ARCTIC Summer Workshop 2023

Day 2 – Morning Session(9am – 10:30am)

Numpy

- Numpy is used for data analytics. Supports a wide variety of mathematical computations such as linear algebra, Fourier transform.
- To install via pip on Mac or Linux, first upgrade pip to the latest version:
  - python -m pip install --upgrade pip
- Install the Numpy stack packages with pip. Developers recommend a user install, using the --user flag to pip (note: don’t use sudo pip, that will give problems). This installs packages for your local user, and does not need extra permissions to write to the system directories:
  - pip install --user numpy

NumPy (Array Creation)

```
import numpy as np

arr1 = np.empty(3)
arr1
arr1 = np.empty([2,3])
arr1
```

```
array([[6.90857411e-310, 6.90857411e-310, 0.00000000e+000],
       [0.00000000e+000, 0.00000000e+000, 0.00000000e+000]])
```

```
# Creating an array of 3 rows and 4 columns with all one val

arr_ones = np.ones((3,4))
arr_ones
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

- Refer:
  - https://numpy.org/doc/stable/reference/routines.array-creation.html
  - https://numpy.org/doc/stable/user/basics.creation.html#arrays-creation

NumPy (Array Manipulation)

```
# First we create an array of size 6

arr = np.arange(6)
arr
```

```
array([0, 1, 2, 3, 4, 5])
```

```
arr.reshape(3,2)
```

```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

NumPy (Array Manipulation)

```
# Let's create a 2 array of size 3X4.

arr_3d = np.arange(24).reshape(2,3,4)
arr_3d
```

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

```
arr = arr_3d.flatten()
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23])
```

NumPy(Array Operation)

```
a = np.array([1,2,3,4])
b = np.array([1.2,5,6.9,7])

print(a+b)
```

```
 2.2  7.   9.9 11. ]
```

```
print(a%3)
```

```
[1 2 0 1]
```

```
print(a<3)
```

```
[ True  True False False]
```

NumPy(Array Operation)

```
# 2D array example

A = np.array( [[1,1],[0,1]] )
B = np.array( [[3,4],[5,6]] )
```

```
A*B
```

```
array([[3, 4],
       [0, 6]])
```

```
A.dot(B)
```

```
array([[ 8, 10],
       [ 5,  6]])
```

NumPy(Array Functions)

```
# Let's use numpy array functions on 2d array

arr_2d = np.arange(12).reshape(3,4)
arr_2d
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
# sum of all elememnts along columns
print(arr_2d.sum(axis=0))
print(arr_2d.min(axis=0))
print(arr_2d.max(axis=0))
```

```
[12 15 18 21]
[0 1 2 3]
[ 8  9 10 11]
```

```
# sum of all elements along rows
print(arr_2d.sum(axis=1))
print(arr_2d.min(axis=1))
print(arr_2d.max(axis=1))
```

```
[ 6 22 38]
[0 4 8]
[ 3  7 11]
```

NumPy(Indexing and Slicing of Arrays)

- Efficient way to access a subset of elements from array

```
a = np.arange(11)
print(a)
a = a**2
print(a)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10]
[  0   1   4   9  16  25  36  49  64  81 100]
```

```
# indexing
print(a[2])
print(a[-2])

# slicing array
print(a[2:7])  # excludes 7th index element and  elements after those
print(a[2:-2]) # exclude (11-2=9th) index element and elements after those
print(a[2:])
print(a[:7])
```

```
4
81
[ 4  9 16 25 36]
[ 4  9 16 25 36 49 64]
[  4   9  16  25  36  49  64  81 100]
[ 0  1  4  9 16 25 36]
```

Numpy(Iterating Over Arrays)

```
students = np.array([['John','Alice','Bob','Sam'],
                     [69,89,12,56],
                     [34,87,90,23]])
```

```
for i in students:
    print (i)
```

```
['John' 'Alice' 'Bob' 'Sam']
['69' '89' '12' '56']
['34' '87' '90' '23']
```

Pandas

- Pandas provide a high level data structure with convenient functions to read and parse data. It is built on top of numpy and can interact with multiple file formats.
- Two Data Types
  - Series - One dimensional labeled array. Can hold data of any type.
  - Dataframe - a two dimensional labeled datastructure that can hold data of any type.
- Refer: https://pandas.pydata.org/docs/user_guide/10min.html

Pandas

```
arr = np.random.rand(3)
```

```
arr
```

```
array([0.19053665, 0.65730881, 0.08112387])
```

```
arr_series = pd.Series(arr)
```

Now we have a series object, which you can easily display in a notbook by typing its name:

```
arr_series
```

```
0    0.190537
1    0.657309
2    0.081124
dtype: float64
```

```
arr_dataframe = pd.DataFrame(arr)
```

```
arr_dataframe
```

	0
0	0.190537
1	0.657309
2	0.081124

Pandas(dtypes,index,columns)

df.dtypes refers to the datatypes of all fields in the dataframe:

```
arr_dataframe.dtypes
```

```
0    float64
dtype: object
```

df.index refers to the labels/rows of each datapoint:

```
arr_dataframe.index
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
arr_dataframe.columns
```

```
RangeIndex(start=0, stop=1, step=1)
```

Pandas(Reading Files)

- Reading CSV file
  - pd.read_csv('path/to/csv/file')
- Reading JSON file
  - pd.read_json('path/to/json/file')
- Refer: https://pandas.pydata.org/docs/user_guide/io.html

Pandas Vs Numpy

- Pandas is mostly used for data analysis tasks whereas NumPy is mostly used for working with numerical values.
- Pandas perform better when number of rows of dataset is more than 500K, whereas for less than 50K rows NumPy performs better.
- Pandas works well for heterogeneous(numerics, alphabets) types of dataset whereas numpy works better with only numerical data
- Pandas is used mostly in tabular(CSV, Excel) data whereas numpy is used in array-like data(arrays, matrix)

Matplotlib

- Matplotlib is the most common library for visualizing data.
- Hierarchy : Figure-Axes-Axis-Ticks
- Advantage
  - Easy to see the property of the data
  - Can plot anything
- Disadvantage
  - It may be complex to plot non-basic plots or adjust the plots to look nice.
- Installing matplotlib
  - pip install matplotlib
- Includes multiple modules - matplotlib.pyplot, pylab, Object level APIs(Matplotlib APIs)

Matplotlib

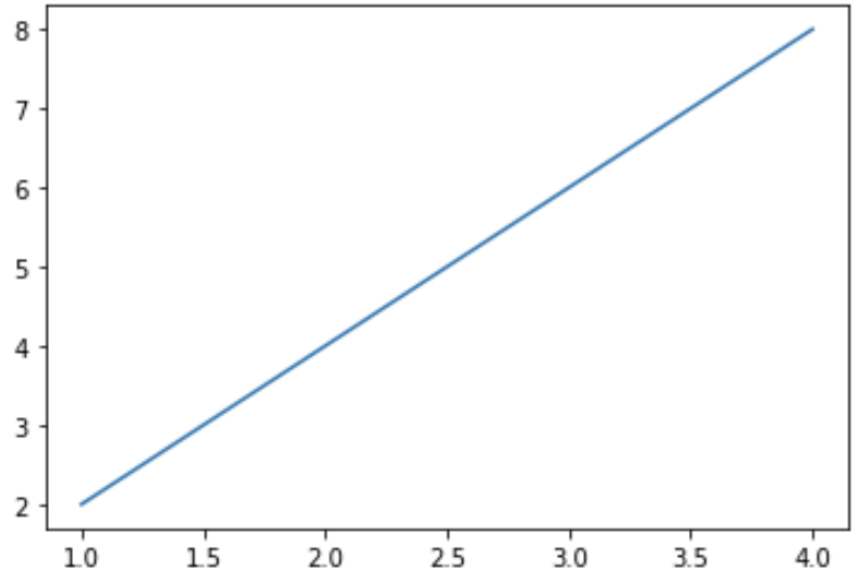
```
import matplotlib
import matplotlib.pyplot as plt
```

Let's plot a list of values as x and y axis. a is taken as x-axis and b is taken as y-axis

```
a = [1,2,3,4]
print(type(a))
b = [2,4,6,8]
print(type(b))
```

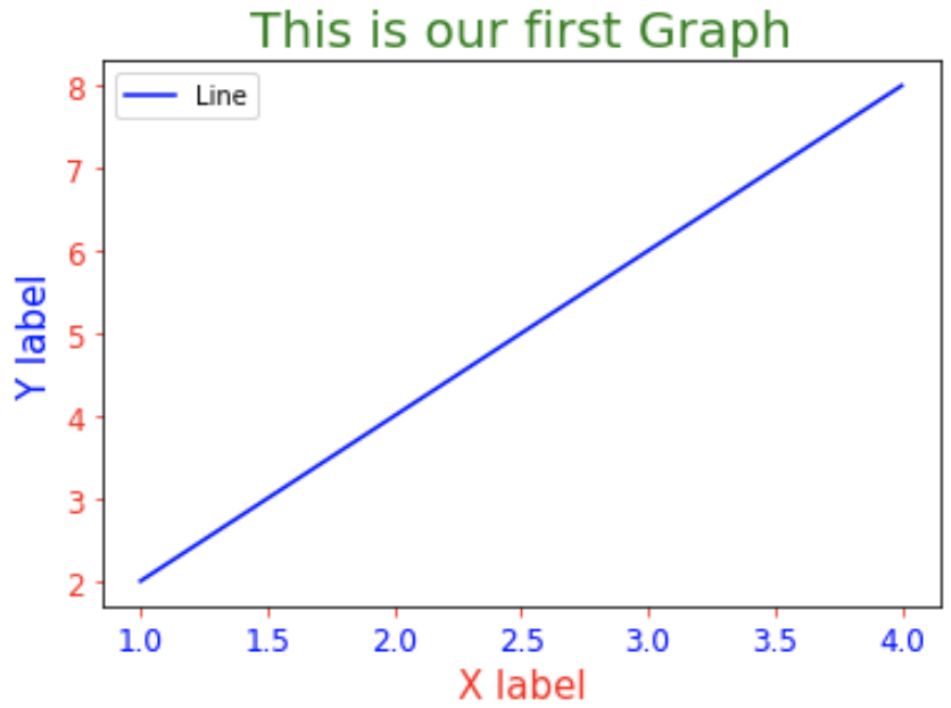
```
<class 'list'>
<class 'list'>
```

```
plt.plot(a,b)
plt.show()
```



Matplotlib

```
plt.title("This is our first Graph", {'fontsize':20,'color':'green'})
plt.plot(a,b,c='blue',label='Line')
plt.legend()
plt.xlabel('X label',fontsize =15, c ='red')
plt.ylabel('Y label',fontsize =15, c ='Blue')
plt.tick_params(axis='x',color='red',labelsize='large',labelcolor='blue')
plt.tick_params(axis='y',color='red',labelsize='large',labelcolor='red')
```



Seaborn

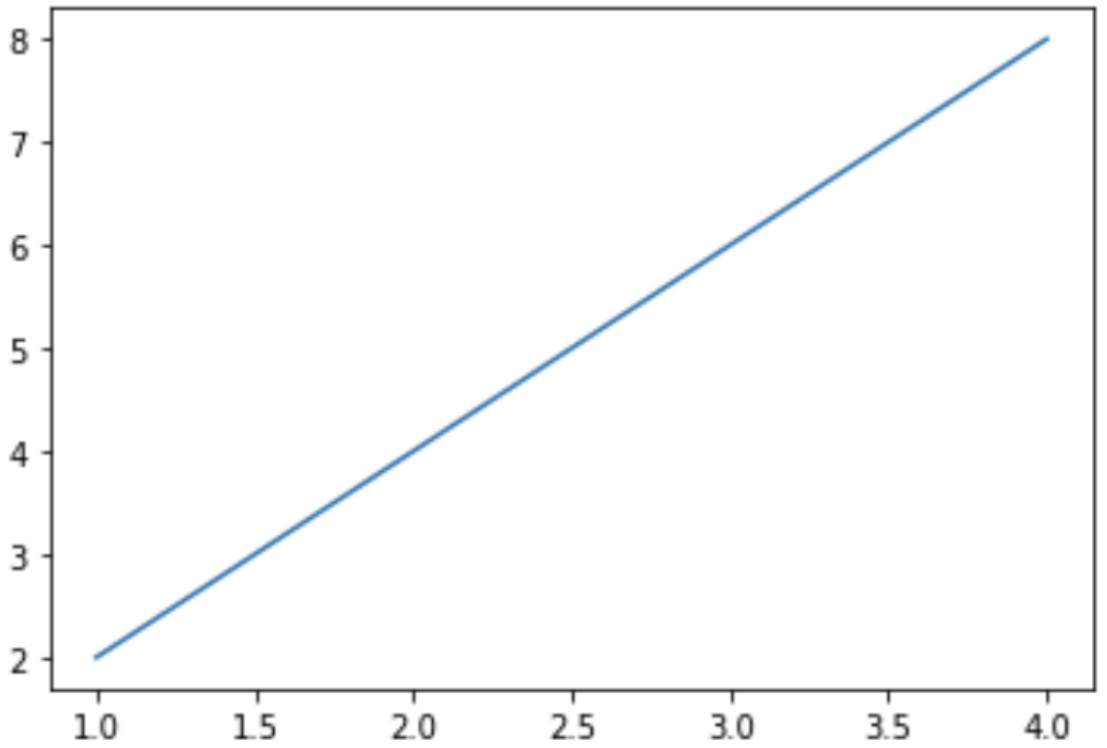
- Library based on Matplotlib
- Wrapper of matplotlib
- Advantages
  - Less code
  - Make common-used plots prettier
- Disadvantage
  - Does not have as wide a collection as matplotlib
- For dealing with categorical data and statistical data
- Refer: https://seaborn.pydata.org/tutorial/introduction.html

Seaborn

```
import numpy as np
import seaborn as sns
a = [1,2,3,4]
b = [2,4,6,8]
x =[2,4,6,8]
y = np.sqrt(x)
```

```
sns.lineplot(x=a,y=b)
```

```
<AxesSubplot:>
```



Seaborn

1.4 Adding Title

```
import seaborn as sns

p = sns.lineplot(x=a,y=b, marker='o')
p.set_xlabel("X-Axis",{'fontsize':20,'c': 'red'})
p.set_ylabel("Y-axis",{'fontsize':20,'c': 'blue'})
p.set_title("Graph")
```

```
Text(0.5, 1.0, 'Graph')
```