**Assignment – I**

**Name: Akshaya Balaji**

**Email:** [**akshayavb99@gmail.com**](mailto:akshayavb99@gmail.com)

**Question: Study the Complete Numpy and Pandas Lectures (Section 2 and 3 of intern-kit) and make a documentation in less than 500 words (Word limit is excluding the codes typed.) in a word document. A documentation can have codes, explanations and logical flowcharts.**

**Introduction to Numpy**

NumPy is the fundamental package for scientific computing with Python. It contains a powerful N-dimensional array object, sophisticated (broadcasting) functions, useful linear algebra, Fourier transform, and random number capabilities. NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

In Python, we can import numpy with an alias for ease of usage as shown below.

**import numpy as np**

All the basic operations using numpy and matplotlib have been executed using Pycharm Community Edition with Python 3.7. The code executed for practice is within one file, but has been split into parts here for the documentation

1. **Creation of arrays:**

**import** numpy **as** np  
  
*#Create sample list*mylist1=[1,2,3,4]  
mylist2=[5,6,7,8]  
  
*#Create array from sample list*myarray1=np.array(mylist1)  
print(myarray1)  
  
*#Create multidimensional (2D) array*myarray2=np.array([mylist1,mylist2])  
print(myarray2)

1. **Array dimension and datatype**

*#Find dimension of array*print(myarray2.shape)  
  
*#Finding the datatype of array elements*print(myarray2.dtype)

1. **Special array functions**
2. zeros(): Creates a new numpy array of dimension (x,y). If only 1 value is mentioned as parameter x, then dimension of array is (1,x)

print(np.zeros(4))

1. ones(): Creates a new numpy array of dimension (x,y). If only 1 value is mentioned as parameter x, then dimension of array is (1,x)

print(np.ones(5))

1. empty(): Similar to zeros function. We get output as numpy array with random values close to zero

print(np.empty(3))

1. eye(): Creates an identity matrix of order n. Order is a parameter for the function

print(np.eye(3))

1. arange(): It is an arithmetic progression function taking 3 parameters (start, stop, diff). start is the 1st term; stop is final term (not included in the final AP array); diff is the diff between consecutive terms of AP

print(np.arange(1,5,2))

1. **Scalar operations on arrays**
2. Array Multiplication

print(myarray1\*myarray1) *#This gives element wise multiplication and the array is considered as scalar*

1. Exponential Multiplication

*#Exponential multiplication*print(myarray1\*\*3)

1. Subtraction and Reciprocal  
     
   *#Subtraction in array is scalar in nature*print(myarray2-myarray2)  
     
   *#Reciprocal operation on arrays gives reciprocal of each element*print(1/myarray1)
2. **Indexing and copying array contents**
3. Indexing one dimensional arrays

arr = np.arange(0,12)  
print(arr) *#Print whole array*print(arr[0]) *#Print element at 0th index of array*print(arr[0:5]) *#Print array elements from index 0 to index 4 (both inclusive)*arr[0:5]=20 *#Sets the values of arr from index 0 to 4 (both inclusive) to 20*print(arr)

1. Copying one dimensional arrays using copy()

Typically array slicing does not create a new copy of the part of the array. Rather, the new variable storing the array slice will still make changes in the original array itself. To create a completely new copy, we use the copy() function.

*#To create a copy of the same array in numpy using copy() function*arr\_copy=arr.copy()

1. Indexing multidimensional arrays

*#Indexing Numpy Arrays (Multidimensional Arrays)*arr2d=np.array([[1,2,3],[4,5,6],[7,8,9]])  
print(arr2d)  
  
*#Slices of 2D array*slice1=arr2d[0:1,0:2] *#This slice stores row index 0(1st row) and column indices 0,1(1st 2 columns)  
#print(slice1)*slice2=arr2d[:2,1:]  
*#print(slice2)  
#Other 1D indexing operations and setting values are allowed for multidimensional arrays as well***for** i **in** range(len(arr2d)):  
 arr2d[i]=i  
*#print(arr2d) #If we use single index to access the elements of a rD array, then the index is considered as the row index and  
# refers to all the elements of that row  
  
#print(arr2d[[0,1]]) #Another method of accessing array rows using row numbers*

1. **Other array functions**
2. arange()

A=np.arange(1,15,2)  
print(A)

1. sqrt() – returns an array of same dimensions as calling array with each corresponding calling array element’s square root

B=np.sqrt(A)  
print(B)

1. exp() – returns an array of same dimensions as calling array, where each element is e to the power of the corresponding calling array element

C=np.exp(A)  
print(C)

1. add() – It takes 2 arrays as arguments and returns the summation of their corresponding elements. Both input arrays must have same dimensions.

print(np.add(A,B))

1. maximum() – It compares the corresponding elements of the 2 input arrays and returns the maximum of the elements at each location

print(np.maximum(A,B))

1. sum() – It returns the row-wise sum of the calling array. To find the column-wise sum, we can pass the argument 0 while calling the function.

print(x.sum()) *#This is used for row-wise sum  
  
#To test columnwise sum*n=np.array([[1,2],[3,4]])  
print(n.sum(0))

1. mean() – It returns the mean of the elements of the array column-wise.

print(x.mean())

1. std() – It returns the standard deviation of the array elements column-wise.

print(x.std())

1. var() – It returns the variance of the array elements column-wise.

print(x.var())

1. any() – It is used for performing logical operations. It essentially performs the function of the ‘or’ operator on the calling array elements and returns true if any one or more of the array elements is true.

condition2=np.array([**True**,**False**,**True**])

print(condition2.any())

1. all() – It is used for performing logical operations. It performs the function of the ‘and’ operator for the calling array elements and returns true only if all array elements are true.

condition2=np.array([**True**,**False**,**True**])

print (condition2.all())

1. sort() – It returns the sorted form of the calling array

unsorted\_arr=np.array([1,2,5,4,7])  
unsorted\_arr.sort()  
print(unsorted\_arr)

1. unique() – It returns an array containing unique elements of the calling array and removes element redundancy.

arr2=np.array([**'solid'**,**'liquid'**,**'gas'**,**'solid'**,**'liquid'**,**'gas'**])

1. in1d() – It returns a Boolean array of dimension of the first array argument, where every element of array1 is checked if it occurs in the second array argument.

print(np.in1d([**'solid'**,**'liquid'**,**'plasma'**],arr2))

1. **Conditional clauses and Boolean operations using arrays**

x=np.array([100,400,500,600]) *#Each member 'a'*y=np.array([10,15,20,25]) *#Each member 'b'*condition=np.array([**True**,**True**,**False**,**False**]) *#Each member ‘cond’  
  
#Use loops indirectly*z=[a **if** cond **else** b **for** a,cond,b **in** zip(x,condition,y)]  
*#print(z)  
  
#Above operation can be performed with numpy as given below  
#np.where(condition,value for yes,value for no)*z2=np.where(condition,x,y)  
print(z2)  
  
z3=np.where(x>0,0,1)  
print(z3)

1. **Saving single and multidimensional arrays**
2. Saving a single array

A single array can be saved with the extension .npy using the save() function.

arr=np.arange(10)  
*#print(arr)*np.save(**'saved\_arr.npy'**,arr) *#A new file called saved\_arr is created with the Numpy format and this stores the array arr*new\_array=np.load(**'saved\_arr.npy'**) *#load is used to load an existing Numpy file*print(new\_array)

1. Saving multiple arrays

Multiple arrays are saved in archive files or zip files using the savez() function.

*#Multiple arrays are stored as a zip file or archive file*array1=np.arange(25)  
array2=np.arange(30)  
np.savez(**'saved\_arrays.npz'**,x=array1,y=array2)  
new\_arrayz=np.load(**'saved\_arrays.npz'**)  
print(new\_arrayz[**'x'**])

1. Saving to a text file(.txt extension)

Arrays can be stored in a text file (.txt extension) using the savetxt() function. We can also specify the delimiter for the array contents in the text file.

np.savetxt(**'textfile\_array.txt'**,array1,delimiter=**','**)  
text\_load=np.loadtxt(**'textfile\_array.txt'**,delimiter=**','**)  
print(text\_load)

1. **Using numpy and matplotlib libraries for graphing data**

**import** matplotlib.pyplot **as** plt *#For graphing data*axes\_val=np.arange(-100,100,10)  
dx,dy=np.meshgrid(axes\_val,axes\_val) *#Maps values in pairs*

function = 2\*dx+3\*dy *#New array called function contains values derived from values of dx and dy. Has same dimensions as dx and dy*

*#Plotting data values*plt.imshow(function)  
plt.title(**'Graph of 2\*dx+3\*dy'**)  
plt.colorbar()  
plt.savefig(**'my\_fig.png'**)

In the python code given above, we first import the required library resources needed for plotting graphs from matplotlib. We then initialise a variable axes\_val to store a range of values from -100 to 100, with a step of 10.

The function meshgrid() from the numpy maps the values for the x and y coordinates for the function defined as 2x+3y as given.

The function imshow() is used to show the graphical plot of the function as color coded values. The attributes like title() and colorbar() are used to bring clarity to the graph contents. The graph plotted can also be saved as an image using the savefig() function.

**Introduction to Pandas**

Pandasis an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.Pandas is a NumFOCUS sponsored project.

In Python, we can import pandas with an alias for ease of usage as shown below.

**import pandas as pd**

All the basic operations using pandas have been executed using Pycharm Community Edition with Python 3.7. The code executed for practice is within one file, but has been split into parts here for the documentation

1. **Series in Pandas**

To work with the Series functionality of the Pandas library, we use the following statement:

**from** pandas **import** Series

1. Some of the common operations using the Series are given below. Series are modifiable.

obj=Series([5,10,15,20])

1. We use the values attribute to print the values stored in Series and the index attribute to print the indices of the values of the Series.

print(obj.values)

print(obj.index)

1. We can use numpy arrays to create series by passing the numpy array as an argument to the Series function.

data\_arr=np.array([**'a'**,**'b'**,**'c'**])  
arr\_obj=Series(data\_arr)  
print(arr\_obj)

1. We can also set custom index values for the values in a Series object. The indices can be numerical, character or string values.

s=Series(data\_arr,index=[100,101,102])  
print(s)

s=Series(data\_arr,index=[**'index1'**,**'index2'**,**'index3'**])  
print(s)

1. Given below is an example of the operations performed with strings

*#The revenue Series object stores the revenues of the companies given as index values in billions*revenue=Series([20,80,40,35],index=[**'Ola'**,**'Uber'**,**'Grab'**,**'Gojek'**])  
print(revenue)  
print(**"Revenue of Ola: "**+str(revenue[**'Ola'**]))  
  
*#(i)Revenue of companies > 35*print(**"Revenue of companies >= 35: "**)  
print(str(revenue[revenue>=35]))  
  
*#(ii) Using Boolean Conditions in Series*print(**"Is lyft in the data?: "**+str(**'lyft' in** revenue))  
  
*#(iii) Conversion of Series into dictionary*revenue\_dict=revenue.to\_dict() *#Converts the series into a dictionary with key-value pairs where key=index*print(revenue\_dict)  
  
*#(iv) Handling NaN values (important for situations where data is missing during data analysis*index\_2=[**'Ola'**,**'Uber'**,**'Grab'**,**'Gojek'**,**'Lyft'**]  
revenue2=Series(revenue,index\_2)  
print(revenue2) *#Revenue value of 'Lyft' index is automatically taken as NaN since no value was defined*print(pd.isnull(revenue2)) *#Only the value corresponding to index 'Lyft' is True due to the NaN revenue value*  
*#(v) Opposite function of pd.isnull(Series obj)*print(pd.notnull(revenue2))  
print()  
  
*#5. Addition of series*print(revenue+revenue2) *#Here the final result is printed in the ascending order of the index values (Here, alphabetically)*  
*#6. Assigning names*revenue2.name=**"Company Revenues"** *#Essentially a title for the Series*revenue2.index.name=**"Company Names"** *#Title for index values*print(revenue2)

1. **DataFrames in Pandas**

A dataframe is a matrix-like object with rows and columns. The rows are defined by index names and the columns are defined by column names.

revenue\_df=pd.read\_clipboard() *#This automatically loads the data last copied into revenue\_df  
  
#For the current table values being copied from Wikipedia there is an error that shows up with the command above.  
#To go past this, save the required rows as a CSV file and use the read\_csv() function with the seperator as '\t'*revenue\_df = pd.read\_clipboard()  
print(revenue\_df)

1. **Indexing in Pandas**

Consider the series defined as series1.

series1=Series([10,20,30,40],index=[**'a'**,**'b'**,**'c'**,**'d'**])

The following are the properties and operations performed on this variable:

1. Index property of Series

index1=series1.index  
print(index1[2])  
print(index1[2:])

1. Negative indices

print(index1[-1:]) *#This displays only the last element from the array index1. This is essentially indexing the array in reverse*

1. Properties of DataFrame Indices

We cannot assign a new value to the indices of a DataFrame, unlike the modifiability of the array indices. Attempting to do so throws a TypeError.

*#index1[0]='e'*print(index1)

1. **Reindexing Methods**

Consider a Series object named as series1 as follows.

The following are the operations and properties related to reindexing methods in dataframes:

1. Creating new indices using reindex() function

series1=Series([1,2,3,4],index=[**'e'**,**'f'**,**'g'**,**'h'**])  
series2=series1.reindex([**'e'**,**'f'**,**'g'**,**'h'**,**'i'**,**'j'**])  
print(series2)  
  
series2=series2.reindex([**'e'**,**'f'**,**'g'**,**'h'**,**'i'**,**'j'**,**'k'**],fill\_value=10)  
print(series2)

1. Reindexing using ffill()

cars=Series([**'Audi'**,**'Merc'**,**'BMW'**],index=[0,4,8])  
ranger=range(13) *#Creates a list of elements from 0 to 12 (both inclusive)*cars=cars.reindex(ranger,method=**"ffill"**)  
  
*#ffill ensures that indices from 0 to 3 has value Audi, 4 to 7 have value Merc and so on.*print(cars)

1. Reindexing methods for rows and columns

df\_1=DataFrame(np.random.randn(25).reshape(5,5),index=[**'a'**,**'b'**,**'c'**,**'d'**,**'e'**],columns=[**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**])  
print(df\_1)  
print()  
  
df\_2=df\_1.reindex([**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**]) *#Reindexing Rows*print(df\_2)  
print()  
  
df\_3=df\_1.reindex(columns=[**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**,**'c6'**])  
print(df\_3)

1. **Dropping entries in Pandas**

Consider the Series object cars as defined below.

cars=Series([**'BMW'**,**'Audi'**,**'Merc'**],index=[**'a'**,**'b'**,**'c'**])

The following operations are then demonstrated on the variable cars:

1. Removal of values using drop() function – Here, the drop() function is called by the Series object and the argument passed to it is the index value from which the Series element is to be removed. It returns the calling Series object with the required element removed from the specified index.

cars=cars.drop(**'a'**)

1. Dropping row values and column values

cars\_df=DataFrame(np.random.randn(9).reshape(3,3),index=[**'BMW'**,**'Audi'**,**'Merc'**],columns=[**'rev'**,**'pro'**,**'exp'**])  
print(cars\_df) *#Current values in the matrix of the dataframe are random values*print()  
  
*#Dropping row values*cars\_df=cars\_df.drop(**'BMW'**,axis=0)*#Drops the index 'BMW' (The row given as 'BMW')  
  
#Dropping column values. To perform operations on the columns, we need to explicitly set the axis as 1*cars\_df=cars\_df.drop(**'pro'**,axis=1)

1. **Handling null data in Pandas**

The null value can be represented as a constant of the numpy library as numpy.nan. Such cases of null data are handled in different ways using pandas and numpy. Consider the Series object series1 as shown below.

series1=Series([**'A'**,**'B'**,**'C'**,**'D'**,np.nan]) *#np.nan is essentially null*

The following operations are all performed on this series object.

1. isnull() – The function evaluates the Series element-wise and returns true for those elements of the calling Series object which have value equal to NaN.

print(series1.isnull())

1. dropna() – This functions returns the modified calling Series object after removing the NaN values from it.

print(series1.dropna())

1. DataFrame operations

df1=DataFrame([[1,2,3],[5,6,np.nan],[7,np.nan,10],[np.nan,np.nan,np.nan]])  
*#When we use the dropna() function, the whole row is dropped even if there is one row element=NaN.  
#To overcome this problem, we can set the how property within the dropna() function as 'all'*print(df1.dropna(how=**'all'**))  
  
*#Column wise dropping null values*print(df1.dropna(axis=1)) *#The dropna() works similar in row and column NaN value drops*print()  
  
*#Threshold property of dropna() [thresh]*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.dropna(thresh=3)) *#This drops all rows where number of data values (not equal to NaN) is lesser than 3  
  
#3. Filling NaN values with chosen numerical values - fillna() function*print(df2.fillna({0:0,1:50,2:100,3:200}))

1. **Selecting and Modifying data in Pandas**

Consider the Series object series1 as given below.

The following operations are all performed on the variable series1:

series1=Series([100,200,300],index=[**'a'**,**'b'**,**'c'**])  
*#The Series always have in-built indices of 0,1,2.... even if we explicitly mention other indices. Hence we can access  
#the series elements by the in-built indices as well  
  
#Conditional Indexing*print(series1[series1>150])  
print(series1[series1==300])  
  
df1=DataFrame(np.arange(9).reshape(3,3),index=[**'car'**,**'bike'**,**'cycle'**],columns=[**'A'**,**'B'**,**'C'**])  
print(df1[[**'A'**,**'B'**]]) *#Printing Multiple Columns*print(df1>5) *# Returns a matrix of dimensions of df1 where each element is a boolean value whose is result is the condition mentioned*

1. **Regulating Data**

Consider the Series object ser\_a and ser\_b as shown below for the operations performed.

*#Data Alignment and Regulation*ser\_a=Series([100,200,300],index=[**'a'**,**'b'**,**'c'**])  
ser\_b=Series([300,400,500,600],index=[**'a'**,**'b'**,**'c'**,**'d'**])  
  
*#Sum of series = ser\_a+ser\_b*print(ser\_a+ser\_b)  
  
*#Dataframes*df1=DataFrame(np.arange(4).reshape(2,2),columns=[**'a'**,**'b'**],index=[**'car'**,**'bike'**])  
df2=DataFrame(np.arange(9).reshape(3,3),columns=[**'a'**,**'b'**,**'c'**],index=[**'car'**,**'bike'**,**'cycle'**])  
print(df1+df2) *#All the elements of 3 index and 3rd column are not available in df1 hence the NaN value after summation  
#To bypass the NaN, use the add function with a fill\_value of 0 for all undefined rows/columns*print(df1.add(df2,fill\_value=0))

1. **Sorting and Ranking**

Consider the Series object ser1 as shown below for the following operations.

ser1=Series([500,1000,1500],index=[**'a'**,**'c'**,**'b'**])  
*#Sorting by index using sort\_index()*print(ser1.sort\_index())  
  
*#Sorting by values using sort\_values()*print(ser1.sort\_values())  
  
*#rank() function - Prints the ranking value after they are sorted in ascending order*print(ser1.rank())  
  
*#Ranking of Series (Basis for Sorting)*ser2=Series(np.random.randn(4))  
print(ser2.rank())

1. **Statistics and Graph Sketches**

Consider the numpy array arr1 and the dataframe df1 as shown for the following operations.

arr1=np.array([[10,np.nan,20],[30,40,np.nan]])  
df1=DataFrame(arr1,index=[1,2],columns=list(**'ABC'**))  
  
*#1. sum() - performs summation along each column. Set axis=1 for summation along indexes(rows) NaN is considered as 0*print(df1.sum())  
print(df1.sum(axis=1))  
print()  
  
*#2. min()/max() - returns minimum/maximum values along the columns.*print(df1.max())  
print()  
  
*#3. idxmax()/idxmin() - returns the index of the maximum/minimum values along the columns*print(df1.idxmax())  
print()  
  
*#4. cumsum() -returns cumulative sum along the columns.NaN is considered as 0*print(df1.cumsum())  
print()  
  
*#5. describe() - Shows the overall stats table for the given data column-wise*print(df1.describe())  
print()  
  
*#Graphs and Statistics*df2=DataFrame(np.random.randn(9).reshape(3,3),index=[1,2,3],columns=list(**'ABC'**))  
  
*#Plotting values column-wise against indices*plt.plot(df2)  
plt.legend(df2.columns,loc=**'lower right'**)  
plt.savefig(**'df2\_plot1.png'**)  
*#plt.show()*print()  
  
*#unique() - returns the unique values in the Series*ser1=Series(list(**'abcccaabd'**))  
print(ser1.unique())  
  
*#value\_counts() - returns the frequency of element occurence in series in descending order*print(ser1.value\_counts())