**DATA ANALYSIS AND VISUALIZATION (ASSIGNMENT : 1)**

**NAME: A J HARAVINDAN**

**IDE: PyCharm**

**Python Interpretor: Python (version 2.7.16)**

**SECTION 2: NUMPY**

Numpy is a library in python used for applying mathematical functions on arrays (single and multi-dimensional arrays).To use numpy library in python we have to select python interpretor from settings and install numpy library to it.

While writing the code using numpy, the first step is to import numpy to the code.

FUCTIONS USED:

np.arange( : : ): This fuction prints the value from starting index to the ending with the difference that is mentioned.

**import** numpy **as** np  
array1= np.arange(10)  
**print** array1

[0 1 2 3 4 5 6 7 8 9]

**Scalar operations on numpy**: Several operations like ‘-‘, ‘+’, ‘\*’, ‘\*\*’ (subtraction, addition, multiplication,raised to power) respectively were performed on the arrays using numpy.

**Indexing numpy arrays**: One interesting thing to note here is that copy of and array is not being done and change in one array will effect other arrays associated with it.

To overcome this we use a function arrcopy=arr.copy( ). This creates separate memory for and arrcopy.

Code:

**import** numpy **as** np  
arr = np.arange(0,12)  
**print** arr  
arr2 = arr[0:6]  
arr2[:] = 29  
**print** arr2  
**print** arr

Output:

[ 0 1 2 3 4 5 6 7 8 9 10 11]

[29 29 29 29 29 29]

[29 29 29 29 29 29 6 7 8 9 10 11]

Indexing in Two dimensional array: Two dimensional array can be formed using numpy arrays. Two dimensional arrays can also be formed using the concept of loops.

Code:

**import** numpy **as** np  
  
arr2d = np.array([[1,2,3],[4,5,6],[7,8,9]])  
**print** arr2d  
arr2d[:2,1:] = 15 //value of 15 is stored in index (0,1),(1,last)  
**print** arr2d  
arr\_len = arr2d.shape[0] //the no. of columns in row0 is found with shape function.  
**for** i **in** range(arr\_len):  
 arr2d[i] = i;  
**print** arr2d;

Output

[[1 2 3]

[4 5 6]

[7 8 9]]

[[ 1 15 15]

[ 4 15 15]

[ 7 8 9]]

[[0 0 0]

[1 1 1]

[2 2 2]]

**Premium array operation**:

In this we perform various operations like sqrt, exp, add, maximum.

Code:

**import** numpy **as** np  
  
A = np.arange(1,15,2)  
**print** A  
B = np.sqrt(A) //square root of array A is stored in B **print** B  
  
C = np.exp(A) //exponential power of array A is stored in c **print** C  
  
D = np.add(A,B) //Two arrays A and B are added and the result is stored in D **print** D  
  
E = np.maximum(A,B)//The maximum value corresponding to each index of A and B is stored in E **print** E

Output:

[ 1 3 5 7 9 11 13]

[1. 1.73205081 2.23606798 2.64575131 3. 3.31662479

3.60555128]

[2.71828183e+00 2.00855369e+01 1.48413159e+02 1.09663316e+03

8.10308393e+03 5.98741417e+04 4.42413392e+05]

[ 2. 4.73205081 7.23606798 9.64575131 12. 14.31662479

16.60555128]

[ 1. 3. 5. 7. 9. 11. 13.]

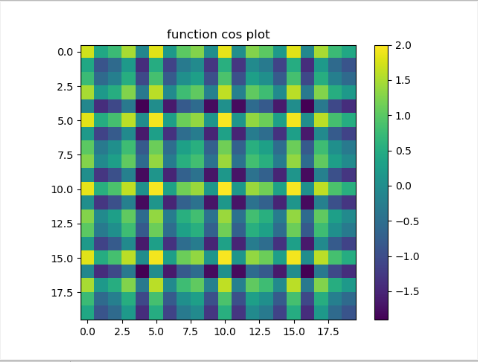
**Statistical Processing and array graphical sketches:**

The data needs to be analysed to come to a conclusion. So data representation is important part. To represent data in graphical format we have functions in python.

import matplotlib.pyplot as plt //This is the plotting library to plot the fuction.

Code

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
axes\_values = np.arange(-100,100,10)//values from -100 to 100 with a interval of 10  
dx, dy = np.meshgrid(axes\_values,axes\_values) //creates a rectangular grid out of an array of axes values and an array of axes values.  
function1 = np.cos(dx)+np.cos(dy) //a function is defined  
**print** function1  
plt.imshow(function1) //opens the graph plotted with the values  
plt.title(**"function cos plot"**)//  
plt.colorbar()  
plt.savefig(**'myfig2.png'**)



**Condition Clauses and Boolean Expressions**:

Here we operate on conditons and Boolean expression on arrays.

np.where( ) function is used //np.where(#condition, #value for yes, #value for No)

Standard functions on numpy:

print x.sum() //prints the sum of all element in the array x

print x.mean() //prints the mean of all element in the array x

print x.std() //prints the standard deviation of all the element in the array x

print x.var() //prints the variance of all the elements in the array x

Logical operations:

condition = np.array([True,True,False,False])

print condition.any() //or operator o/p=True

print condition.all() //and operator o/p=False

Sorting in arrays:

**import** numpy **as** np  
unsorted\_array = np.array([1,2,6,12,4,67]) //unsorted array  
unsorted\_array.sort() //sorting operation is done  
**print** unsorted\_array //the sorted array is printed

**SECTION 3:** **DATA MANIPULATION WITH PANDAS:**

Pandas is a library in python which is used for analyting the data and to perform mathematical operation on data sets. To use pandas library in python we have to select python interpretor from settings and install pandas library to it.

1.Series : Series is a one dimensional array with index value capable of holding any datatype. It is nothing but like a column in a excel sheet.

2.DataFrame: Dataframe is a two dimensional data structure. The data is arranged in rows and columns manner. It consists of row, column, and data.

Code(Series):

**import** pandas **as** pd //importing panda library as pd  
**from** pandas **import** Series //from panda library series is imported  
**import** numpy **as** np //numpy library is imported  
x = Series([5,10,15,20]) //a series is created with data   
**print** x //the series is printed

Output:

0 5

1 10

2 15

3 20

dtype: int64

In the above code we can see that the data is printed in tabular manner with index.

Here the index is automatically generated but it also can be mentioned by the coder.

Code:

**import** pandas **as** pd  
**from** pandas **import** Series  
**import** numpy **as** np  
data\_array = np.array([**'a'**,**'b'**,**'c'**])  
s = Series(data\_array,index=[100,101,102])  
**print** s

Output:

100 a

101 b

102 c

dtype: object

Code(DataFrame):

**import** pandas **as** pd //importing panda library  
**from** pandas **import** DataFrame //dataframe is imported from panda library  
data = [[**'Haravindan'**,20],[**'Akshay'**,21],[**'Omkar'**,22]] //creation of a list  
df = pd.DataFrame(data,columns=[**'Name'**,**'Age'**]) //using the function dataframe a table is formed.  
**print** df

Output:

Name Age

0 Haravindan 20

1 Akshay 21

2 Omkar 22

**Indexing**: In pandas direct indexing is not possible. It gives an error. This is done to avoid many problems if the data sets are huge. Any change in the index can lead to big problem and confusion. An example is shown below.

Code:

**import** pandas **as** pd  
**from** pandas **import** Series  
series1 = Series([10,20,30,40],index=[**'a'**,**'b'**,**'c'**,**'d'**]) //creating a series with data and index  
**print** series1  
index1 = series1.index  
**print** index1  
index1[0] = **'e' //First index value ‘a’ is being changed to ‘e’ but gives an error  
print** index1

Error:

Traceback (most recent call last):

File "C:/Users/Haravindan/PycharmProjects/Example1/Sample.py", line 7, in <module>

index1[0] = 'e'

File "C:\Python27\lib\site-packages\pandas\core\indexes\base.py", line 3938, in \_\_setitem\_\_

raise TypeError("Index does not support mutable operations")

TypeError: Index does not support mutable operations

**Re-indexing:**

Re-indexing can be done using the functions .reindex( ) or .ix[ ]

.ix[ ] functions does the reindexing simultaneously for both row and column whereas .reindex() does one at a time.

Code:

**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
**import** numpy **as** np  
**from** numpy.random **import** randn  
  
df\_1 = DataFrame(randn(25).reshape(5,5),index=[**'a'**,**'b'**,**'c'**,**'d'**,**'e'**], columns=[**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**]) //creating a random data of 5\*5 DataFrame and indexing them  
**print** df\_1  
df\_2 = df\_1.ix[[**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**],[**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**,**'c6'**]] //reindexing using the function .ix[ ]  
**print** df\_2

**Droping enteries from datatypes:**

In python we can drop a set of data using a condition and by using a function .drop( )

A row , column, or the whole dataframe can be dropped. But the function should be used carefully since data might get lost in this process.

Code:

**import** pandas **as** pd  
**import** numpy **as** np  
**from** pandas **import** Series,DataFrame  
  
names\_df = DataFrame(np.arange(9).reshape(3,3),index=[**'AJ'**,**'Akshay'**,**'Adi'**],columns=[**'X'**,**'Y'**,**'Z'**]) //a dataframe is created of 3\*3 with index specified  
**print** names\_df  
names\_df = names\_df.drop(**'AJ'**,axis=0) //the row containing AJ is droped  
**print** names\_df  
  
names\_df = names\_df.drop(**'Y'**,axis=1) //the column containing Y is droped  
**print** names\_df

Output:

X Y Z

AJ 0 1 2

Akshay 3 4 5

Adi 6 7 8

X Y Z

Akshay 3 4 5

Adi 6 7 8

X Z

Akshay 3 5

Adi 6 8

**Handling Null Values:**

The null values can be handled by the function .dropna( )

Code:

**import** numpy **as** np  
**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
df1 = DataFrame([[1,2,3],[5,6,np.nan],[7,np.nan,10],[np.nan,np.nan,np.nan]])  
**print** df1  
**print** df1.dropna() //drop all rows with null values

**print** df1.dropna(how=**'all'**) //drop the rows which has only null values

Output:

0 1 2 //1st print statement

0 1.0 2.0 3.0

1 5.0 6.0 NaN

2 7.0 NaN 10.0

3 NaN NaN NaN

0 1 2 //2nd print statement

0 1.0 2.0 3.0

0 1 2 //3rd print statement

0 1.0 2.0 3.0

1 5.0 6.0 NaN

2 7.0 NaN 10.0

Code:

df2 = DataFrame([[1,2,3,np.nan],[4,5,6,7],[8,9,np.nan,np.nan],[12,np.nan,np.nan,np.nan]])  
**print** df2  
**print** df2.dropna(thresh=3) //drops only those rows which has more than or equal to 3 null values in a row

Output:

0 1 2 3 //1st print statement

0 1 2.0 3.0 NaN

1 4 5.0 6.0 7.0

2 8 9.0 NaN NaN

3 12 NaN NaN NaN

0 1 2 3 //2nd print statement

0 1 2.0 3.0 NaN

1 4 5.0 6.0 7.0

**Data Alignment:** Let us suppose that there are 4 elements in a dataframe(2\*2) and another dataframe which has 9 elements (3\*3). We want to perform operation like addition and subtraction on the dataframes. But the problem is both the dataframe are not of same size. In this case we use (fill\_value=0) which will take the remaining unknown values of the smaller dataframe as zero and do the operation needed.

Code:

**import** numpy **as** np  
**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
df1 = DataFrame(np.arange(4).reshape(2,2),columns=[**'a'**,**'b'**],index=[**'car'**,**'bike'**])  
**print** df1  
df2 = DataFrame(np.arange(9).reshape(3,3),columns=[**'a'**,**'b'**,**'c'**],index=[**'car'**,**'bike'**,**'cycle'**])  
**print** df2  
**print** df1+df2  
  
df1 = df1.add(df2,fill\_value=0)  
**print** df1

Output:

a b

car 0 1

bike 2 3

a b c

car 0 1 2

bike 3 4 5

cycle 6 7 8

a b c

bike 5.0 7.0 NaN

car 0.0 2.0 NaN

cycle NaN NaN NaN

a b c

bike 5.0 7.0 5.0

car 0.0 2.0 2.0

cycle 6.0 7.0 8.0

**Ranking and sorting of arrays:**

The sorting of an array is done using the function .sort( )

Rank is found using the function .rank( )

Code:

**import** numpy **as** np  
**import** pandas **as** pd  
**from** pandas **import** Series  
ser =Series([5000,200,1500],index=[**'a'**,**'c'**,**'b'**]) ///Series of data is created with index  
**print** ser //the series is printed  
**print** ser.sort\_index() //the sorting is done with respect to the index values(a,b,c)  
**print** ser.sort\_values() //the sorting is done with respect to the values  
**print** ser.rank() //the rank is found wrt the values

Output:

a 5000

c 200

b 1500

dtype: int64

a 5000

b 1500

c 200

dtype: int64

c 200

b 1500

a 5000

dtype: int64

a 3.0

c 1.0

b 2.0

dtype: float64

**Statistics and Graphical sketches with pandas:**

As we know that in data analytics there is a huge amount of data. So graphical representations can make the job easy to analyse and work on it.

As discussed above to work with graphs we need to use the matplotlib.pyplot library.

Code:

**from** pandas **import** Series,DataFrame  
**import** numpy **as** np  
**from** numpy.random **import** randn  
**import** matplotlib.pyplot **as** plt  
  
array1 = np.array([[10,np.nan,20],[30,40,np.nan]])  
**print** array1  
df1 = DataFrame(array1,index=[1,2],columns=list(**'ABC'**)) //a dataframe is created with array1 and index and columns specified  
**print** df1 //df1 is printed**print** df1.sum() //sum along each column**print** df1.sum(axis=1) //sum along index**print** df1.min() //min values of df1  
**print** df1.max() //max value of df1  
  
**print** df1.idxmax()  
**print** df1.cumsum() //cumulative sum is printed  
**print** df1.describe()  
df2 = DataFrame(randn(9).reshape(3,3),index=[1,2,3],columns=list(**'ABC'**))  
**print** df2 //using random in numpy the data is filled,index and column are specified  
  
plt.plot(df2) //the df2 is plotted  
plt.legend(df2.columns,loc=**"lower right"**) //colour is specified  
plt.savefig(**'samplepic.png'**) //the png file is saved  
plt.show() ///the png file is opened

Output:

[[10. nan 20.]

[30. 40. nan]]

A B C

1 10.0 NaN 20.0

2 30.0 40.0 NaN

A 40.0

B 40.0

C 20.0

dtype: float64

1 30.0

2 70.0

dtype: float64

A 10.0

B 40.0

C 20.0

dtype: float64

A 30.0

B 40.0

C 20.0

dtype: float64

A 2

B 2

C 1

dtype: int64

A B C

1 10.0 NaN 20.0

2 40.0 40.0 NaN

A B C

count 2.000000 1.0 1.0

mean 20.000000 40.0 20.0

std 14.142136 NaN NaN

min 10.000000 40.0 20.0

25% 15.000000 40.0 20.0

50% 20.000000 40.0 20.0

75% 25.000000 40.0 20.0

max 30.000000 40.0 20.0

A B C

1 -1.846576 -1.798658 2.159837

2 -0.135526 1.382144 1.059793

3 -1.042596 0.701935 -0.245994

