**Panda Tutorial**

**Series**

A Series is a one-dimensional array-like object containing a sequence of values (of similar types to NumPy types) and an associated array of data labels, called its index.

The simplest Series is formed from only an array of data:

In [11]: obj = pd.Series([4, 7, -5, 3])

obj.index # gives RangeIndex(start, stop, step)

obj.values # returns array

We can create a series with each data point signified by a index:

obj2 = pd.Series([1,2,3], index=[‘q’,’w’,’e’])

It’s kind of dictionary and same as ‘dict’ data structure of python. That’s why we can pass a dictionary to it.

sdata = { ‘ohio’:780, ‘colarado’:1200,’tennessee’:900 }

obj3 = pd.Series(sdata)

The resultant object is in sorting order. To disable sorting you can pass explicit keys list as in

states = [ ‘tenessee’, ‘ohio’, ‘colarado’ ]

pd.isnull()/notnull() functions can be used to check if argument passed is NULL or not. pd.Series also has these instances.

You can associate name with Series object and index as in:

obj3.name =’Population’

obj3.index.name = ‘State’

Series.value\_counts() returns the number of repetitions of all values.

Series.isin(<list>) returns list of bool with True for the element of Series which is contained in the list.

Index.get\_indexer method, which gives you an index array

from an array of possibly non-distinct values into another array of distinct values

**DataFrame**

A DataFrame is a rectangular table of data and contains an ordered collections of columns each of which can have a different value(i.e. int, str or Boolean)

We can pass a dict with an index and a series associated with a key to DataFrame constructor. All series should be of equal length. Each index refers to a column

frame = pd.DataFrame(data, columns=[<sequence\_of\_columns>)

To retrieve the header – frame.columns

Rows can be retrieved either by name or by ‘loc’ attribute. e.g.

frame.loc[‘three’]

We can access each element using row and column position by frame.iat[I, j]

You can retrieved aggregated data for both rows and columns by passing rows and columns to loc in that order. e.g frame.loc[‘three’, [ ‘colorado’, ‘nevada’][<Boolean\_condition>]

You can deploy ‘loc’ functionality using ‘iloc’. It’s just that you have to pass index numbers instead of whole strings.

A single column can be assigned a common value as:

frame[‘<col>’] = ‘<value>’

Obj.index() returns the indices or keys used to refer to records.

The del method can then be used to remove this column as in: del frame2['eastern']

*Essentials of Panda:*

> **Reindexing**: When you want to rearrange data according to new index.

obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c']

In [93]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

In [94]: obj2

Out[94]:

a -5.3, b 7.2, c 3.6, d 4.5, e NaN

You might want to fill certain values while reindexing. This can be done by passing method as key parameter to reindex function as in:  
obj3.reindex(range(6), method='ffill')

You can pass either row, column or both to DataFrame constructor

frame = pd.DataFrame(np.arange(9).reshape((3, 3)), index=['a', 'c', 'd'], columns=['Ohio', 'Texas', 'California'])

frame2 = frame.reindex(['a', 'b', 'c', 'd'])

In [101]: frame2

Ohio Texas California

a 0.0 1.0 2.0

b NaN NaN NaN

c 3.0 4.0 5.0

d 6.0 7.0 8.0

> **Dropping a row/column:**

You can drop a row or column from a DataFrame using:

obj.drop(<label\_either\_col\_or\_row>, axis=1 or 0) # Here 0 refers to indices and 1 refers to columns. It returns the copy of frame with requisite changes. To make changes in the current frame itself pass ->

inplace=True to drop function.

You can filter by column value as:

data[data['three'] > 5] # Here, ‘three’ is column name

**> Addition**

Addition of data frames will be like addition of elements present in 2-D array with non-common element tagged as NaN which can be filled to ‘0’ if required

df1.add(df2, fill\_value=0)

Following arithmetic operators ara available: add()/mul()/sub()/div()/pow() with r substitute of each

When we subtract list from an array. The subtraction operator is applied to all rows in array. This phenomenon is called broadcasting.

If you want to subtract a series from 2-D array column-wise pass axis=’column’

Some NumPy methods can work on DataFrame too:

np.abs(frame) # Change sign of all –ve numbers in DataFrame

func = lambda x: x.max()-x.min()

frame.apply(func)

This will work per column. If you rather want to apply this function row-wise, pass axis=’column’

**> Sorting and Ranking:**

series.sort\_index() sorts the series by indices. To sort by columns and by descending order:

series.sort\_index(axis=1, ascending=False)

Simply sort a series by **.sort\_values()** function.

To sort a frame by a given column, use frame.sort\_values(by=’<col\_name>’) function

DataFrame.idxmax/idmin returns the index with upper/lower values