ASSIGNMENT 1

**NumPy**

-> Also called as numerical python.

->A library in python specifically to deal with arrays and array operations.

->We need to import numPy package before using it using statement

**import numpy as np** (#you can use any othr name instead of np)

To create an array in 1-d,

**mylist\_1 = [1,2,3,4]**

**array1d = np.array(mylist\_1)**

To create an array in 2-d,

**mylist\_2 = [5,6,7,8]**

**array2d = np.array([mylist\_1,mylist\_2])**

To print it, use statement

**print(array1d)**

**print(array2d)**

We can also create array with only ones, zeros, or even empty array using

np.ones(n), np.zeros(n), np.empty(n) functions

where n is the size of array

for creating identity matrix, use np.eye(n) where n is size of matrix

We can also do scalar operations on numPy arrays, n which each value is operated by its corresponding value in other matrix

Multiplication,exponential multipication,subtraction,division are done using operator ‘\*’ , ‘\*\*’ , ‘-’ , ‘/’

**arr = np.arange(0,12)** will give us an array of 12 real numbers.

**Arr[0:5] = 20** Will change all array elements from index 0 to index 4 and change it to 20.

->One important thing in numPy arrays is **slicing**

**arr1[:] = 29** will select whole rray and change it into 29.

**arr1 = arr[0:6]**

**arr1[:] = 29**

All element are modified

arr is also modified even though we changed its slice i.e. arr1.

this is bcoz no extra memory is being allocated for arr1. It is just a view of arr.

whatever change you make in arr1 will be changed in its original array

Done for memory optimization purposes

creating new array copy which does not point to same mem location

**arrcopy = arr.copy()**

**Indexing in NumPy Arrays**

**print(arr2d[0])** This will print the 0th row

print(arr2d[0][0]) This will print 0th row and 0th column intersection, i.e. top left cell of array

**slices of 2d arrays**

**slice1 = arr2d[0:1,0:2]**

0:1 means 0 included and 1 not included

So 0:2 means 0 and 1 included and 2 excluded

arr2d[0:1,0:2] means show 0th and 1st column of 0th row

print(slice1)

**slice2 = arr2d[:2,1:]** from 0 till 2 and from till end we have to consider arr2d

**print(slice2)**

slice2 = 15 you just cannot give 15 to slice2 like this

**arr2d[:2,1:] = 15**

**print(arr2d)**

Using loops to index

**arrlen = arr2d.shape[0]**

**for i in range(arrlen):**

**arr2d[i] = i**

**print(arr2d)** if you first insert tab and then write print statement, the the statement is considered in for loop

One more way of accessing the rows

**print(arr2d[[0,1]])**

**print(arr2d[[1,0]])** #first column becomes second and vice versa

Universal array functions include **arange,sqrt,exp,random,addition,maximum**

**# Saving single arrays**

arr = np.arange(10)

print(arr)

#The RAM is limited. At a particular instant of time,we may not need full array in the main memory(RAM).

#So an alternative is to store thearray in the hard drive and retrieve it whenver required.

#And delete variable when we dont want it

np.save(**'saved\_array'**,arr)

#New file is created - saved\_array.npy in .npy format

#np.save take two arguments

#argument 1 is the name of .npy file

#argument 2 is the array which we want to store in the file

new\_array = np.load(**'saved\_array.npy'**)

#np.load will load the .npy file array which we created

print(new\_array)

#Saving multiple arrays

#Multiple numpy arrays can be stored in a zip file

array\_1 = np.arange(25)

array\_2 = np.arange(30)

#np.savez('saved\_archive.npz',array\_1,array\_2) This is wrong

np.savez(**'saved\_archive.npz'**,x = array\_1,y = array\_2)

# np.savez is used to store multiple array

# npz is he file format which we use for savez function

load\_archive = np.load(**'saved\_archive.npz'**)

#load function is same for both single and multiple arrays

print(**"load\_archive of x is"**)

print(load\_archive[**'x'**])

print(**"load\_archive of y is"**)

print(load\_archive[**'y'**])

#We cannot see the .npy or .npz file in the system. And also, there is no text editor to read these files.

#For that, we have measure to convert them into text file

#Save to txtfile

np.savetxt(**'notepadfile.txt'**,array\_1,delimiter=**','**)

#savetxt function takes three arguments

#arg1 is name of the file

#arg2 is the array which we want to store

#arg3 is the delimiter

#delimter is the character by which we want to separate the array in txt file

# loading of txt files

load\_txt\_file = np.loadtxt(**'notepadfile.txt'**,delimiter=**','**)

#loadtxt function has two argumnts

#arg1 is name of file

#arg2 is delimiter which computer should know so that it will decode the array

print(**"load\_txt\_file is"**)

print(load\_txt\_file)

# After printing the array using loadtxt function, its is howing float values instead of integer values because

# when we use loadtxt function, it cob=nverts integer to float datatype

numPy also has capability to deal with functions,

one such function used is meshgrid

axes\_values = np.arange(-100,100,10) #x coordinates on graph

dx, dy = np.meshgrid(axes\_values,axes\_values)

# Meshgrid groups the values with itsef again and again

# like cartesian cross product

**PANDAS**

Pandas is a library that specifically deals with arrays of customised indexes, like series and dataframes, etc.

To create series, use

ser1 = Series([1,2,3,4])

Creating series using different ethods and their operations

#Use numpy arrays to create series

data\_array = np.array([**'a'**,**'b'**,**'c'**,**'d'**])

s = Series(data\_array)

print(s)

#Using custom indexes

s1 = Series(data\_array,index = [101,102,103,104])

print(s1)

#Using String in indexes

s2 = Series(data\_array,index= [**'yash'**,**'shubhi'**,**'shailu'**,**'uma'**])

print(s2)

#Using real life examples

revenue = Series([20,80,40,35],index = [**'ola'**,**'uber'**,**'grab'**,**'gojek'**])

#Accessing data from custom indexes

print(revenue[**'ola'**])

#Accessing data after applying some conditions

print(revenue[revenue >= 35])

#Use boolean conditions

print(**'ola' in** revenue) #returns true if 'ola' is present in revenue

print(**'lyft' in** revenue) # will return false as 'lyft' is not present in revenue

# Converting series into dictionary

revenue\_dict = revenue.to\_dict()

print(revenue\_dict)

# nan values

index\_2 = [**'ola'**,**'uber'**,**'grab'**,**'gojek'**,**'lyft'**]

revenue2 = Series(revenue,index\_2)

print(revenue2)

#To check if null values are present in data or not, use isnull

print(pd.isnull(revenue2))

#ola False

#uber False

#grab False

#gojek False

#lyft True

print(pd.notnull(revenue2))

#ola True

#uber True

#grab True

#gojek True

#lyft False

#Addition of series (+)

print(revenue + revenue2) #The result is obtained in ascending order of alphabetical order of indexes

# Assigning names

revenue2.name = **"Company Revenues"** #Acts as title for data

revenue2.index.name = **"Company Name"**

print(revenue2)

Dataframes are series with more than one columns

revenue\_df = pd.read\_clipboard()

#pd.read\_clipboard() is a functionality of pandas wheren whatever dataframe we copied in

the cliboard is stored in the variable, in this case revenue\_df

print(revenue\_df)

#indexes and columns in dataframe

print(revenue\_df.columns)

print(revenue\_df[**'1 '**])

#Accessing multiple columns

print(DataFrame(revenue\_df,columns=[**'1 '**, **'Walmart '**, **'Retail '**]))

#NaN values

revenue\_df2 = DataFrame(revenue\_df,columns=[**'1 '**, **'Walmart '**, **'Retail '**,**'profit '**])

#Here, column profit is not present. But it willl not throw an error. Rather, it will

create a new column with NaN values

print(**"Revenue\_df2 is\n"**)

print(revenue\_df2)

#head and tail function

print(revenue\_df2.head(2))

# .head(n) prints the first n rows of dataframe

print(**"\n"**)

print(revenue\_df2.tail(2))

# .tail(n) prints the last n rows of dataframe

#Acessing rows in DataFrame, we use ix[index] function

print(revenue\_df.ix[0])

print(revenue\_df.ix[4])

#Assigning values to datafrae

#Using numpy

array1 = np.array([1,2,3,4,5])

revenue\_df2[**'profit '**] = array1

print(revenue\_df2)

#Using series

profits = Series([900,1000],index=[3,4])

revenue\_df2[**'profit '**] = profits

print(revenue\_df2)

#Deletion -- Once we delete the data, we will not get it back

**del** revenue\_df2[**'profit '**]

print(revenue\_df2)

#Dictionary function to dataframe

sample = {

**'compay'**:[**'A'**,**'B'**],

**'Profit'**:[1000,5000],

}

sample\_df = DataFrame(sample)

print(sample)

print(**"Datafreme is \n"**)

print(sample\_df)

#Creating new indexes using reindex

series2 = series1.reindex([**'e'**,**'f'**,**'g'**,**'h'**,**'i'**,**'j'**])

print(series2)

#Using fillvalue

series2 = series2.reindex([**'e'**,**'f'**,**'g'**,**'h'**,**'i'**,**'j'**,**'k'**],fill\_value=10)

#The value assigned to new created index in fill\_value

print(series2)

# Using reindex methods => ffill

cars = Series([**'Audi'**,**'Merc'**,**'BMW'**],index = [0,4,8])

print(cars)

ranger = range(13)

print(ranger)

cars = cars.reindex(ranger,method=**'ffill'**)

print(cars)

#Creating new dataframe using randn

df\_1 = DataFrame(randn(25).reshape(5,5),index = [**'a'**,**'b'**,**'c'**,**'d'**,**'e'**],columns = [**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**])

print(df\_1)

df\_2 = df\_1.reindex([**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**])

print(df\_2)

df\_3 = df\_2.reindex(columns = [**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**,**'c6'**])

print(df\_3)

#using ix[] method to reindex

#Only applicable for python 2.7.x

#df\_4 = df\_1.ix[['a','b','c','d','e','f'],['c1','c2','c3','c4','c5','c6']]

#print(df\_4)

Sorting and Ranking

ser1 = Series([500,1000,1500],index=[**'a'**,**'c'**,**'b'**])

#sorting by index

print(ser1.sort\_index())

#sort by values

print(ser1.sort\_values())

ser2 = Series(randn(10))

print(ser2)

#ranking of series

ser2 = ser2.sort\_values()

print(ser2.rank())