., 2., 0.],
       [0., 0., 0., 0., 2.]])
```

9) Compute the norm (distance) between a vector and itself

In [24]:
```
print("The norm between array_4 and itself:")
np.linalg.norm(array_4 - array_4)
```

```
The norm between array_4 and itself:
```

Out[24]:
```
0.0
```

10) Compute the norm (distance) between two different vectors

In [25]:
```
print("The norm between array_1 and array_4:")
np.linalg.norm(array_1 - array_4)
```

```
The norm between array_1 and array_4:
```

Out[25]:
```
5.477226
```

11) Apply a set of linear coefficients to a matrix of observations.

From your problem set you can see the form of this model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

which can also be represented in vector notation as:

$$y = x^T \beta$$

Use the vectors that you created in problem 5 of problem set 1 and evaluate it here. Evaluate it twice, once using matrix multiplication and once with dot products

In [26]:
```
x = np.array([1,2,3,4])
B = np.array([5,6,7,8])

dot_product = np.dot(x, B)
mat_mul = np.matmul(x.T, B)

print("Dot product of matrices x and B: " + str(dot_product))
print("Matrix multiplication result of matrices x and B: " + str(mat_mul))
```

```
Dot product of matrices x and B: 70
Matrix multiplication result of matrices x and B: 70
```

Bonus Material: Additional Indexing Topics

Before considering the following indexing procedures, think about the following question. Can I index a vector (nx1) using a matrix (nxm)? What would happen if I try?

In [27]:
```
X = np.random.randint(10, size=(10, 3))
y = np.expand_dims(np.array([1, 0, 1, 1, 0, 0, 2, 2, 1, 0], dtype=np.int32),a
```

Think of the above matrix, X, as a feature matrix (10x3) and the above vector, y, as a response vector/matrix (10x1). How can I index and get the first index of X or y?

In [28]:
```
y[0,0]
```

Out[28]: 1

In [29]:
```
X[0,0]
```

Out[29]: 5

What if I want multiple elements from this array that are not sequential? Such as element 0 and element 7?

In [30]:
```
print(y[0,0])
print(y[7,0])
```

```
1
2
```

Pretty straighforward, eh? Can I do this in one go?

In [31]:
```
print(y[[0,7],[0,0]])
```

```
[1 2]
```

Not too shabby! Now, is there anything preventing me from re-indexing the same element? Let's try!

In [32]:
```
print(y[[7,7],[0,0]])
```

```
[2 2]
```

woah

Finally, lets take this to a rediculous conclusion... What happens if I supply more index calls (as a matrix) than the variable has in shape?

In [33]:
```
print(y[[7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

In [34]:
```
Image(picturename)
```

Out[34]:

Conclusion

This notebook's purpose was to refresh lessons taught in prerequisite courses for CS3400. Ultimately this goal was accomplished by completing the Jupyter notebook stub that was provided for Lab 1. Thanks to this lab many past lessons came rushing back, such as how to properly use a remote resource such as MSOE's ROSIE super computer, and also how to properly run and use Jupyter Notebooks on this remote resource. Primarily, many lessons on linear alegbra were recalled thanks to the various Numpy array mannipulations that were requested to be performed in this notebook.