

# Takenmind Assignment #1

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.

## Numpy Package

For scientific computing that provides n dimensional array class, tools and useful math operations e.g. linear algebra, Fourier transform... their objects are used on scipy, pandas and other packages.

**numpy.array[<list>]** <- Create array multidimensional from hardcoded data.

**<object>.shape** <- Dimensionality

**.dtype** <- data structure type.

**.min()** <- Mini.value

**.max()** <- Max.value

**.copy()** <- Generate a new data space variable.0

Method utils:

**numpy.zeros(<dimensions>)**

**.ones(<dimensions>)**

**.empty(<dimensions>)** <- Undefined Array

**.eye(<dimensions>)** <-Identity matrix. Diagonal.

Scalation and new version compatibility:  $5/2 = 2.5$ !!!-> **from \_\_future\_\_ import division** ->  $5/2 = 2.5$

Array Operations:

Assigination

Add/Substract

Multiplication

Slicing [:] and slice assigination, with tied memory space between identifier's data. Select rows, cols, dimensions, elements.

Copying arrays.

Loops along arrays. Utils:

`arr_rows = arr2d.shape [ 0 ] -> len(arr2d.shape)[ 0 ]`

`arr_cols = arr2d.shape [ 1 ] -> len(arr2d.shape)[ 0 ]`

Example for area access: `arr2d[1:,1:]`

### Universal numpy functions

**numpy.arange**( <start>, <end>, <step> )

**.add** ( <arrayA>, <arrayB>, ... )

**.sqrt**( <array> )

**.maximum/minimum**( <arrayA>, <arrayB> )

**.sum**( ) -> Sum of element values.

**.mean**( )

**.std**( )

**.var**( )

**.sort**( )

Other functions associated to numpy arrays on scripy.org

### Save/load large data to/from numpy arrays

Save single array: **np.save**( '<file\_array>', <array> )

Load single array: **np.load**( '<file\_array>.npz' )

Save several arrays: **np.savez**( '<file\_arrays>', x = <arrayA>, y = <arrayB>,...)

or **np.savez\_compressed**( '<file\_arrays>', x = <arrayA>, y = <arrayB>,...)

Load several arrays: **arrays = np.load**( '<file\_arrays>.npz' ) ; **array['x']** and **array['y']** to fetch.

Save to text file: **np.savetxt**( '<txt\_array>', <array>, delimiter = {, ; : .} )

Load from text file: **np.loadtxt**( '<txt\_array>', delimiter = {, ; : .} ) **Note:** converts integer to float.

### Conditional clause and boolean operations with numpy

Conditions and loops enclosed by [ ]

**np.where**( <condition>, <valueYes>, <valueNo> )

Logic arrays True/False Values: **.all**( ) is AND, **.any**( ) is OR.

Inclusion: **numpy.in1d**( <array>, <array\_numpy> ) returns contained values.

## Matplotlib/pyplot package

Matplotlib is an object oriented plotting library, generally applied to mathematical/statistical plot graphs.

- Define axes values with numpy array
- Examples with two variable functions: **np.meshgrid** and **np.cos**
- Define function values; print values.
- Plot values: **matplotlib.pyplot.imshow**( <function> )
- Add title; **matplotlib.pyplot.title**( '<title>' )
- Add scale bar: **matplotlib.pyplot.colorbar**()
- Save: **matplotlib.pyplot.savefig**( '<image\_file>' )

## Panda's to analyze data

Panda's library is specific to analyze data and perform math operations on datasets.

Panda's provides powerful structures for data analytics, time series, and statistics about.

Add package pandas: Settings ->Project Interpreter -> + -> Add 'pandas'. Example applications:

```
import pandas as pd
```

```
import numpy as np
```

```
from pandas import Series.
```

```
object = Series([ 5, 10, 15, 20 ])
```

```
print object -> index + data
```

```
object.index -> [ 0,1,2,3 ] = RangeIndex ( start=0, stop = 4, step = 1 )
```

```
object.values -> [ 5, 10, 15, 20 ]
```

- Numpy arrays to series

```
data_array = np.array ( [ 'a', 'b', 'c', 'd' ] )
```

```
s = Series (data_array)
```

```
object.index -> [ 0,1,2,3 ] = RangeIndex ( start=0, stop = 4, step = 1 )
```

```
object.values -> [ 'a', 'b', 'c', 'd' ]
```

- Custom index on series

```
s = Series( data_array, index = [ 'id1', 'id2', 'id3', 'id4' ] )
```

- Using real life example

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

```
revenue[ 'uber' ] -> 80
```

Boolean conditions:

```
revenue[ revenue >= 35 ] -> gojek, grab, uber
```

```
'ola' in revenue -> True
```

Convert to dictionary: **revenue\_dict = revenue.to\_dict**

NaN values:

```
index2 = [ 'ola', 'uber', 'grab', 'gojek', 'lyft' ]
```

```
revenue2 = Series( [ revenue, index2] ) -> lyft - NaN
```

```
pd.isnull( revenue2 ) -> True for NaN
```

```
pd.notnull( revenue2 ) -> not True for NaN
```

- Addition of series

```
add_revenues = revenue + revenue2 -> sum of values for each index class.
```

- Assigning names

```
revenue2.name = 'Co. revenues'
```

```
revenue2.index.name = 'Co. name'
```

- Dataframes (pandas.pydata.org). Is a functionality to analyze data, simulate a matrix with rows and cols along an index of rows created in addition.

**import numpy as np**

**import pandas as pd**

**from pandas import Series, Dataframe**

Exercise: from wikipedia find list of largest companies by revenue:

([https://en.wikipedia.org/wiki/List\\_of\\_largest\\_companies\\_by\\_revenue](https://en.wikipedia.org/wiki/List_of_largest_companies_by_revenue))

copy first six rows of list, included head. After load with pandas:

**revenue\_df = pd.read\_clipboard( )**

**print revenue\_df**

# index and columns

**print revenue\_df.columns**

**print revenue\_df ['Rank']** <- see data of column associated by label 'Rank'

**print Dataframe( revenue\_df = [ <array of label columns included> ]**

- Example to create a new Dataframe object

**new\_df = Dataframe (revenue\_df, columns = ['Rank', 'Name',...]**

# NaN values. Create new label 'Profit' without data at new Dataframe.

**DataFrame\_df2 = (revenue\_df, columns=['Rank', 'Name', 'Profit',...])** <-New column profit with NaN.

**print revenue\_df2**

# head and tail: first and last rows.

**revenue\_df.head( 2)** -> index 0, 1 ; first two rows.

**revenue\_df.tail( 2)** -> index (n-1), n ; last two rows.

# access rows in Dataframe

**revenue\_df.ix[ 0]** -> first row

# assign values to Dataframe. Two methods: from numpy, or from Series.

**array1 = np.array([1, 2, 3, 4, 5, 6])** -> Create numpy array.

**revenue\_df2 [ 'Profit' ] = array1** -> Assign values from numpy array,

-----

**profits = Series([ 900, 1000 ], index = [3, 5] )** -> Create one pandas series

**revenue\_df2 [ 'Profit' ] = profits** -> all values NaN except index 3 and index 5 that was assigned.

# Deletion of columns at Dataframe object

**del revenue\_df2 [ 'Profit' ]** -> Erase 'Profit' column.

- Dictionary functions to Dataframe:

**sample = { 'Company': [ A, B ], 'Profit': [ 1000, 5000 ] }**

**sample\_df = Dataframe ( sample )** -> enables Dataframe with indexes 0, 1; and Profit, Company data.

- Index objects

Performing index operations, as a set of series or Dataframes. Index as a label, but returns an array with positional 'u' labels. Then that indexes can be called by position.

```
series = Series([ 10, 20, 30, 40 ], index = list( 'abcd' ))
```

```
index1 = series.index -> index1[ 2] is 'c'
```

```
# negative indexes: index1[ -2: ] last two elements 'c', 'd' ; index1[ :-2 ] first two elements 'a', 'b'
```

```
# range of indexes: index[ 2:4] -> 'c', 'd'
```

Note: Indexes can't be modified, there are not mutable data.

- Reindexing methods

How can be reindex indexes and columns on Series and DataFrames.

```
from numpy.random import randn
```

```
series1 = Series( [ 1, 2, 3, 4 ], index = list( 'efgh' ))
```

```
# Creating new series with reindex.
```

```
series1 = series1.reindex( list( 'efghi' )) -> 'i' index doesn't have defined value (NaN).
```

```
# Using fill_value: series2 = series1.reindex( list( 'efghijk' ), fill_value = 10) -> 'j', 'k' values are ten.
```

```
# Using reindex methods: ffill
```

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

```
ranger = range(13) -> [ 0, 1, 2, ..., 11, 12 ]
```

```
cars.reindex( ranger, method = "ffill" ) -> Forward fill -> 0-3 is 'Audi', 4-7 is 'Merc', 8-12 is 'BMW'
```

```
# Create new dataframe using randn:
```

```
df1 = DataFrame (randn(25).reshape(5,5), index = list('abcde'), columns = [ 'c1', 'c2', 'c3', 'c4', c5])
```

```
# create new row f index: df2 = df1.reindex(list('abcdef'))
```

```
# create new col c6 index: df3 = df2.reindex(columns = ['c1', 'c2', 'c3', 'c4', 'c5', 'c6'])
```

```
# use .ix[] to reindex rows/colsw with one sentence
```

```
df4 = df1( list('abcdef' ), ['c1', 'c2', 'c3', 'c4', 'c5', 'c6'])
```

- Dropping entries from datatypes

```
#Series
```

```
cars = series(['BMW', 'Audi', 'Merc'], index = list('abc'))
```

```
cars = cars.drop('a') -> removes 'a' row.
```

```
# DataFrame
```

```
cars_df = DataFrame(np.arange(9).reshape(3,3), index =(['BMW', 'Audi', 'Merc'], index = ['a', 'b', 'c']),  
                    columns = ['revenue', 'profit', 'expenses'])
```

```
cars_df = cars.df.drop( 'BMW' ) -> Optional default parameter axis=0. Removes row with index 'BMW'
```

```
cars_df = cars.df.drop('profit', axis = 1 ) -> remove profit column.
```

- Handling null data: create series or dataframes with np.nan = NaN

**series1 = Series([ 'A', 'B', 'C', 'D', nan ])**

**series1.isnull()** -> True if NaN: F, F, F, F. T

**series1.dropna()** -> Optional parameter axis: 0= rows, 1= columns. Remove rows/columns that contains NaN

**df1.dropna()** -> The same for DataFrame.. parameters how = "all", thresh = n min. number of values to not remove.

Example1 : **df1 = DataFrame( [ 1, 2, 3 ], [ 5, 6, nan ], [ 8, nan, 10 ], [ nan, nan, nan, nan ] )**

df1	0	1	2
0	1.0	2.0	3.0
1	5.0	6.0	NaN
2	8.0	NaN	10.0
3	NaN	NaN	NaN

**df4 = df1.dropna( axis = 1)**

-> Remove all columns with NaN

-> **Empty DataFrame, Index ( 1, 2, 3, 4 )**

**df2 = df1.dropna()** -> Remove all rows with NaN

df2	0	1	2
0	1.0	2.0	3.0

**df3 = df1.dropna( how='all' )** -> Remove rows with all NaN

df3	0	1	2
0	1.0	2.0	3.0
1	5.0	6.0	NaN
2	8.0	NaN	10.0

Example2: **df1 = DataFrame( [ 0, 1, 2, nan ], [ 4, 5, 6, 7 ], [ 8, 9, nan, nan ], [12, nan, nan, nan ] )**

df1	0	1	2	3
0	0	1.0	2.0	NaN
1	4.0	5.0	6.0	7.0
2	8.0	9.0	NaN	NaN
3	12.0	NaN	NaN	NaN

**df2 = df1.dropna( thresh=3)**

df2	0	1	2	3
0	0	1.0	2.0	NaN
1	4.0	5.0	6.0	7.0

**fillna [ 0 ]** fills NaN with zero values ; **fillna[{0:0, 1:50, 2:100, 3:200}]** <- for columns

- Selecting and modifying data: series(<labels>)

**sr1 = Series( [ 100, 200, 300 ], index=['A', 'B', 'C'])**

**sr1['A']** -> 100

**sr1[0]** -> 100

**sr1[ 0:2 ]** -> A 100  
B 200

**sr1[sr1>150]** -> B 200  
C 300

**sr1[sr1==300]** -> C 300

Conditional indexes: examples: **series1[series1>30]** or **series1[series1 == 300]**

- Using DataFrame df and accessing

**df > 5** -> True or false depends of this condition

**df.ix['bike']** -> column 'bike' or **df.ix[1]** -> column n°1

- Data alignment

Sum of series **ser\_a + ser\_b** -> NaN if not defined. Same for subtraction **ser\_a - ser\_b**

- Sorting/Ranking Series and Dataframes.

Sort by index: **ser1.sort\_index()**

Sort by values: **ser1.sort\_values()**

Rank: meet position rank of values **ser1.rank()**

- Statistics & graph sketches with pandas

**df.sum(axis = 0)** -> sums along each column (NaN is 0)

**df.sum(axis = 1)** -> sums along each row (NaN is 0)

**df.min()** **df.max()** -> minimum and maximum values

**df.idxmax()** index of row where value is max. Same for min.

**df.cumsum()** -> cumulative sum.

**df.describe()** -> statistical distribution values.

**ser1.unique** -> Number of unique values on Series.

**ser1.value\_counts** -> frequency of presentation for values.