Python Pandas Series & DataFrame

Python Pandas:

Pandas is most popular library. It provides various functions related to Scientific Data analysis, like

- **It can read and write different data formats like int, float, double**
- **It can calculate data that is organized in row and columns.**
- **It can select sub set of data values and merge two data sets.**
- **It can support reshape of data values.**
- **It can support visualization library matplot.**

Data Structure:

Pandas Data Structure is a way to store & organize data values in a specific manner so that various specific functions can be applied on them. Examples- array, stack, queue, linked list, series, DataFrame etc.

"Series Vs DataFrame" Data Structure:

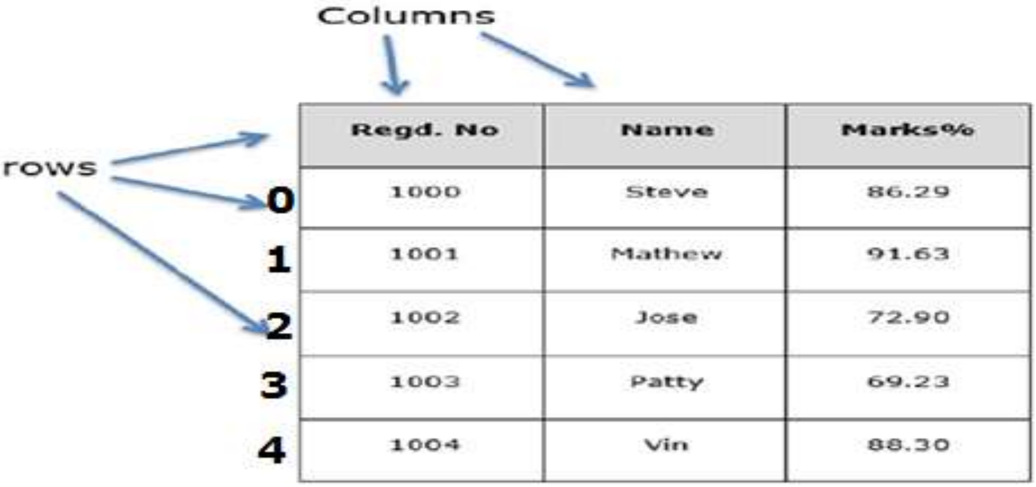
Property	Series	DataFrame
Dimensions	One-Dimensional (1-D)	Two-Dimensional (2-D)
Types of data	Homogenous (In Series, all data values should be of same type)	Heterogeneous (In DataFrame, data values may be of different type)
Value Mutable	Yes, Mutable (Values of elements can be changed)	Yes, Mutable (Values of elements can be changed)
Size Mutable	Size is Immutable. Once the size of series created, it cannot be changed. (If add/delete element, then new series object will be created.)	Size is Immutable. Once the size of DataFrame created, it can be changed. (That mean elements in DataFrame can add/delete.)

"DataFrame" Data Structure:

A DataFrame is another Pandas Data Structure that represent 2–Dimensional array of indexed data. It stores the different types of data in tabular form as rows and columns.

Features of DataFrame:

- **Potentially columns are used to store of different types of data.**
- **Row and column can delete that mean Size is Mutable.**
- **Data value can be changed that mean value is Mutable.**
- **Can Perform Arithmetic operations on rows and columns**
- **It has two indexes, Row index on Axis-0, Column on Axis-1.**
- **The data values are identical with combination of row index and column index**
- **The indexes can be numbers, characters or strings.**



	Regd. No	Name	Marks%
0	1000	Steve	86.29
1	1001	Mathew	91.63
2	1002	Jose	72.90
3	1003	Patty	69.23
4	1004	Vin	88.30

Creation of DataFrame:

A DataFrame object can be created by using following syntax.

Syntax:

pandas.DataFrame(data, index, columns, dtype, copy)

Where

1. **Data:** **data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame**
2. **Index:** **For the row labels, (Optional)**
   Default np.arange(n), if no index is passed
3. **Column:** **For column labels, (Optional)**
   Default np.arange(n), if no index is passed
4. **Dtype:** **Data type of each column**
5. **Copy:** **This parameter is used for copying of data, (Optional)**
   Default is False, if not passed

A pandas DataFrame can be created using various inputs like –

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

1. Creation of empty DataFrame by using DataFrame( ):

Syntax:

```
        DataFrame_object = pandas.DataFrame( )
        # D and F are capital in DataFrame( )
```

Example:

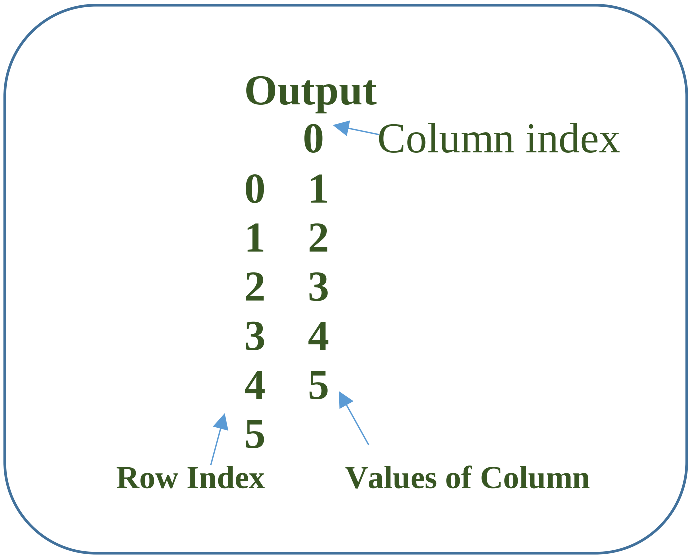
```
        #import the pandas library and aliasing as pd
        import pandas as pd
        df = pd.DataFrame()
        print (df)
```

Output:

```
        Empty DataFrame
        Columns: []
        Index: []
```

2. Create a DataFrame from Lists:

```
        import pandas as pd
        data = [1,2,3,4,5]
        df = pd.DataFrame(data)
        print (df)
```



```
Output
     0
0    1
1    2
2    3
3    4
4    5
5
```

```
import pandas as pd
data = [['Tina',10],['Naman',12],['Rita',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print (df)
```

```
      Output:

      Name      Age
0     Tina      10
1     Naman     12
2     Rita      13
```

```
import pandas as pd
```

```
data = [['Tina',10],['Naman',12],['Rita',13]]
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)
print (df)
```

```
      Output:

       Name        Age
0      Tina        10.0
1      Naman       12.0
2      Rita        13.0
```

Note − The dtype parameter changes the type of Age column to floating point.

3. Create a DataFrame from Dict of ndarrays / Lists

All the ndarrays must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

If no index is passed, then by default, index will be range(n), where n is the array length.

```
import pandas as pd
data = {'Name':['Tina', 'John', 'Seema', 'Reena'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print (df)
```

```
      Age        Name
0      28        Tina
1      34        John
2      29        Seema
3      42        Reena
```

Note − The values 0,1,2,3. They are the default index assigned to each using the function range(n).

```
import pandas as pd
data = {'Name':['Tina', 'Jaohn', 'Seema', 'Reena'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print (df)
```

```
          Age    Name
rank1     28     Tina
rank2     34     John
rank3     29     Seema
rank4     42     Reena
```

4. Create a DataFrame from List of Dictionaries

List of Dictionaries can be passed as input data to create a DataFrame. The dictionary keys are by default taken as column names.

```
dict={'Student:':['Tina','Geeta','Moti','Mangal'],
        'Marks':[23,45,76,32],
        'Sports':['Badminton','Volleyball','Kabaddi','Cricket']}
print("Dictionary is\n",dict)
df=pd.DataFrame(dict)
print("DataFrame is\n",df)
```

Output:

```
Dictionary is
 {'Sports': ['Badminton', 'Volleyball', 'Kabaddi', 'Cricket'], 'Marks':
[23, 45, 76, 32], 'Student:': ['Tina', 'Geeta', 'Moti', 'Mangal']}

DataFrame is
     Marks     Sports         Student:
0     23         Badminton     Tina
1     45         Volleyball     Geeta
2     76         Kabaddi         Moti
3     32         Cricket          Mangal
```

Note: indices are as same series (0 to 3) but columns in DataFrame are the indices of dictionary and display in sorted order.

```
df2=pd.DataFrame(dict, index=[‘I’,’II’,’III’,’IV’]
print(df2)
```

Output:

```
     Marks     Sports         Student:
I    23          Badminton     Tina
II    45         Volleyball     Geeta
III    76        Kabaddi         Moti
IV    32         Cricket          Mangal
```

```
df=pd.DataFrame(dict,index=['I','II','III','IV'],
columns=['Student','Sports'])
print(df)
```

Output:

```
Student         Sports
I      Tina        Badminton
```

```
II    Geeta      Volleyball
III   Moti       Kabaddi
IV    Mangal     Cricket
```

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print (df)
```

```
     a    b      c
0    1    2      NaN
1    5    10     20.0
```

Note − NaN (Not a Number) is appended in missing areas.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print (df)
```

```
           a    b     c
first      1    2     NaN
second     5    10    20.0
```

5. Creation of DataFrame from dictionary of dictionaries

```
Student={‘Sci’:{‘Name’:’Babu’,’Age’:15,’City’:’Ajmer’},
          {‘Arts’:{‘Name’:’John’,’Age’:17,’City’:’Jaipur’},
          {‘Com’:{‘Name’:’Heera’,’Age’:14,’City’:’Bikaner’}}
Df=pd.DataFrame(Student)
```

OutPut:

```
          Arts      Com       Sci
Age       17        14        15
City      Jaipur    Bikaner   Ajmer
Name      John      Heera     Babu
```

Note: Keys of Inner dictionary makes index and Keys of Outer dictionary makes columns of DataFrame. (Sorted form)

```
d1= {'Year-1':1500,'Year-2':2000}
d2= {'Year-1':2500,'Year-3':3000}
dict={'I':d1,'II':d2}
df=pd.DataFrame(dict)
print(df)
```

Output:

```
          I          II
Year-1    1500.0     2500.0
```

```
Year-2     2000.0     NaN
Year-3     NaN        3000.0
```

Example:

```
dict={'Population':{'Delhi':2000,'Mumbai':3000,'Kolkata':3500,'Chenni
':4000},
   'Hospitals':{'Delhi':200,'Mumbai':300,'Kolkata':350,'Chenni':400},
   'School':{'Delhi':20,'Mumbai':30,'Kolkata':35,'Chenni':40}}
df=pd.DataFrame(dict)
print(df)
```

Output:

```
               Hospitals  Population  School
Chenni         400        4000        40
Delhi          200        2000        20
Kolkata        350        3500        35
Mumbai         300        3000        30
```

Example:

```
d1={'Delhi':{'Population':100,'Hospitals':12,'Schools':52},
   'Mumbai':{'Population':200,'Hospitals':15,'Schools':60},
   'Kolkatta':{'Population':250,'Hospitals':17,'Schools':72},
   'Chenni':{'Population':300,'Hospitals':42,'Schools':62}}
df=pd.DataFrame(d1)
print(df)
```

Output:

```
               Chenni     Delhi  Mumbai  Kolkatta
Hospitals      42         12     15         17
Population     300        100    200        250
Schools        62         52     60         72
```

6. Create a DataFrame from Dictionary of Series

Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print (df)
```

```
        one     two
a       1.0     1
b       2.0     2
c       3.0     3
d       NaN     4
```

Problem: Write a program to create a dataframe from a list containing two lists. Each contains Target and actual Sales figure of four zonal offices. Give appropriate row label.

Solution:

```
Import pandas as pd
Target=[5000,6000,7000,8000]
Sales=[1000,2000,3000,4000]
ZoneSales=[Target, Sales]
Df=pd.DataFrame(ZoneSales, columns=[‘ZoneA’, ‘ZoneB’, ‘ZoneC’, ‘ZoneD’], index=[‘Target’,’Sales’])
Print(Df)
```

```
          ZoneA    ZoneB    ZoneC    ZoneD
Target     5000     6000     7000     8000
Sales      1000     2000     3000     4000
```

Problem: create two series object as “staff and salaries” to store number of staff and salary in various branches. Create another series object as “average” to calculate average salary of each branch. Then create DataFrame to display the records.

```
import pandas as pd
staff=pd.Series([10,20,40,50])
salary=pd.Series([20000,10000,30000,40000])
avg=salary/staff
org={'People':staff,'Salary':salary,'Average':avg}
df=pd.DataFrame(org)
print(df)
```

Output:

```
     Average  People     Salary
0    2000.0     10       20000
1    500.0      20       10000
2    750.0      40       30000
3    800.0      50       40000
```

7. Create a DataFrame from 2-D array

```
Import numpy as np
Import pandas as pd
Narr1=np.array([[10,20,30],[40,50,60]],np.int32)
Df=pd.DataFrame(Narr1)
Print(Df)
```

```
      0     1     2

0    10    20    30
1    40    50    60
```

```
Df=pd.DataFrame(Narr1, columns=[‘One’,’Two’,’Three’])
Print(Df)
```

```
     One   Two   Three

0    10    20    30
1    40    50    60
```

```
Df=pd.DataFrame(Narr1,columns=[‘One’,’Two’,’Three’], index=[‘A’,’B’])
Print(Df)
```

```
     One   Two   Three

A    10    20    30
B    40    50    60
```

8. Create a DataFrame from another DataFrame

```
import pandas as pd
import numpy as np
Df1=pd.DataFrame([[10,20,30],[40,50,60]])
Df2=(Df1)
print(Df2)
```

```
     0   1   2

0    10  20  30
1    40  50  60
```

Attributes of DataFrame Object

DataFrame Attribute	Description of Attribute
Index	Row label or index name of Row of DataFrame
columns	Column label or index name of column of DataFrame
Axes	Returns list of both axis(axis 0 for index (Row) and axis 1 for column)
Dtypes	Returns datatype of data values of DataFrame
Size	Returns integer values as number of elements in DataFrame
shape	Returns a tuple referencing dimension of DataFrame (rows, columns)
values	Returns Numpy representation of DataFrame
empty	Returns True if DataFrame is empty otherwise False.
Ndim	Returns an integer value representing number of axes/ dimensions.
T	Transpose index and columns

DataFrame Attribute	Output

import pandas as pd
data=[[10,20,30],[40,50,60]]
Cols=['Col-1','Col-2','Col-3']
Rows=['Row-1','Row-2']
Df1=pd.DataFrame(data, columns=Cols, index=Rows)
print(Df1)
	Col-1 Col-2 Col-3
	Row-1 10 20 30
	Row-2 40 50 60

Print("Index Attribute")
Print(Df1.index)
	Index Attribute:
	Index(['Row-1','Row-2'], dtype='object')

print("Columns Attribute")
print(Df1.columns)
	Columns Attribute
	Index(['Col-1','Col-2','Col-3'],dtype='object')

print("Asex Attribute")
print(Df1.axes)
	Asix Attribute
	[Index(['Row-1', 'Row-2'], dtype='object'), Index(['Col-1','Col-2','Col-3'], dtype='object')]

print("dtypes Attribute")
print(Df1.dtypes)
	dtypes Attribute
	Col-1 int64
	Col-2 int64
	Col-3 int64

print("size Attribute")
print(Df1.size)
	size Attribute
	6

print("shape Attribute")
print(Df1.shape)
	shape Attribute
	(2, 3)

print("Values Attribute")
print(Df1.values)
	Values Attribute
	[[10 20 30]
	 [40 50 60]]

print("ndim Attribute")
print(Df1.ndim)
	ndim Attribute
	2

print("empty Attribute")
print(Df1.empty)
	empty Attribute
	False

print("Transposing(T) Attribute")
print(Df1.T)
	Transposing(T) Attribute
	       Row-1 Row-2
	Col-1 10 40
	Col-2 20 50
	Col-3 30 60

Print(No of Rows in DataFrame”)
Print(len(Df1))
	No. of rows in DataFrame
	2

data=[[None,20,30],[40,50,None]]
Cols=['Col-1','Col-2','Col-3']
Rows=['Row-1','Row-2']
Df1=pd.DataFrame(data, columns=Cols, index=Rows)
print(Df1)
	Col-1 Col-2 Col-3
	Row-1 NaN 20 30.0
	Row-2 40.0 50 NaN

print("No. non NA Values in DataFrame")
print(Df1.count())
	No. non NA Values in DataFrame
	Col-1 1
	Col-2 2
	Col-3 1

print("No. of non NA Values in Axes-0 of DataFrame")
print(Df1.count(0))
OR
print(Df1.count(axis=’index’))
	No. of non NA Values in Axes-0 of DataFrame
	Col-1 1
	Col-2 2
	Col-3 1

print("No. non NA Values in Axes-1 of DataFrame")
print(Df1.count(1))
OR
print(Df1.count(axis=’columns’))
	No. non NA Values in Axes-1 of DataFrame
	Row-1 2
	Row-2 2

print("Access column Values from DataFrame")	Access column Values from DataFrame Row-1 NaN

print(Df1['Col-1'])	Row-2 40.0 Name: Col-1, dtype: float64

print("Access Multiple column Values from DataFrame")
print(Df1[['Col-1',’Col-3’]])
	Access Multiple column Values from DataFrame
	              Col-1 Col-3
	Row-1 NaN 30.0
	Row-2 40.0 NaN

print('Access specific Value')
print(Df1['Col-2']['Row-2'])
	Access specific Value
	50

print("change Values of whole column") Df1['Col-1']=100 print(Df1)	change Values of whole column Col-1 Col-2 Col-3 Row-1 100 20 30.0 Row-2 100 50 NaN

print("change specific Values")
Df1['Col-2']['Row-2']=200
print(Df1)
	change specific Values
	              Col-1 Col-2 Col-3
	Row-1 100 20 30.0
	Row-2 100 200 NaN

print("Add New Column")
Df1['Col-4']=400
print(Df1)
	Add New Column
	              Col-1 Col-2 Col-3 Col-4
	Row-1 NaN 20 30.0 400
	Row-2 40.0 50 NaN 400

print("Add New Row")
Df1.at['Row-3']=300
print(Df1)
	Col-1 Col-2 Col-3 Col-4
	Row-1 NaN 20.0 30.0 400.0
	Row-2 40.0 50.0 NaN 400.0
	Row-3 300.0 300.0 300.0 300.0

print("Delete column")
del Df1['Col-4']
print(Df1)
	Delete column
	              Col-1 Col-2 Col-3
	Row-1 NaN 20.0 30.0
	Row-2 40.0 50.0 NaN
	Row-3 300.0 300.0 300.0

print("Delete Row")
Df1=Df1.drop('Row-3') OR
Df1.drop('Row-3',inplace=True)
print(Df1)
	Delete Row
	              Col-1 Col-2 Col-3
	Row-1 NaN 20.0 30.0
	Row-2 40.0 50.0 NaN

print("Rename Index")
Df1.rename(index={'Row-1':'R-1','Row-3':'R-3'}, inplace=True)
	Rename Index
	            Col-1 Col-2 Col-3
	R-1 NaN 20.0 30.0

print(Df1)	Row-2 40.0 50.0 NaN R-3 300.0 300.0 300.0

print("Rename Column")
Df1.rename(columns={'Col-1':'C-1','Col-3':'C-3'}, inplace=True)
print(Df1)
	Rename column
	C-1 Col-2 C-3
	R-1 NaN 20.0 30.0
	Row-2 40.0 50.0 NaN
	R-3 300.0 300.0 300.0

Accessing / Selecting Sub Set from DataFrame:

DF_Object.loc(start_row: end_row, start_column: end_column)

```
dict={'Population':{'Delhi':2000,'Mumbai':3000,'Kolkata':3500,'Chenni':4000},
      'Hospitals':{'Delhi':200,'Mumbai':300,'Kolkata':350,'Chenni':400},
      'School':{'Delhi':20,'Mumbai':30,'Kolkata':35,'Chenni':40}}

df=pd.DataFrame(dict)
print(df)
```

Output:

```
                Hospitals  Population  School
Chenni          400        4000        40
Delhi           200        2000        20
Kolkata         350        3500        35
Mumbai          300        3000        30
```

Access a Row:

```
print("Access a row---------")
print(df.loc['Delhi'])
```

Output:

```
Access a row---------
Hospitals     200
Population    2000
School        20
```

```
print("Access Multiple row--------")
print(df.loc[['Delhi','Chenni']])
```

Ouput:

```
Access Multiple row--------
            Hospitals  Population  School
Delhi       200        2000          20
Chenni       400        4000          40
```

```
print("Access a column-------")
print(df.loc[:,'Population':'School'])
```

Output:

```
Access a column-------
                  Population  School
Chenni            4000          40
Delhi             2000          20
Kolkata           3500          35
Mumbai            3000          30
```

```
print("Access Mix of row & column-------")
print(df.loc['Delhi':'Mumbai','Hospital':'Population'])
```

Output:

```
Access Mix of row & column-------
                  Hospital
Delhi             200
Kolkata           350
Mumbai            300
```

loc[] Vs   iloc[]:

loc[] used to select / access the subset of DataFrame with the help of given row and column index (label). Where iloc[] used to access the subset of DataFrame by using the numeric index positions instead of labels.

Syntax:

DataFrame_Object.iloc[strat row index: end row index, start column index: end column index]

```
Print(Df.iloc[1:4])
```

Output:

```
              Hospital     Population  School
Delhi         200          2000          20
Kolkata       350          3500          35
Mumbai          300        3000         30
```

```
print(df.iloc[0:2,1:2])
```

Output:

```
              Population
Chenni       4000
Delhi         2000
```

Adding / Modifying Row / Column Value in DataFrame:

1. Adding / Modifying Columns Value

Df. Column_name=New Value

Df.[Column]=New Value

```
print("Add New Column=-----")
df['Density']=1200
print(df)
```

Output:

```
Add New Column=-----
              Hospital     Population  School  Density
Chenni       400           4000          40      1200
Delhi         200          2000          20      1200
Kolkata       350          3500          35      1200
Mumbai          300        3000          30      1200
```

```
print("Change Values of Column=-----")
df['Density']=[1200,1300,1320,1240]
print(df)
```

```
Change Values of Column=-----
              Hospital     Population  School  Density
Chenni       400           4000          40      1200
Delhi         200          2000          20      1300
Kolkata       350          3500          35      1320
Mumbai          300        3000          30      1240
```

2. Adding / Modifying Row’s Value

```
Df.at[row name,:]= Value
Df.loc[row name,:]= Value
```

```
print("Add New Row--------")
df.at['Banglore']=1500    OR    df.at['Banglore',:]=1500
print(df)
```

Output:

```
Add New Row--------
             Hospital    Population  School
Chenni       400.0       4000.0       40.0
Delhi        200.0       2000.0      20.0
Kolkata      350.0       3500.0      35.0
Mumbai       300.0       3000.0      30.0
Banglore     1500.0      1500.0      1500.0
```

```
print("Change Values of Row--------")
df.at['Banglore',:]=[1500,1300,2300]
print(df)
```

Output:

```
Change Values of Row--------
             Hospital    Population  School
Chenni       400.0       4000.0       40.0
Delhi        200.0       2000.0      20.0
Kolkata      350.0       3500.0      35.0
Mumbai       300.0       3000.0      30.0
Banglore     1500.0      1300.0      2300.0
```

3. Change / Modify Single Value

```
Df.column_name[row_name]= New value
```

```
print("Change Population of Delhi to 5000")
df.Population['Delhi']=5000
print(df)
```

Output:

```
Change Population of Delhi to 5000
             Hospital  Population  School
Chenni       400         4000        40
Delhi        200         5000        20
Kolkata      350         3500        35
Mumbai       300         3000        30
```

4. Delete Row / Column

```
del df[column_name]
print("Delete column--------")
del df['School']
print(df)
```

Output:

```
Delete column-------------
            Hospital  Population
Chenni      400         4000
Delhi       200         5000
Kolkata     350         3500
Mumbai      300         3000
```

```
print("Delete row-----")
df=df.drop('Mumbai')
print(df)
```

Output:

```
Delete row-----
            Hospital    Population  School
Chenni      400         4000        40
Delhi       200         5000        20
Kolkata     350         3500        35
```

5.Rename Row Index

```
print("Rename index---")
df=df.rename(index={'Delhi':'New Delhi','Kolkata':'Colcata'})
print(df)
```

Output:

```
Rename index---
            Hospital  Population  School
Chenni        400       4000          40
New Delhi     200       2000          20
Colcata       350       3500          35
Mumbai        300       3000          30
```

Note: df.rename(index={'Delhi':'New Delhi','Kolkata':'Colcata'}, inplace=True)
The inplace=True parameter change in orginal DataFrame.

```
df.rename(index={'Delhi':'New Delhi','Kolkata':'Colcata'})
print(df)
```

6. Rename Column Index

```
print("Rename Column Index:----")

df.rename(columns={'School':'college'}, inplace=True)
OR
df= df.rename(columns={'School':'College'}, inplace=True)

print(df)
```

Output:

```
Rename Column Index:----
              Hospital  Population  College
Chenni          400        4000          40
New Delhi       200        2000          20
Colcata         350        3500          35
Mumbai          300        3000          30
```

7. Boolean Indexing

The Boolean indexing refers to the index of the DataFrame as Boolean Values (True or False) (1 or 0). The advantage of Boolean index is to divide the DataFrame in Two sub groups.

Example:

```
print("Boolean Indexing-----")
d=['Mon','Tue','Wed','Thu','Fri','Sat']
cls=[2,0,0,7,0,6]
dic={'Day':d,'No. of Classes':cls}
df=pd.DataFrame(dic,index=[True,False,False,True,False,True])
print(df)
```

Output:

```
Boolean Indexing-----
              Day          No. of Classes
True          Mon            2
False         Tue            0
False         Wed            0
True          Thu            7
False         Fri             0
True          Sat             6
```

8. Access Values by using Boolean Index

```
df.loc[True]       OR       df.loc[1]      => It will show all True indexed records
df.loc[False]      OR       df.loc[0]      => It will show all False indexed records
```

```
print("Show True Index records----")
print(df.loc[True])
```

Output:

```
Show True Index records----
              Day          No. of Classes
True          Mon            2
True          Thu            7
True          Sat            6
```

```
print("Show False Index records----")
print(df.loc[False])
```

Output:

```
Show False Index records----
              Day          No. of Classes
False         Tue            0
False         Wed            0
False         Fri            0
```

```
print("Boolean Indexing-----")
d=['Mon','Tue','Wed','Thu','Fri','Sat']
cls=[2,0,0,7,0,6]
dic={'Day':d,'No. of Classes':cls}
df=pd.DataFrame(dic,index=[1,0,0,1,0,1])
print(df)
```

Output:

```
      Day          No. of Classes
1     Mon            2
0     Tue            0
0     Wed            0
1     Thu            7
0     Fri            0
1     Sat            6
```

```
print("Show True Index records----")
print(df.loc[1])
```

Output:

```
Show True Index records----
        Day            No. of Classes
1       Mon              2
1       Thu              7
1       Sat              6
```

Exporting DataFrame into CSV file.

Following template in Python in order to export your Pandas DataFrame to a CSV file:

df.to_csv(r'Path where you want to store the exported CSV file\File Name.csv', index = False)

To include the index, simply remove "**, index = False**" from the code:

```
import pandas as pd
cars={'Brand':['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],'Price': [22000,25000,27000,35000]}
df=pd.DataFrame(cars)
print("Write DataFrame into csv file-----")
df.to_csv(r'C:\export_dataframe.csv', index = False, header=True)
print(df)
```

Output:

```
Write DataFrame into csv file -----
          Brand               Price
0         Honda Civic         22000
1         Toyota Corolla      25000
2         Ford Focus          27000
3         Audi A4             35000
```

Importing csv file into DataFrame

The csv (Comma Separated Values) file can be read in DataFrame by using the read_csv( ) in Pandas.

Syntax:

DF.read_csv("Path of csv file", header, sep, index_col)

header: This allows to specify which row will be used as column names for dataframe. Default value is `header=0`, which means the first row of the CSV file will be treated as column names. If csv file doesn't have a header, then simply set `header=None`.

sep: Specify a custom delimiter for the CSV input, the default is a comma.

```
pd.read_csv('file_name.csv',sep='\t')     # Tab to separate
```

index_col: This is to allow you to set which columns to be used as the index of the dataframe. The default value is None, and pandas will add a new column start from 0 to specify the index column

```
pd.read_csv('file_name.csv',index_col='Name')
# 'Name' column as index
```

```
import pandas as pd
print("Read csv file and store into DataFrame-----")
df=pd.read_csv('C:\export_dataframe.csv')
print(df)
```

Output:

```
Read csv file and store into DataFrame-----
        Brand             Price
0       Honda Civic       22000
1       Toyota Corolla    25000
2       Ford Focus        27000
3       Audi A4           35000
```

****Finish****