

# lab\_01

September 10, 2025

## 1 Lab 01 - Linear Algebra and Numpy

Welcome to the first lab in CS3400! If you can edit this you probably have a working instance of jupyter notebook (either locally or on ROSIE). If you are looking at this as an pdf, maybe you still need to get an instance of Jupyter running. Please follow the necessary steps in Experiment 1.

## 2 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 will have a username on ROSIE by the start of class, but if it is a new account you may have to reset your password. Ask your instructor for the default password. 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.

### 2.1 Accessing ROSIE

An objective of this class is to give you some more experience using remote resources and ROSIE is a great resource to have. Our current admin is Dr. Retert. Please refer to ROSIE's [webpage](#) as a first step in finding solutions to issues you may be having. Your instructor is also a good resource if troubleshooting is required.

#### 2.1.1 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](#). Please follow the link and install Putty on your machine.

#### 2.1.2 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 - Activation](#).

#### 2.1.3 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](#) and start a jupyter notebook session to run (and complete) this notebook.

## 3 Experiment 2 - Structuring 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.

### 3.1 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

### 3.2 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!

```
[1]: # importing all required libraries
import numpy as np
import scipy
import scipy.stats as stats
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import Image

# finding the paths and placing them in variables
datapath = 'data/IRIS.csv'
```

### 3.3 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.loadtxt)`

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! 1. Make a figure with a line plot 2. Make another figure with a scatter plot 3. 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  $\leq 5$  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!)

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

```
[2]: # viewing the .csv file
df = pd.read_csv(datapath)
df
```

```
[2]:      sepal_length  sepal_width  petal_length  petal_width      species
0              5.1           3.5           1.4           0.2  Iris-setosa
1              4.9           3.0           1.4           0.2  Iris-setosa
2              4.7           3.2           1.3           0.2  Iris-setosa
3              4.6           3.1           1.5           0.2  Iris-setosa
4              5.0           3.6           1.4           0.2  Iris-setosa
..           ...           ...           ...           ...           ...
145             6.7           3.0           5.2           2.3  Iris-virginica
146             6.3           2.5           5.0           1.9  Iris-virginica
147             6.5           3.0           5.2           2.0  Iris-virginica
148             6.2           3.4           5.4           2.3  Iris-virginica
149             5.9           3.0           5.1           1.8  Iris-virginica
```

[150 rows x 5 columns]

```
[3]: # load .csv file into matrix / skip_header=1 removes the first row of data /
      ↪ usecols uses only numerical columns
matrix = np.genfromtxt(datapath, delimiter=',', skip_header=1,
      ↪ usecols=(0,1,2,3))
```

### 3.3.2 2) Display its dimensions (data.shape)

```
[4]: # displaying the shape of the matrix
print("The Shape is: ", matrix.shape)
```

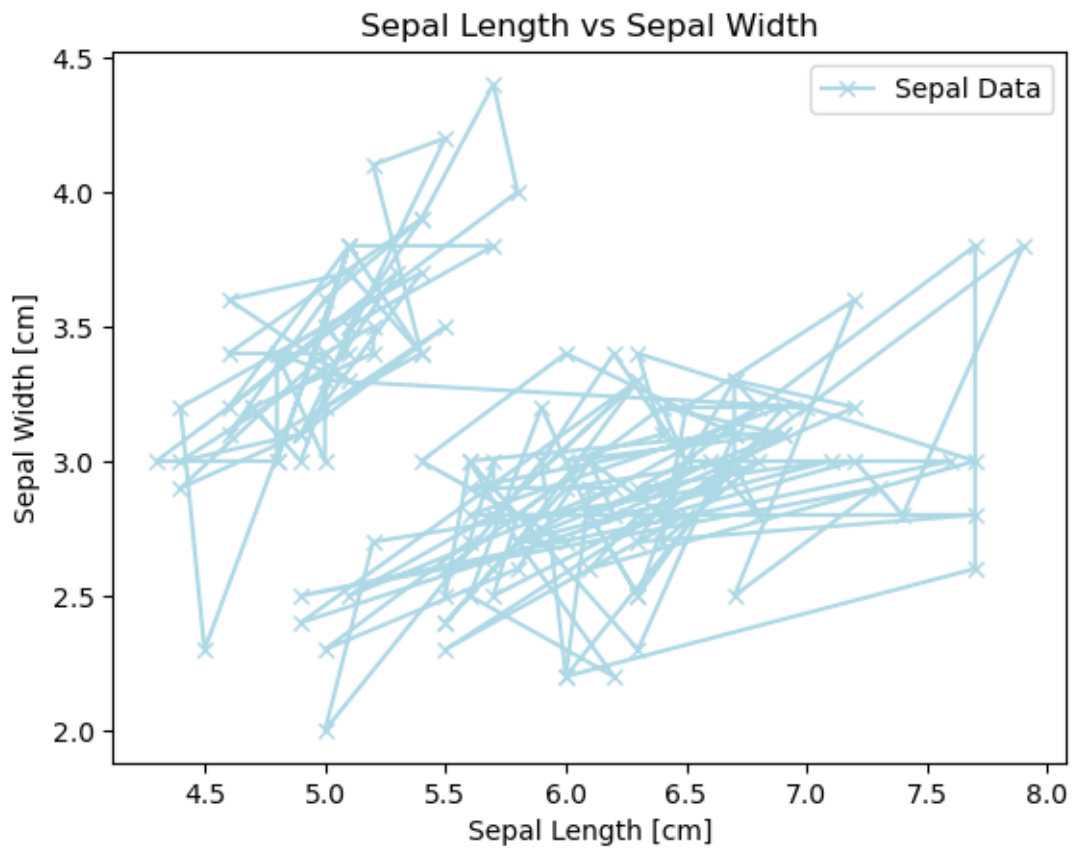
The Shape is: (150, 4)

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

#### 3.3.4 3-A) Make a line plot of the first two dimensions using matplotlib

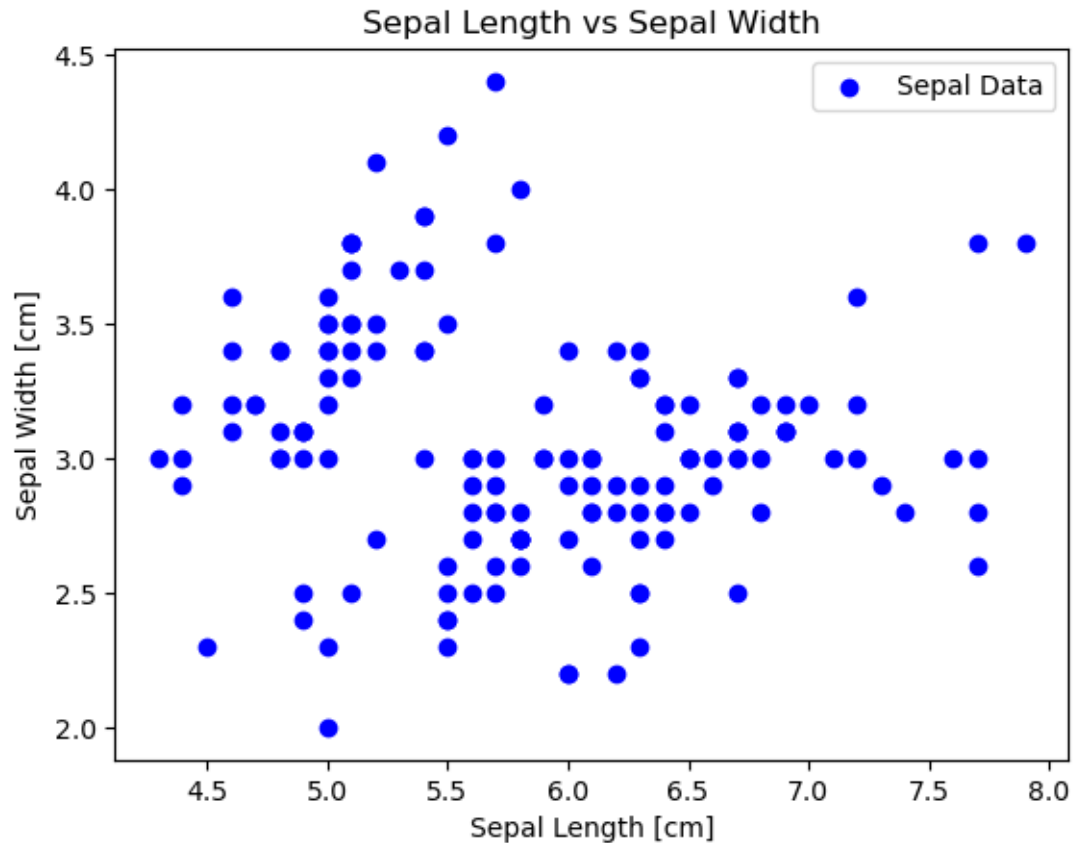
```
[5]: # plotting sepal_length vs sepal_width
plt.plot(matrix[:,0], matrix[:,1], label="Sepal Data", color='lightblue',
      ↪ marker='x')
plt.xlabel("Sepal Length [cm]")
plt.ylabel("Sepal Width [cm]")
```

```
plt.legend()
plt.title("Sepal Length vs Sepal Width")
plt.show()
```



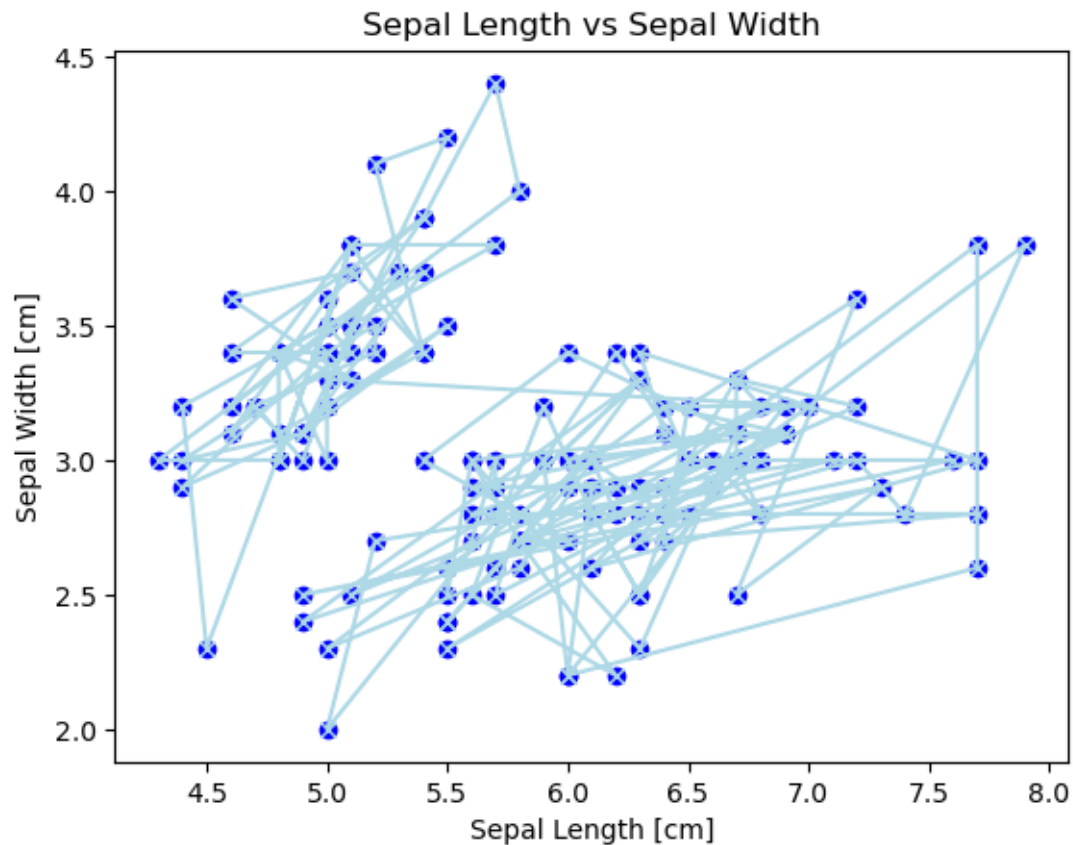
### 3.3.5 3-B) Make a scatter plot of the first two dimensions using matplotlib lib

```
[6]: # plotting sepal_length vs sepal_width
plt.scatter(matrix[:,0], matrix[:,1], label="Sepal Data", color='blue')
plt.xlabel("Sepal Length [cm]")
plt.ylabel("Sepal Width [cm]")
plt.legend()
plt.title("Sepal Length vs Sepal Width")
plt.show()
```



### 3.3.6 3-C) Make a third figure displaying both the same line and scatter plots

```
[7]: plt.plot(matrix[:,0], matrix[:,1], label="Sepal Data", color='lightblue',
            marker='x')
plt.scatter(matrix[:,0], matrix[:,1], label="Sepal Data", color='blue')
plt.xlabel("Sepal Length [cm]")
plt.ylabel("Sepal Width [cm]")
plt.title("Sepal Length vs Sepal Width")
plt.show()
```



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

```
[8]: # the 150th observation, since we start at 0, we use 149
print(matrix[149, :])
```

```
[5.9 3.  5.1 1.8]
```

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

```
[9]: print(matrix[48:52, :])
```

```
[[5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]]
```

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

```
[10]: # creating a boolean mask
bool_mask = matrix[:,0] <= 5

# performing filtering
filtered_rows = matrix[bool_mask]

# the first five results
print(filtered_rows[:5])
```

```
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [4.6 3.4 1.4 0.3]]
```

**3.3.9 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!)

```
[11]: # calculating the median
entry_median = np.median(filtered_rows, axis=0)
print("Median: ", entry_median)

# calculating the standard deviation
entry_std = np.std(filtered_rows, axis=0)
print("Standard Deviation: ", entry_std)

# calculating the mode
entry_mode = stats.mode(filtered_rows, axis=0, keepdims=True)
print("Mode: ", entry_mode.mode[0])
```

```
Median:  [4.85 3.1  1.45 0.2 ]
Standard Deviation:  [0.21323403 0.38649062 0.77295597 0.34708608]
Mode:  [5.  3.  1.4 0.2]
```

## 4 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.

```
[12]: 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.

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

```

[13]: # creating a numpy vector
vector = np.array([1, 2, 3, 4, 5])
print("Numpy Vector: ", vector)

print("")

# creating a numpy matrix
matrix = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

print("Matrix: ")
for row in matrix:
    print(row)

```

Numpy Vector: [1 2 3 4 5]



```
Matrix:
[1 2 3]
[4 5 6]
[7 8 9]
```

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

```
[14]: # printing out the number of dimensions in a numpy vector
print("Number of Dimensions in Vector: ", vector.ndim)

# printing out the number of dimensions in a numpy matrix
print("Number of Dimensions in Matrix: ", matrix.ndim)
```

```
Number of Dimensions in Vector: 1
Number of Dimensions in Matrix: 2
```

#### 4.0.3 3. Print the shape (length and dimension) of each of your numpy vectors and matrices

```
[15]: # printing out the number of dimensions in a numpy vector
print("Shape of Vector: ", vector.shape)

# printing out the number of dimensions in a numpy matrix
print("Shape of Matrix: ", matrix.shape)
```

```
Shape of Vector: (5,)
Shape of Matrix: (3, 3)
```

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

```
[16]: # printing out the number of dimensions in a numpy vector
print("Data Type of Vector: ", vector.dtype)

# printing out the number of dimensions in a numpy matrix
print("Data Type of Matrix: ", matrix.dtype)
```

```
Data Type of Vector: int64
Data Type of Matrix: int64
```

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

```
[17]: # m x n where m -> row and n -> column

# matrix A is 3x3
matrix_a = np.array([
    [1, 2, 3],
    [1, 2, 3],
```

```

    [1, 2, 3]
])

# matrix B is 2x3
matrix_b = np.array([
    [1, 2, 3],
    [1, 2, 3]
])

# finding the dot product
# need to use try-catch for safety

try:
    print(np.dot(matrix_a, matrix_b))
except ValueError as error:
    print(error)

```

shapes (3,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)

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

[18]: *# m x n where m -> row and n -> column*

```

# matrix A is 3x3
matrix_a = np.array([
    [1, 2, 3],
    [1, 2, 3],
    [1, 2, 3]
])

# matrix B is 3x2
matrix_b = np.array([
    [1, 2],
    [1, 2],
    [1, 2]
])

# finding the dot product
# need to use try-catch for safety

try:
    print(np.dot(matrix_a, matrix_b))
    print("Success")
except ValueError as error:
    print(error)

```

```
[[ 6 12]
 [ 6 12]
 [ 6 12]]
```

Success

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

```
[19]: # m x n where m -> row and n -> column
```

```
# matrix A is 3x3
matrix_a = np.array([
    [1, 2, 3],
    [1, 2, 3],
    [1, 2, 3]
])

# matrix B is 2x3
matrix_b = np.array([
    [1, 2, 3],
    [1, 2, 3]
])

# finding the dot product
# need to use try-catch for safety

try:
    print(matrix_a + matrix_b)
except ValueError as error:
    print(error)
```

operands could not be broadcast together with shapes (3,3) (2,3)

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

```
[20]: # m x n where m -> row and n -> column
```

```
# matrix A is 3x3
matrix_a = np.array([
    [1, 2, 3],
    [1, 2, 3],
    [1, 2, 3]
])

# matrix B is 3x3
matrix_b = np.array([
```

```

    [1, 2, 3],
    [1, 2, 3],
    [1, 2, 3]
])

# finding the dot product
# need to use try-catch for safety

try:
    print(matrix_a + matrix_b)
    print("Success")
except ValueError as error:
    print(error)

```

```

[[2 4 6]
 [2 4 6]
 [2 4 6]]
Success

```

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

```

[21]: # defining a new vector
vector = np.array([1, 2, 3])

# distance computation
distance_one = np.linalg.norm(vector)
print("Computed Distance: ", distance_one)

```

Computed Distance: 3.7416573867739413

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

```

[22]: # defining vector one and vector two
vector_one = np.array([1, 2, 3])
vector_two = np.array([1, 2, 3])

# computing distance between the two vectors
distance_two = np.linalg.norm(vector_one - vector_two)
print("Computer Distance: ", distance_two)

```

Computer Distance: 0.0

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

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

$$y = \beta_0^{12} + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

which can also be represented in vector notation as:

$$y = x^T$$

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

```
[ ]: # creating a coefficient beta vector
beta_vector = np.array([0.5, 1, -1, 2])

# matrix of observations
obs_matrix = np.array([

    [1, 2, 3, 4],
    [1, 2, 3, 4],
    [1, 2, 3, 4]
])

# x^T x beta
y_matrix = obs_matrix @ beta_vector
print(y_matrix)

# using dot product
y_dot = np.dot(obs_matrix, beta_vector)
print(y_dot)

[7.5 7.5 7.5]
[7.5 7.5 7.5]
```

## 5 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?

```
[24]: 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),axis=1)
```

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?

```
[25]: y[0,0]
```

```
[25]: np.int32(1)
```

```
[26]: X[0,0]
```

```
[26]: np.int32(9)
```

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

```
[27]: print(y[0,0])  
      print(y[7,0])
```

```
1  
2
```

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

```
[28]: 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!

```
[29]: print(y[[7,7],[0,0]])
```

```
[2 2]
```

**woah**

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

```
[30]: print(y[[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]
```

```
[31]: # Image(picturename)
```