Pandas

Pandas contains high level data structures and manipulation tools to make data analysis fast and easy in Python.

In [2]:

```
import pandas as pd #I am importing pandas as pd
from pandas import Series, DataFrame # Series and Data Frame are two d
```

Series

Series is a one-dimensional array like object containing an array of data(any Numpy data type, and an associated array of data labels, called its index.

In [13]:

```
mjp= Series([5,4,3,2,1])# a simple series
print mjp        # A series is represented by index on the left and va
print mjp.values # similar to dictionary. ".values" command returns va
```

```
0    5
1    4
2    3
3    2
4    1
dtype: int64
[5 4 3 2 1]
```

In [14]:

```
print mjp.index # returns the index values of the series
```

```
Int64Index([0, 1, 2, 3, 4], dtype='int64')
```

In [27]:

```
jeeva = Series([5,4,3,2,1,-7,-29], index =['a','b','c','d','e','f','h'
print jeeva # try jeeva.index and jeeva.values
print jeeva['a'] # selecting a particular value from a Series, by usin
```

```
a     5
b     4
c     3
d     2
e     1
f    -7
h   -29
dtype: int64
5
```

In [28]:

```
jeeva['d'] = 9 # change the value of a particular element in series
print jeeva
jeeva[['a','b','c']] # select a group of values
```

```
a     5
b     4
c     3
d     9
e     1
f    -7
h   -29
dtype: int64
```

Out[28]:

```
a    5
b    4
c    3
dtype: int64
```

In [31]:

```
print jeeva[jeeva>0] # returns only the positive values
print jeeva *2 # multiplies 2 to each element of a series
```

```
a    5
b    4
c    3
d    9
e    1
dtype: int64
a    10
b     8
c     6
d    18
e     2
f   -14
h   -58
dtype: int64
```

In [34]:

```
import numpy as np
np.mean(jeeva) # you can apply numpy functions to a Series
```

Out[34]:

```
-2.0
```

In [37]:

```
print 'b' in jeeva # checks whether the index is present in Series or
print 'z' in jeeva
```

```
True
False
```

In [46]:

```
player_salary ={'Rooney': 50000, 'Messi': 75000, 'Ronaldo': 85000, 'Fab
new_player = Series(player_salary)# converting a dictionary to a serie
print new_player # the series has keys of a dictionary
```

```
Fabregas      40000
Messi         75000
Ronaldo       85000
Rooney        50000
Van persie    67000
dtype: int64
```

In [49]:

```
players =['Klose', 'Messi', 'Ronaldo', 'Van persie', 'Ballack']
player_1 =Series(player_salary, index= players)
print player_1 # I have changed the index of the Series. Since, no val
```

```
Klose           NaN
Messi         75000
Ronaldo       85000
Van persie    67000
Ballack         NaN
dtype: float64
```

In [53]:

```
pd.isnull(player_1)#checks for Null values in player_1, pd denotes a p
```

Out[53]:

```
Klose          True
Messi         False
Ronaldo       False
Van persie    False
Ballack        True
dtype: bool
```

In [52]:

```
pd.notnull(player_1)# Checks for null values that are not Null
```

Out[52]:

```
Klose         False
Messi          True
Ronaldo        True
Van persie     True
Ballack       False
dtype: bool
```

In [64]:

```
player_1.name ='Bundesliga players' # name for the Series
player_1.index.name='Player names' #name of the index
player_1
```

Out[64]:

```
Player names
Klose             NaN
Messi           75000
Ronaldo         85000
Van persie      67000
Ballack           NaN
Name: Bundesliga players, dtype: float64
```

In [67]:

```
player_1.index =['Neymar', 'Hulk', 'Pirlo', 'Buffon', 'Anderson'] # is
player_1
```

Out[67]:

```
Neymar        NaN
Hulk        75000
Pirlo       85000
Buffon      67000
Anderson      NaN
Name: Bundesliga players, dtype: float64
```

Data Frame

Data frame is a spread sheet like structure, containing ordered collection of columns. Each column can have different value type. Data frame has both row index and column index.

In [74]:

```
states ={'State' :['Gujarat', 'Tamil Nadu', ' Andhra', 'Karnataka', 'Ke
                  'Population': [36, 44, 67,89,34],
                  'Language' :['Gujarati', 'Tamil', 'Telugu', 'Kannada'
india = DataFrame(states) # creating a data frame
india
```

Out[74]:

	Language	Population	State
0	Gujarati	36	Gujarat
1	Tamil	44	Tamil Nadu
2	Telugu	67	Andhra
3	Kannada	89	Karnataka
4	Malayalam	34	Kerala

In [75]:

```
DataFrame(states, columns=['State', 'Language', 'Population']) # chang
```

Out[75]:

	State	Language	Population
0	Gujarat	Gujarati	36
1	Tamil Nadu	Tamil	44
2	Andhra	Telugu	67
3	Karnataka	Kannada	89
4	Kerala	Malayalam	34

In [82]:

```
new_farme = DataFrame(states, columns=['State', 'Language', 'Populatio
#if you pass a column that isnt in states, it will appear with Na valu
```

In [86]:

```
print new_farme.columns
print new_farme['State'] # retrieveing data like dictionary
```

```
Index([u'State', u'Language', u'Population', u'Per Capita Income'], dty
a       Gujarat
b    Tamil Nadu
c        Andhra
d     Karnataka
e        Kerala
Name: State, dtype: object
```

In [89]:

```
new_farme.Population # like Series
```

Out[89]:

```
a    36
b    44
c    67
d    89
e    34
Name: Population, dtype: int64
```

In [91]:

```
new_farme.ix[3] # rows can be retrieved using .ic function
# here I have retrieved 3rd row
```

Out[91]:

```
State                Karnataka
Language               Kannada
Population                  89
Per Capita Income          NaN
Name: d, dtype: object
```

In [94]:

```
 new_farme
```

Out[94]:

	State	Language	Population	Per Capita Income
a	Gujarat	Gujarati	36	NaN
b	Tamil Nadu	Tamil	44	NaN
c	Andhra	Telugu	67	NaN
d	Karnataka	Kannada	89	NaN
e	Kerala	Malayalam	34	NaN

In [97]:

```
new_farme['Per Capita Income'] = 99 # the empty per capita income colu
new_farme
```

Out[97]:

	State	Language	Population	Per Capita Income
a	Gujarat	Gujarati	36	99
b	Tamil Nadu	Tamil	44	99
c	Andhra	Telugu	67	99
d	Karnataka	Kannada	89	99
e	Kerala	Malayalam	34	99

In [99]:

```
new_farme['Per Capita Income'] = np.arange(5) # assigning a value to t
new_farme
```

Out[99]:

	State	Language	Population	Per Capita Income
a	Gujarat	Gujarati	36	0
b	Tamil Nadu	Tamil	44	1
c	Andhra	Telugu	67	2
d	Karnataka	Kannada	89	3
e	Kerala	Malayalam	34	4

In [104]:

```
series = Series([44,33,22], index =['b','c','d'])
new_farme['Per Capita Income'] = series
#when assigning list or arrays to a column, the values lenght should m
new_farme # again the missing values are displayed as NAN
```

Out[104]:

	State	Language	Population	Per Capita Income
a	Gujarat	Gujarati	36	NaN
b	Tamil Nadu	Tamil	44	44
c	Andhra	Telugu	67	33
d	Karnataka	Kannada	89	22
e	Kerala	Malayalam	34	NaN

In [119]:

```
new_farme['Development'] = new_farme.State == 'Gujarat'# assigning a ne
print new_farme
del new_farme['Development'] # will delete the column 'Development'
new_farme
```

```
        State   Language  Population  Per Capita Income Development
a     Gujarat   Gujarati          36                NaN        True
b  Tamil Nadu      Tamil          44                 44       False
c      Andhra     Telugu          67                 33       False
d   Karnataka    Kannada          89                 22       False
e      Kerala  Malayalam          34                NaN       False
```

Out[119]:

	State	Language	Population	Per Capita Income
a	Gujarat	Gujarati	36	NaN
b	Tamil Nadu	Tamil	44	44
c	Andhra	Telugu	67	33
d	Karnataka	Kannada	89	22
e	Kerala	Malayalam	34	NaN

In [16]:

```
new_data ={'Modi': {2010: 72, 2012: 78, 2014 : 98},'Rahul': {2010: 55,
elections = DataFrame(new_data)
print elections# the outer dict keys are columns and inner dict keys a
elections.T # transpose of a data frame
```

```
      Modi  Rahul
2010    72     55
2012    78     34
2014    98     22
```

Out[16]:

	2010	2012	2014
Modi	72	78	98
Rahul	55	34	22

In [17]:

```
DataFrame(new_data, index =[2012, 2014, 2016]) # you can assign index
```

Out[17]:

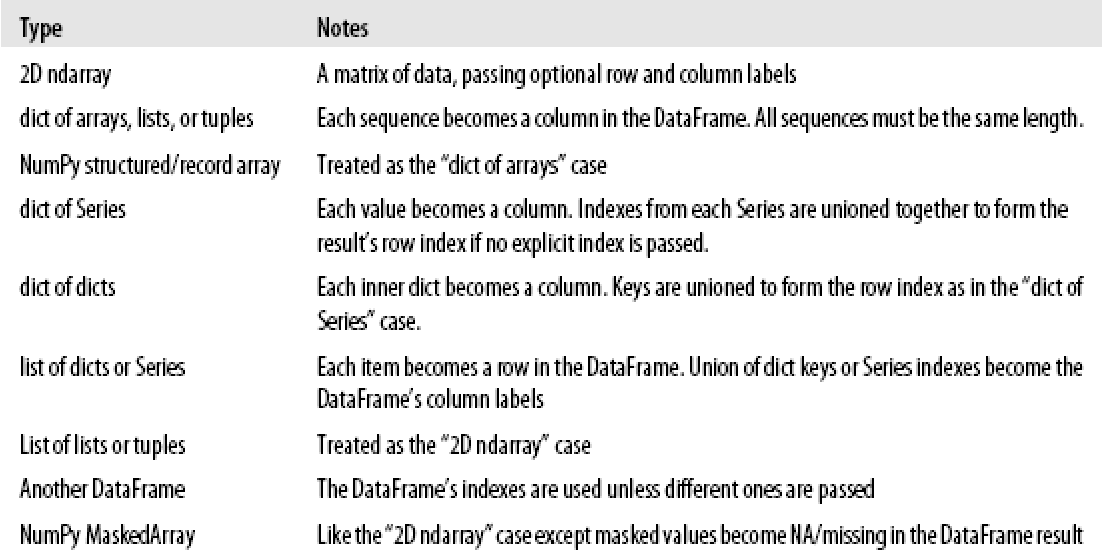
	Modi	Rahul
2012	78	34
2014	98	22
2016	NaN	NaN

```
In [18]:  ex= {'Gujarat':elections['Modi'][:-1], 'India': elections['Rahul'][:2]
          px =DataFrame(ex)
          px
```

```
Out[18]:
               Gujarat  India
          2010      72     55
          2012      78     34
```

```
In [150]: from IPython.display import Image
          i = Image(filename='Constructors.png')
          i # list of things you can pass to a dataframe
```

Out[150]:



Type	Notes
2D ndarray	A matrix of data, passing optional row and column labels
dict of arrays, lists, or tuples	Each sequence becomes a column in the DataFrame. All sequences must be the same length.
NumPy structured/record array	Treated as the "dict of arrays" case
dict of Series	Each value becomes a column. Indexes from each Series are unioned together to form the result's row index if no explicit index is passed.
dict of dicts	Each inner dict becomes a column. Keys are unioned to form the row index as in the "dict of Series" case.
list of dicts or Series	Each item becomes a row in the DataFrame. Union of dict keys or Series indexes become the DataFrame's column labels
List of lists or tuples	Treated as the "2D ndarray" case
Another DataFrame	The DataFrame's indexes are used unless different ones are passed
NumPy MaskedArray	Like the "2D ndarray" case except masked values become NA/missing in the DataFrame result

```
In [155]: px.index.name = 'year'
          px.columns.name = 'politicians'
          px
```

```
Out[155]:
          politicians  Gujarat  India
                 year
                 2010       72     55
                 2012       78     34
```

```
In [156]: px.values
```

```
Out[156]: array([[72, 55],
                 [78, 34]], dtype=int64)
```

In [3]:

```
jeeva = Series([5,4,3,2,1,-7,-29], index =['a','b','c','d','e','f','h'
index = jeeva.index
print index #u denotes unicode
print index[1:]# returns all the index elements except a.
index[1] = 'f' # you cannot modify an index element. It will generate
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call 
<ipython-input-3-e8b7ee2d0552> in <module>()
      3 print index #u denotes unicode
      4 print index[1:]# returns all the index elements except a.
----> 5 index[1] = 'f' # you cannot modify an index element. It will ge

C:\Users\tk\AppData\Local\Enthought\Canopy32\User\lib\site-packages\pa
    177         """This method will not function because object is imm
    178         raise TypeError("'%s' does not support mutable operati
--> 179                         self.__class__)
    180 
    181     __setitem__ = __setslice__ = __delitem__ = __delslice__ = 

TypeError: '<class 'pandas.core.index.Index'>' does not support mutabl


Index([u'a', u'b', u'c', u'd', u'e', u'f', u'h'], dtype='object')
Index([u'b', u'c', u'd', u'e', u'f', u'h'], dtype='object')
```

In [22]:

```
print px
2013 in px.index # checks if 2003 is an index in data frame px
```

```
      Gujarat  India
2010       72     55
2012       78     34
```

Out[22]:

```
False
```

Reindex

In [27]:

```
var = Series(['Python', 'Java', 'c', 'c++', 'Php'], index =[5,4,3,2,1])
print var
var1 = var.reindex([1,2,3,4,5])# reindex creates a new object
print var1
```

```
5    Python
4      Java
3         c
2       c++
1       Php
dtype: object
1       Php
2       c++
3         c
4      Java
5    Python
dtype: object
```

In [28]:

```
var.reindex([1,2,3,4,5,6,7])# introduces new indexes with values Nan
```

Out[28]:

```
1       Php
2       c++
3         c
4      Java
5    Python
6       NaN
7       NaN
dtype: object
```

In [31]:

```
var.reindex([1,2,3,4,5,6,7], fill_value =1) # you can use fill value t
```

Out[31]:

```
1       Php
2       c++
3         c
4      Java
5    Python
6         1
7         1
dtype: object
```

In [35]:

```
gh =Series(['Dhoni', 'Sachin', 'Kohli'], index =[0,2,4])
print gh
gh.reindex(range(6), method ='ffill') #ffill is forward fill. It forwa
```

```
0     Dhoni
2    Sachin
4     Kohli
dtype: object
```

Out[35]:

```
0     Dhoni
1     Dhoni
2    Sachin
3    Sachin
4     Kohli
5     Kohli
dtype: object
```

In [36]:

```
gh.reindex(range(6), method ='bfill')# bfill, backward fills the value
```

Out[36]:

```
0     Dhoni
1    Sachin
2    Sachin
3     Kohli
4     Kohli
5       NaN
dtype: object
```

In [45]:

```
import numpy as np
fp = DataFrame(np.arange(9).reshape((3,3)),index =['a','b','c'], colum
fp
```

Out[45]:

	Gujarat	Tamil Nadu	Kerala
a	0	1	2
b	3	4	5
c	6	7	8

In [55]:

```
fp1 =fp.reindex(['a', 'b', 'c', 'd'], columns = states) # reindexing c
fp1
```

Out[55]:

	Gujarat	Assam	Kerala
a	0	NaN	2
b	3	NaN	5
c	6	NaN	8
d	NaN	NaN	NaN

Other Reindexing arguments

limit When forward- or backfilling, maximum size gap to fill

level Match simple Index on level of MultiIndex, otherwise select subset of
copy Do not copy underlying data if new index is equivalent to old index. True by default (i.e. always copy data).

Dropping entries from an axis

In [62]:

```
er = Series(np.arange(5), index =['a','b','c','d','e'])
print er
er.drop(['a','b']) #drop method will return a new object  with values
```

```
a    0
b    1
c    2
d    3
e    4
dtype: int32
```

Out[62]:

```
c    2
d    3
e    4
dtype: int32
```

In [77]:

```
states ={'State' :['Gujarat', 'Tamil Nadu', ' Andhra', 'Karnataka', 'Ke
                  'Population': [36, 44, 67,89,34],
                  'Language' :['Gujarati', 'Tamil', 'Telugu', 'Kannada'
india = DataFrame(states, columns =['State', 'Population', 'Language']
print india
india.drop([0,1])# will drop index 0 and 1
```

```
        State  Population   Language
0     Gujarat          36   Gujarati
1  Tamil Nadu          44      Tamil
2      Andhra          67     Telugu
3   Karnataka          89    Kannada
4      Kerala          34  Malayalam
```

Out[77]:

	State	Population	Language
2	Andhra	67	Telugu
3	Karnataka	89	Kannada
4	Kerala	34	Malayalam

In [82]:

```
india.drop(['State', 'Population'], axis =1 )# the function dropped po
```

Out[82]:

	Language
0	Gujarati
1	Tamil
2	Telugu
3	Kannada
4	Malayalam

Selection, Indexing and Filtering

In [102]:

```
var = Series(['Python', 'Java', 'c', 'c++', 'Php'], index =[5,4,3,2,1])
var
```

Out[102]:

```
5    Python
4      Java
3         c
2       c++
1       Php
dtype: object
```

In [103]:

```
print var[5]
print var[2:4]
```

```
Python
3      c
2    c++
dtype: object
```

In [104]:

```
var[[3,2,1]]
```

Out[104]:

```
3      c
2    c++
1    Php
dtype: object
```

In [109]:

```
var[var == 'Php']
```

Out[109]:

```
1    Php
dtype: object
```

In [111]:

```
states ={'State' :['Gujarat', 'Tamil Nadu', ' Andhra', 'Karnataka', 'Ke
                    'Population': [36, 44, 67,89,34],
                    'Language' :['Gujarati', 'Tamil', 'Telugu', 'Kannada'
india = DataFrame(states, columns =['State', 'Population', 'Language'])
india
```

Out[111]:

	State	Population	Language
0	Gujarat	36	Gujarati
1	Tamil Nadu	44	Tamil
2	Andhra	67	Telugu
3	Karnataka	89	Kannada
4	Kerala	34	Malayalam

In [114]:

```
india[['Population', 'Language']] # retrieve data from data frame
```

Out[114]:

	Population	Language
0	36	Gujarati
1	44	Tamil
2	67	Telugu
3	89	Kannada
4	34	Malayalam

In [115]:

```
india[india['Population'] > 50] # returns data for population greater
```

Out[115]:

	State	Population	Language
2	Andhra	67	Telugu
3	Karnataka	89	Kannada

In [117]:

```
india[:3] # first three rows
```

Out[117]:

	State	Population	Language
0	Gujarat	36	Gujarati
1	Tamil Nadu	44	Tamil
2	Andhra	67	Telugu

In [4]:

```
# for selecting specific rows and columns, you can use ix function
import pandas as pd
states ={'State' :['Gujarat', 'Tamil Nadu', ' Andhra', 'Karnataka', 'Ke
                  'Population': [36, 44, 67,89,34],
                  'Language' :['Gujarati', 'Tamil', 'Telugu', 'Kannada'
india = DataFrame(states, columns =['State', 'Population', 'Language'],
india
```

Out[4]:

	State	Population	Language
a	Gujarat	36	Gujarati
b	Tamil Nadu	44	Tamil
c	Andhra	67	Telugu
d	Karnataka	89	Kannada
e	Kerala	34	Malayalam

In [128]:

```
india.ix[['a','b'], ['State','Language']] # this is how you select sub
```

Out[128]:

	State	Language
a	Gujarat	Gujarati
b	Tamil Nadu	Tamil