CHAPTER-1 Data Handling using Pandas –I

Pandas:

- It is a package useful for data analysis and manipulation.
- Pandas provide an easy way to create, manipulate and wrangle the data.
- Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

Data scientists use Pandas for its following advantages:

- Easily handles missing data.
- It uses Series for one-dimensional data structure and DataFrame for multi-dimensional data structure.
- It provides an efficient way to slice the data.
- It provides a flexible way to merge, concatenate or reshape the data.

DATA STRUCTURE IN PANDAS

A data structure is a way to arrange the data in such a way that so it can be accessed quickly and we can perform various operation on this data like- retrieval, deletion, modification etc.

Pandas deals with 3 data structure-

1. Series
2. Data Frame
3. Panel

We are having only series and data frame in our syllabus.

Series

Series-Series is a one-dimensional array like structure with homogeneous data, which can be used to handle and manipulate data. What makes it special is its index attribute, which has incredible functionality and is heavily mutable.

It has two parts-

1. **Data part (An array of actual data)**
2. **Associated index with data (associated array of indexes or data labels)**

e.g.-

Index	Data
0	10
1	15
2	18
3	22

- ✓ We can say that **Series** is a labeled *one-dimensional array which can hold any type of data.*
- ✓ Data of **Series** is *always mutable*, means it can be changed.
- ✓ But the size of Data of **Series** is *always immutable*, means it cannot be changed.
- ✓ **Series** may be considered as a **Data Structure with two arrays** out which **one array** works as *Index (Labels)* and the **second array** works as *original Data.*
- ✓ ***Row Labels*** in Series are called *Index*.

Syntax to create a Series:

<Series Object>=pandas.Series (data, index=idx *(optional)*)

- ✓ Where data may be *python sequence (Lists)*, ndarray, scalar value or a python dictionary.

How to create Series with nd array

Program-

```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22])
s = pd.Series(arr)
print(s)
```

Here we create an array of 4 values.

Default Index

Output-

```
0  10
1  15
2  18
3  22
```

Data

How to create Series with Mutable index

Program-

```
import pandas as pd
import numpy as np
arr=np.array(['a','b','c','d'])
s=pd.Series(arr,
   index=['first','second','third','fourth'])
print(s)
```

Output-

```
first     a
second    b
third     c
fourth    d
```

Creating a series from Scalar value

To create a series from scalar value, an index must be provided. The scalar value will be repeated as per the length of index.

```
1  import pandas as pd
2  s = pd.Series(50, index =[0, 1, 2, 3, 4])
3  print(s)
4
```

```
0    50
1    50
2    50
3    50
4    50
dtype: int64
```

Creating a series from a Dictionary

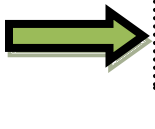
```
1  # import the pandas lib as pd
2  import pandas as pd
3
4  # create a dictionary
5  d = {'Name' : 'Hardik', 'Iplteam' : 'MI', 'Runs' : 1500}
6
7  # create a series
8  s = pd.Series(d)
9
10 print(s)
11
```

```
Name       Hardik
Iplteam        MI
Runs         1500
dtype: object
```

Mathematical Operations in Series

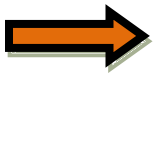
```
import pandas as pd
s=pd.Series([1,2,3,4,5])
print('To Multiply all values in a series by 2')
print('-----------------------------------------------------')
print(s*2)
print('To Find the Square of  all the values in a series ')
print('-----------------------------------------------------')
print(s**2)
print('To print all the values in a series that are greater than 2')
print('-----------------------------------------------------')
print(s[s>2])
```

```
To Multiply all values in a series by 2
-----------------------------------------------------
0     2
1     4
2     6
3     8
4    10
dtype: int64
```



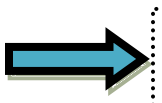
Print all the values of the Series by multiplying them by 2.

```
To Find the Square of  all the values in a series
-----------------------------------------------------
0     1
1     4
2     9
3    16
4    25
dtype: int64
```



Print Square of all the values of the series.

```
To print all the values in a series that are greater than 2
-----------------------------------------------------
2    3
3    4
4    5
dtype: int64
```



Print all the values of the Series that are greater than 2.

Example-2

```
import pandas as pd
s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
s2=pd.Series([10,20,30,40,50],index=['a','b','c','d','e'])
s3=pd.Series([5,14,23,32],index=['a','b','c','d'])
print('To Add Series1 & series2')
print('-----------------------------------------------------')
print(s1+s2)
print('To Add Series2 & Series3')
print('-----------------------------------------------------')
print(s2+s3)
print('To Add Series2 & series3 and Filled Non Matching Index with 0')
print('-----------------------------------------------------')
print(s2.add(s3,fill_value=0))
```

```
To Add Series1 & series2
-----------------------------------------------------
a    11
b    22
c    33
d    44
e    55
dtype: int64
To Add Series2 & Series3
-----------------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e     NaN
dtype: float64
To Add Series2 & series3 and Filled Non Matching Index with 0
-----------------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e    50.0
dtype: float64
```

While adding two series, if Non-Matching Index is found in either of the Series, Then NaN will be printed corresponds to Non-Matching Index.

If Non-Matching Index is found in either of the series, then this Non-Matching Index corresponding value of that series will be filled as 0.

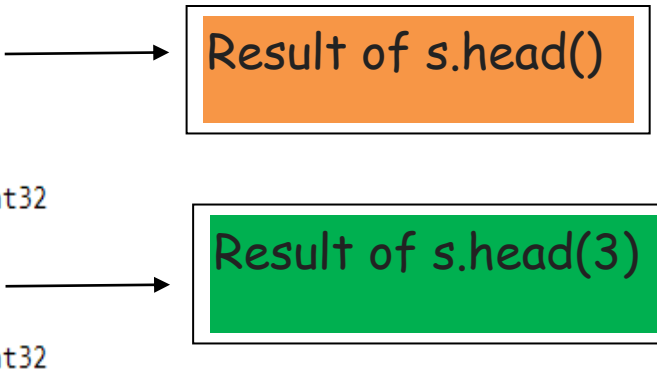
Head and Tail Functions in Series

head (): It is used to access the first 5 rows of a series.

Note :To access first 3 rows we can call series_name.head(3)

```
:
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77,42,48,97])
4  # create a series from array
5  s = pd.Series(arr)
6  # to print fiest 5 rows
7  print (s.head())
8  # To print first 3 rows
9  print(s.head(3))
```

```
0    10
1    15
2    18
3    22
4    55
dtype: int32
0    10
1    15
2    18
dtype: int32
```



tail(): It is used to access the last 5 rows of a series.

Note :To access last 4 rows we can call series_name.tail (4)

```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77,42,48,97])
# create a series from array
s = pd.Series(arr)
# to print last 5 rows
print (s.tail())
# To print last 4 rows
print(s.tail(4))
```

```
4    55
5    77
6    42
7    48
8    97
dtype: int32
5    77
6    42
7    48
8    97
dtype: int32
```

Selection in Series

Series provides index label loc and ilocand [] to access rows and columns.

1. loc index label :-

Syntax:-series_name.loc[StartRange: StopRange]

Example-

```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr)
5  print(s)
6  print(s.loc[:2])
7  print(s.loc[3:4])
8  s.loc[2:3]
```

To Print Values from Index 0 to 2

To Print Values from Index 3 to 4

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
0    10
1    15
2    18
dtype: int32
3    22
4    55
dtype: int32

2    18
3    22
dtype: int32
```

2. Selection Using iloc index label :-

Syntax:-series_name.iloc[StartRange : StopRange]

Example-

```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr)
5  print(s)
6  print(s.iloc[:2])
7  print(s.iloc[3:4])
8  s.iloc[2:3]
```

To Print Values from Index 0 to 1.

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
0    10
1    15
dtype: int32
3    22
dtype: int32

2    18
dtype: int32
```

3. Selection Using [] :

Syntax:-series_name[StartRange> : StopRange] or

series_name[index]

Example-

```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr)
5  print(s)
6  print(s[1])
7  print('\n')
8  print(s[3:4])
9  s[:3]
```

To Print Values at Index 3.

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
15


3    22
dtype: int32

0    10
1    15
2    18
dtype: int32
```

Indexing in Series

Pandas provide index attribute to get or set the index of entries or values in series.

Example-

```
:
1  import pandas as pd
2  import numpy as np
3  arr=np.array(['a','b','c','d'],)
4  s=pd.Series(arr,index=['first','second','third','fourth'])
5  print(s)
6  # To print only indexes in series
7  print('\n indexes in Series are:::')
8  print(s.index)
9
```

```
first     a
second    b
third     c
fourth    d
dtype: object

 indexes in Series are:::
Index(['first', 'second', 'third', 'fourth'], dtype='object')
```

Slicing in Series

Slicing is a way to retrieve subsets of data from a pandas object. A slice object syntax is –

SERIES_NAME [start:end: step]

The segments start representing the first item, end representing the last item, and step representing the increment between each item that you would like.

Example :-

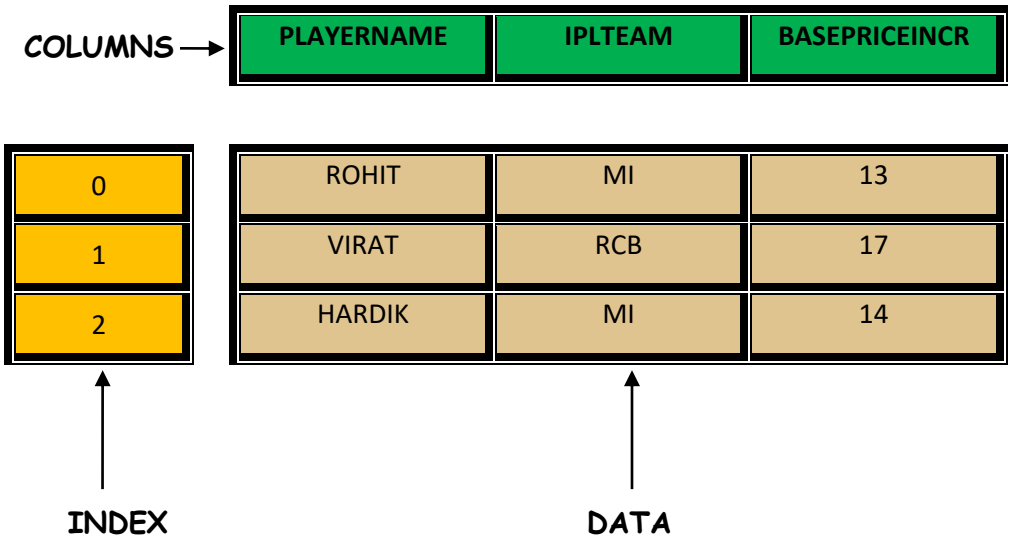
```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr,index=['A','B','C','D','E','F'])
5  print(s)
6  print(s[1:5:2])
7  print(s[0:6:2])
8
```

```
A    10
B    15
C    18
D    22
E    55
F    77
dtype: int32
B    15
D    22
dtype: int32
A    10
C    18
E    55
dtype: int32
```

DATAFRAME

DATAFRAME-It is a two-dimensional object that is useful in representing data in the form of rows and columns. It is similar to a spreadsheet or an SQL table. This is the most commonly used pandas object. Once we store the data into the Dataframe, we can perform various operations that are useful in analyzing and understanding the data.

DATAFRAME STRUCTURE



PROPERTIES OF DATAFRAME

1. A Dataframe has axes (indices)-
   - Row index (axis=0)
   - Column index (axes=1)
2. It is similar to a spreadsheet , whose row index is called index and column index is called column name.
3. A Dataframe contains Heterogeneous data.
4. A Dataframe Size is Mutable.
5. A Dataframe Data is Mutable.

A data frame can be created using any of the following-

1. **Series**
2. **Lists**
3. **Dictionary**
4. **A numpy 2D array**

How to create Empty Dataframe

```
: import pandas as pd
df=pd.DataFrame()
print(df)
```

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

How to create Dataframe From Series

Program-

```
import pandas as pd

s = pd.Series(['a','b','c','d'])

df=pd.DataFrame(s)

print(df)
```

Output-

```
   0
0  a
1  b
2  c
3  d
```

Default Column Name As 0

DataFrame from Dictionary of Series

Example-

```
import pandas as pd
name=pd.Series(['Hardik','Virat'])
team=pd.Series(['MI','RCB'])
dic={'Name':name,'Team':team}
df=pd.DataFrame(dic)
print(df)
```

```
     Name Team
0  Hardik   MI
1   Virat  RCB
```

DataFrame from List of Dictionaries

Example-

```
1  import pandas as pd
2  l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3       {'Name': 'Vinod', 'SirName':'Verma'},
4       {'Name': 'Rajesh', 'SirName':'Mishra'}]
5  df1=pd.DataFrame(l)
6  print(df1)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma
2  Rajesh    Mishra
```

Iteration on Rows and Columns

If we want to access record or data from a data frame row wise or column wise then iteration is used. Pandas provide 2 functions to perform iterations-

1. iterrows ()
2. iteritems ()

iterrows()

It is used to access the data row wise. Example-

```
1  import pandas as pd
2  l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3       {'Name': 'Vinod', 'SirName':'Verma'}]
4  df1=pd.DataFrame(l)
5  print(df1)
6  for(row_index,row_value) in df1.iterrows():
7      print('\n Row index is ::',row_index)
8      print('Row Value is::')
9      print(row_value)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma

 Row index is :: 0
Row Value is::
Name          Sachin
SirName     Bhardwaj
Name: 0, dtype: object

 Row index is :: 1
Row Value is::
Name         Vinod
SirName      Verma
Name: 1, dtype: object
```

iteritems()

It is used to access the data column wise.

Example-

```
1  import pandas as pd
2  l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3       {'Name': 'Vinod', 'SirName':'Verma'}]
4  df1=pd.DataFrame(l)
5  print(df1)
6  for(col_name,col_value) in df1.iteritems():
7      print('\n')
8      print('Column Name is ::',col_name)
9      print('Column Values are::')
10     print(col_value)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma


Column Name is :: Name
Column Values are::
0    Sachin
1     Vinod
Name: Name, dtype: object


Column Name is :: SirName
Column Values are::
0    Bhardwaj
1       Verma
Name: SirName, dtype: object
```

Select operation in data frame

To access the column data ,we can mention the column name as subscript.

e.g. - df[empid] This can also be done by using df.empid.

To access multiple columns we can write as df[[col1, col2,---]]

Example -

```
import pandas as pd
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi'],
          'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
df=pd.DataFrame(empdata)
print(df)
```

```
   empid     ename         Doj
0    101    Sachin   12-01-2012
1    102     Vinod   15-01-2012
2    103   Lakhbir   05-09-2007
3    104      Anil  17-01- 2012
4    105  Devinder   05-09-2007
5    106  UmaSelvi   16-01-2012
```

```
>>df.empid or df['empid']
0    101
1    102
2    103
3    104
4    105
5    106
Name: empid, dtype: int64

>>df[['empid','ename']]
   empid          ename
0    101         Sachin
1    102          Vinod
2    103        Lakhbir
3    104           Anil
4    105       Devinder
5    106       UmaSelvi
```

To Add & Rename a column in data frame

```
import pandas as pd

s = pd.Series([10,15,18,22])

df=pd.DataFrame(s)

df.columns=['List1'] ⟶ To Rename the default column of Data Frame as List1

df['List2']=20 ⟶ To create a new column List2 with all values as 20

df['List3']=df['List1']+df['List2']

Add Column1 and Column2 and store in New column List3

print(df)
```

Output-

	List1	List2	List3
0	10	20	30
1	15	20	35
2	18	20	38
3	22	20	42

To Delete a Column in data frame

We can delete the column from a data frame by using any of the the following –

1. del
2. pop()
3. drop()

```
>>del df['List3'] ⟶We can simply delete a column by passing column name in subscript with df
>>df
```

Output-

```
  List1  List2
0    10    20
1    15    20
2    18    20
3    22    20
```

```
>>df.pop('List2') ⟶ we can simply delete a column by passing column name in pop method.
>>df
```

```
   List1
0    10
1    15
2    18
3    22
```

To Delete a Column Using drop()

```
import pandas as pd
s= pd.Series([10,20,30,40])
df=pd.DataFrame(s)
df.columns=['List1']
df['List2']=40
df1=df.drop('List2',axis=1) ⟶ (axis=1) means to delete Data column wise
df2=df.drop(index=[2,3],axis=0)⟶ (axis=0) means to delete data row wise with given index
print(df)
print(" After deletion::")
print(df1)
print (" After row deletion::")
print(df2)
```

```
Output-
   List1  List2
0    10    40
1    20    40
2    30    40
3    40    40
After deletion::
    List1
0    10
1    20
2    30
3    40
After row deletion::
   List1
0    10
1    20
```

Accessing the data frame through loc() and iloc() method or indexing using Labels

Pandas provide loc() and iloc() methods to access the subset from a data frame using row/column.

Accessing the data frame through loc()

It is used to access a group of rows and columns.

Syntax-

Df.loc[StartRow : EndRow, StartColumn : EndColumn]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df.loc['Qtr3', : ])
10 print(df.loc['Qtr1':'Qtr3', : ])
11
```

To access a single row

To access multiple Rows Qtr1 to Qtr3

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
TCS       3000
WIPRO     3600
L&T      35000
Name: Qtr3, dtype: int64
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
```

Example 2:-

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df.loc[ : ,'TCS' ])
10 print(df.loc[ : , 'TCS':'WIPRO'])
11
```

To access single column

To access Multiple Column namely TCS and WIPRO

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
Qtr1    2500
Qtr2    2000
Qtr3    3000
Qtr4    2000
Name: TCS, dtype: int64
       TCS  WIPRO
Qtr1  2500   2800
Qtr2  2000   2400
Qtr3  3000   3600
Qtr4  2000   2400
```

Example-3

```
1 import pandas as pd
2 empdata={ 'empid':[101,102,103,104,105,106],
3           'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi'],
4           'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
5 df=pd.DataFrame(empdata)
6 print(df)
7 print(df.loc[0])
8 df.loc[0:2]
```

To access first row

To access first 3 Rows

```
   empid     ename          Doj
0    101    Sachin   12-01-2012
1    102     Vinod   15-01-2012
2    103   Lakhbir   05-09-2007
3    104      Anil  17-01- 2012
4    105  Devinder   05-09-2007
5    106  UmaSelvi   16-01-2012
empid           101
ename        Sachin
Doj      12-01-2012
Name: 0, dtype: object
```

	empid	ename	Doj
0	101	Sachin	12-01-2012
1	102	Vinod	15-01-2012
2	103	Lakhbir	05-09-2007

Accessing the data frame through iloc()

It is used to access a group of rows and columns based on numeric index value.

Syntax-

Df.loc[StartRowindexs : EndRowindex, StartColumnindex : EndColumnindex]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df.iloc[0 :2 ,1:2 ])
10 print(df.iloc[ : , 0:2])
11
```

To access First two Rows and Second column

To access all Rows and First Two columns Record

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
      WIPRO
Qtr1   2800
Qtr2   2400
      TCS  WIPRO
Qtr1  2500   2800
Qtr2  2000   2400
Qtr3  3000   3600
Qtr4  2000   2400
```

head() and tail() Method

The method head() gives the first 5 rows and the method tail() returns the last 5 rows.

```
import pandas as pd
empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',
                 '17-01-2012','05-09-2007','16-01-2012'],
          'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']
         }
df=pd.DataFrame(empdata)
print(df)
print(df.head())
print(df.tail())
```

Output-

```
         Doj  empid    ename
0 12-01-2012    101   Sachin
1 15-01-2012    102    Vinod
2 05-09-2007    103  Lakhbir
3 17-01-2012    104     Anil
4 05-09-2007    105  Devinder
5 16-01-2012    106  UmaSelvi
```

→ Data Frame

```
         Doj  empid    ename
0 12-01-2012    101   Sachin
1 15-01-2012    102    Vinod
2 05-09-2007    103  Lakhbir
3 17-01-2012    104     Anil
4 05-09-2007    105  Devinder
```

→ head() displays first 5 rows

```
         Doj  empid    ename
1 15-01-2012    102    Vinod
2 05-09-2007    103  Lakhbir
3 17-01-2012    104     Anil
4 05-09-2007    105  Devinder
5 16-01-2012    106  UmaSelvi
```

→ tail() display last 5 rows

To display first 2 rows we can use head(2) and to returns last2 rows we can use tail(2) and to return 3rd to 4th row we can write df[2:5].

```
import pandas as pd
empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',
                 '17-01-2012','05-09-2007','16-01-2012'],
          'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']
         }
df=pd.DataFrame(empdata)
print(df)
print(df.head(2))
print(df.tail(2))
print(df[2:5])
```

Output-

```
          Doj  empid    ename
0  12-01-2012    101   Sachin
1  15-01-2012    102    Vinod
2  05-09-2007    103  Lakhbir
3  17-01- 2012   104     Anil
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
```

```
          Doj  empid  ename
0  12-01-2012    101  Sachin
1  15-01-2012    102   Vinod
```

→ head(2) displays first 2 rows

```
          Doj  empid    ename
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
```

→ tail(2) displays last 2 rows

```
          Doj  empid    ename
2  05-09-2007    103  Lakhbir
3  17-01- 2012   104     Anil
4  05-09-2007    105  Devinder
```

→ df[2:5] display 2nd to 4th row

Boolean Indexing in Data Frame

Boolean indexing helps us to select the data from the DataFrames using a boolean vector. We create a DataFrame with a boolean index to use the boolean indexing.

```
1  import pandas as pd
2  dic= {
3         'Name': ['Sachin Bhardwaj', 'Vinod Verma', 'Rajesh Mishra'],
4         'Age': [32, 35, 40]
5      }
6  # creating a DataFrame with boolean index vector
7  df = pd.DataFrame(dic, index = [True, False, True])
8  print(df)
9  print(df.loc[True])
10 print()
11 print('Result of iloc method')
12 print(df.iloc[1])
```

To Return Data frame where index is True

We can pass only integer value in iloc

```
                 Name  Age
True   Sachin Bhardwaj   32
False      Vinod Verma   35
True     Rajesh Mishra   40
                 Name  Age
True  Sachin Bhardwaj   32
True    Rajesh Mishra   40

Result of iloc method
Name    Vinod Verma
Age              35
dtype: object
```

Concat operation in data frame

Pandas provides various facilities for easily combining together **Series, DataFrame.**

```
pd.concat(objs, axis=0, join='outer', join_axes=None,ignore_index=False)
```

- **objs** – This is a sequence or mapping of Series, DataFrame, or Panel objects.
- **axis** – {0, 1, ...}, default 0. This is the axis to concatenate along.
- **join** – {'inner', 'outer'}, default 'outer'. How to handle indexes on other axis(es). Outer for union and inner for intersection.
- **ignore_index** – boolean, default False. If True, do not use the index values on the concatenation axis. The resulting axis will be labeled 0, ..., n - 1.
- **join_axes** – This is the list of Index objects. Specific indexes to use for the other (n-1) axes instead of performing inner/outer set logic.

The Concat() performs concatenation operations along an axis.

Example-1

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  df3=pd.concat([df1,df2])
9  print(df3)
10
```

```
  id Value1 Value2
0  1      A      B
1  2      C      D
2  3      E      F
3  4      G      H
4  5      I      J
0  2      K      L
1  3      M      N
2  6      O      P
3  7      Q      R
4  8      S      T
```

Example-2

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  df3=pd.concat([df1,df2],ignore_index=True)
9  print(df3)
10
```

```
  id Value1 Value2
0  1      A      B
1  2      C      D
2  3      E      F
3  4      G      H
4  5      I      J
5  2      K      L
6  3      M      N
7  6      O      P
8  7      Q      R
9  8      S      T
```

If you want the row labels to adjust automatically according to the join, you will have to set the argument ignore_index as True while calling the concat() function.

Example-3

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  merge={'Data1':df1,'Data2':df2}
9  df3=pd.concat(merge)
10 print(df3)
11
```

```
        id Value1 Value2
Data1 0  1      A      B
      1  2      C      D
      2  3      E      F
      3  4      G      H
      4  5      I      J
Data2 0  2      K      L
      1  3      M      N
      2  6      O      P
      3  7      Q      R
      4  8      S      T
```

pandas also provides you with an option to label the DataFrames, after the concatenation, with a key so that you may know which data came from which DataFrame.

Example-4

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  df3=pd.concat([df1,df2],axis=1)
9  print(df3)
10
```

```
  id Value1 Value2 id Value1 Value2
0  1      A      B  2      K      L
1  2      C      D  3      M      N
2  3      E      F  6      O      P
3  4      G      H  7      Q      R
4  5      I      J  8      S      T
```

To concatenate DataFrames along column, you can specify the axis parameter as 1.

Merge operation in data frame

Two DataFrames might hold different kinds of information about the same entity and linked by some common feature/column. To join these DataFrames, pandas provides multiple functions like merge(), join() etc.

Example-1

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  dic3 = {'id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
7          'Value3': [12, 13, 14, 15, 16, 17, 15, 12, 13, 23]}
8  df1=pd.DataFrame(dic1)
9  df2=pd.DataFrame(dic2)
10 df3=pd.concat([df1,df2])
11 df4=pd.DataFrame(dic3)
12 df5=pd.merge(df3,df4,on='id')
13 print(df5)
```

```
  id Value1 Value2  Value3
0  1      A      B      12
1  2      C      D      13
2  2      K      L      13
3  3      E      F      14
4  3      M      N      14
5  4      G      H      15
6  5      I      J      16
7  7      Q      R      17
8  8      S      T      15
```

This will give the common rows between the two data frames for the corresponding column values ('id').

Example-2

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
dic3 = {'id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'Value3': [12, 13, 14, 15, 16, 17, 15, 12, 13, 23]}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.concat([df1,df2])
df4=pd.DataFrame(dic3)
df5=pd.merge(df3,df4,left_on='id', right_on='id')
print(df5)
```

	id	Value1	Value2	Value3
0	1	A	B	12
1	2	C	D	13
2	2	K	L	13
3	3	E	F	14
4	3	M	N	14
5	4	G	H	15
6	5	I	J	16
7	7	Q	R	17
8	8	S	T	15

It might happen that the column on which you want to merge the Data Frames have different names (unlike in this case). For such merges, you will have to specify the arguments left_on as the left DataFrame name and right_on as the right DataFrame name.

Join operation in data frame

It is used to merge data frames based on some common column/key.

1. Full Outer Join:- The full outer join combines the results of both the left and the right outer joins. The joined data frame will contain all records from both the data frames and fill in NaNs for missing matches on either side. You can perform a full outer join by specifying the how argument as outer in merge() function.

Example-

```
:  1 import pandas as pd
   2 dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
   3         'Value2': ['B', 'D', 'F', 'H', 'J']}
   4 dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
   5         'Value2': ['L', 'N', 'P', 'R', 'T']}
   6 df1=pd.DataFrame(dic1)
   7 df2=pd.DataFrame(dic2)
   8 df3=pd.merge(df1,df2,on='id',how='outer')
   9 print(df3)
```

	id	Value1_x	Value2_x	Value1_y	Value2_y
0	1	A	B	NaN	NaN
1	2	C	D	K	L
2	3	E	F	M	N
3	4	G	H	NaN	NaN
4	5	I	J	NaN	NaN
5	6	NaN	NaN	O	P
6	7	NaN	NaN	Q	R
7	8	NaN	NaN	S	T

The resulting DataFrame had all the entries from both the tables with NaN values for missing matches on either side. However, one more thing to notice is the suffix which got appended to the column names to show which column came from which DataFrame. The default suffixes are x and y, however, you can modify them by specifying the suffixes argument in the merge() function.

Example-2

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, left_on='id',right_on='id',how='outer',suffixes=('_left','_right'))
print(df3)
```

```
  id Value1_left Value2_left Value1_right Value2_right
0  1          A          B         NaN         NaN
1  2          C          D           K           L
2  3          E          F           M           N
3  4          G          H         NaN         NaN
4  5          I          J         NaN         NaN
5  6        NaN        NaN           O           P
6  7        NaN        NaN           Q           R
7  8        NaN        NaN           S           T
```

2.Inner Join :- The inner join produce only those records that match in both the data frame. You have to pass inner in how argument inside merge() function.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='inner')
print(df3)
```

```
  id Value1_x Value2_x Value1_y Value2_y
0  2        C        D        K        L
1  3        E        F        M        N
```

3. RightJoin :-The right join produce a complete set of records from data frame B(Right side Data Frame) with the matching records (where available) in data frame A(Left side data frame). If there is no match right side will contain null. You have to pass right in how argument inside merge() function.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='right')
print(df3)
```

```
  id Value1_x Value2_x Value1_y Value2_y
0  2        C        D        K        L
1  3        E        F        M        N
2  6      NaN      NaN        O        P
3  7      NaN      NaN        Q        R
4  8      NaN      NaN        S        T
```

4.Left Join :- The left join produce a complete set of records from data frame A(Left side Data Frame) with the matching records (where available) in data frame B(Right side data frame). If there is no match left side will contain null. You have to pass left in how argument inside merge() function.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='left')
print(df3)
```

```
  id Value1_x Value2_x Value1_y Value2_y
0  1        A        B      NaN      NaN
1  2        C        D        K        L
2  3        E        F        M        N
3  4        G        H      NaN      NaN
4  5        I        J      NaN      NaN
```

5. Joining on Index :-Sometimes you have to perform the join on the indexes or the row labels. For that you have to specify right_index(for the indexes of the right data frame) and left_index(for the indexes of left data frame) as True.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3= pd.merge(df1, df2, right_index=True, left_index=True)
print(df3)
```

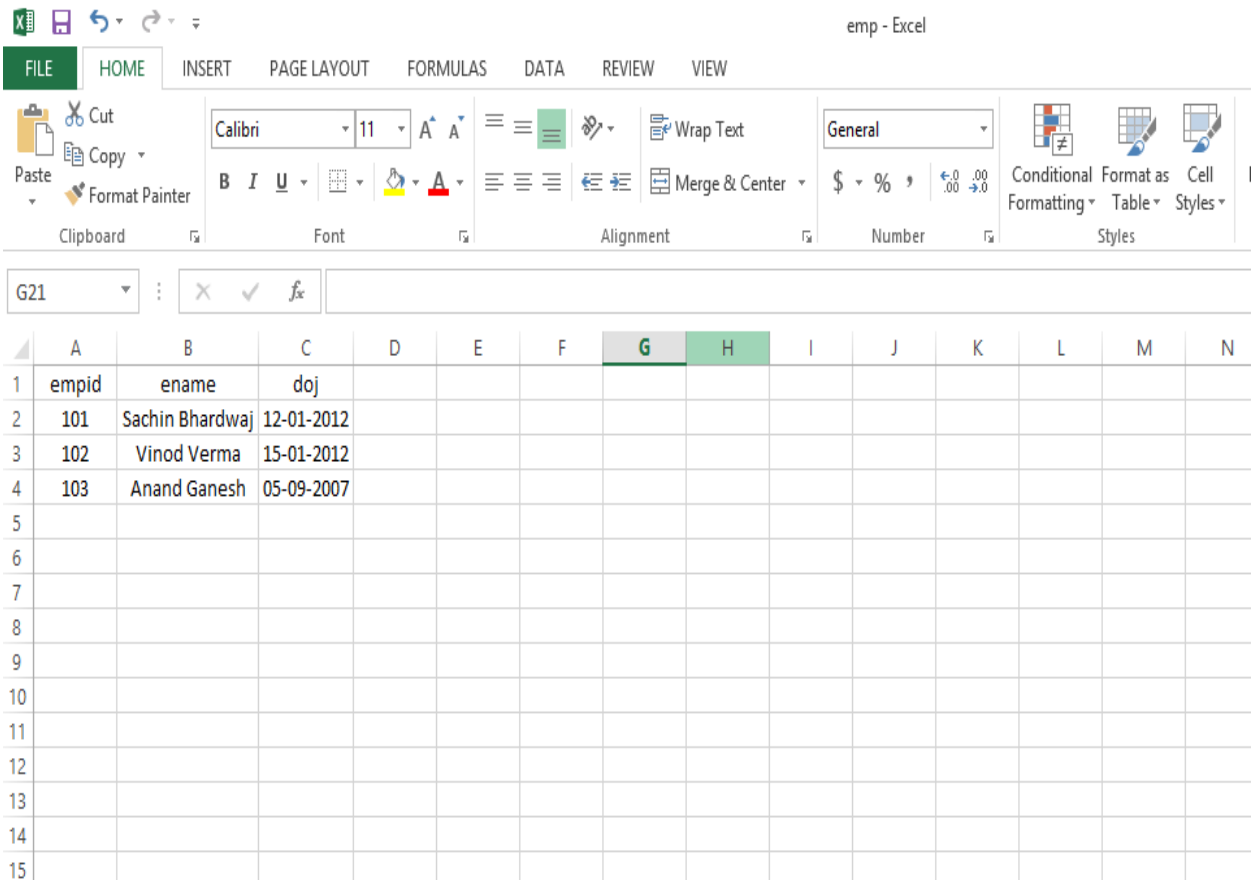
```
  id_x Value1_x Value2_x id_y Value1_y Value2_y
0    1        A        B    2        K        L
1    2        C        D    3        M        N
2    3        E        F    6        O        P
3    4        G        H    7        Q        R
4    5        I        J    8        S        T
```

CSV File

A CSV is a comma separated values file, which allows data to be saved in a tabular format. CSV is a simple file such as a spreadsheet or database. Files in the csv format can be imported and exported from programs that store data in tables, such as Microsoft excel or Open Office.

CSV files data fields are most often separated, or delimited by a comma. Here the data in each row are delimited by comma and individual rows are separated by newline.

To create a csv file, first choose your favorite text editor such as- Notepad and open a new file. Then enter the text data you want the file to contain, separating each value with a comma and each row with a new line. Save the file with the extension.csv. You can open the file using MS Excel or another spread sheet program. It will create the table of similar data.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	empid	ename	doj											
2	101	Sachin Bhardwaj	12-01-2012											
3	102	Vinod Verma	15-01-2012											
4	103	Anand Ganesh	05-09-2007											
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														

pd.read_csv() method is used to read a csv file.

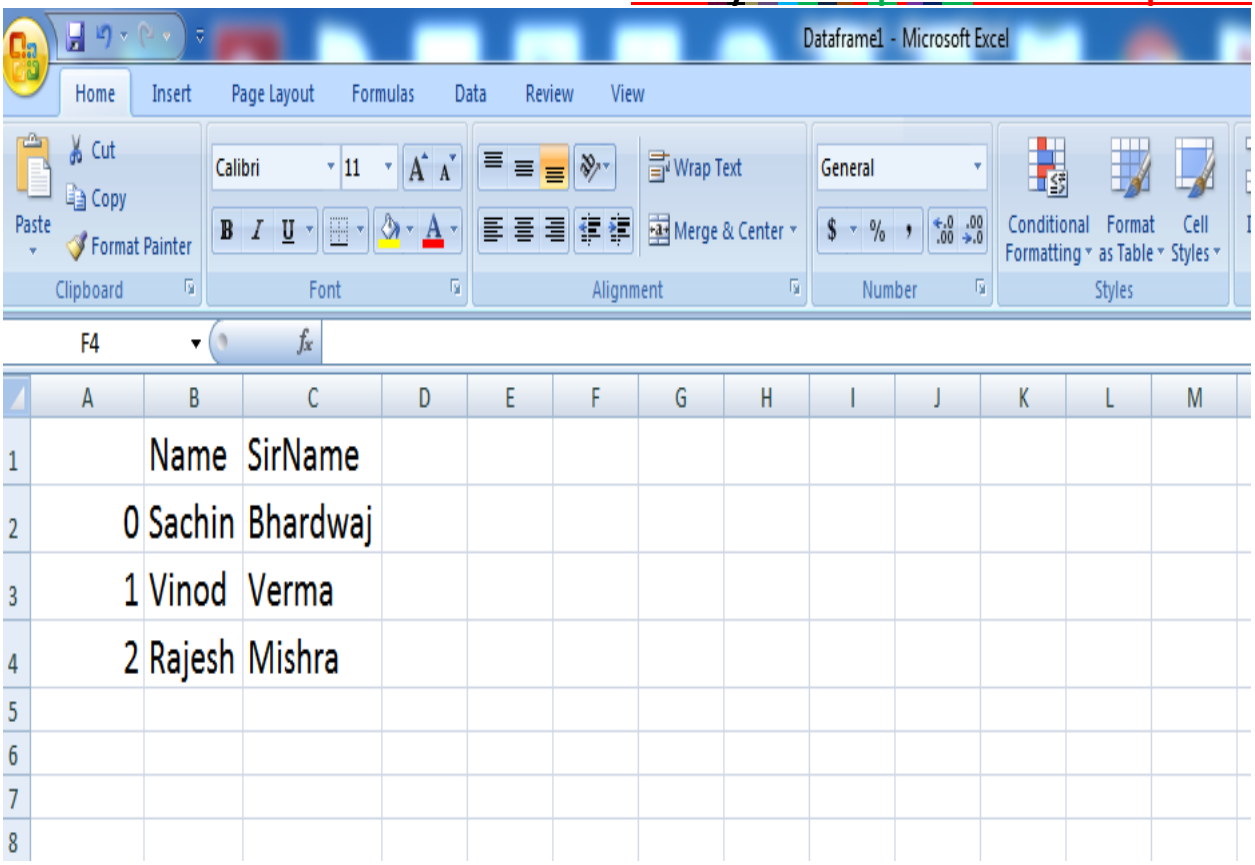
```
1  # importing pandas module
2  import pandas as pd
3  # making data frame
4  df = pd.read_csv("E:\emp.csv")
5  print(df)
6
```

```
   empid           ename         doj
0    101  Sachin Bhardwaj  12-01-2012
1    102      Vinod Verma  15-01-2012
2    103     Anand Ganesh  05-09-2007
```

Exporting data from dataframe to CSV File

To export a data frame into a csv file first of all, we create a data frame say df1 and use dataframe.to_csv(' E:\Dataframe1.csv ') method to export data frame df1 into csv file Dataframe1.csv.

```
1 import pandas as pd
2 l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3      {'Name': 'Vinod', 'SirName':'Verma'},
4      { 'Name': 'Rajesh','SirName':'Mishra'}]
5 df1=pd.DataFrame(l)
6 # saving the dataframe
7 df1.to_csv('E:\Dataframe1.csv')
```



And now the content of df1 is exported to csv file Dataframe1.

Unit-2-Data Handling using Pandas-II

Descriptive Statistics

Statistics is a branch of mathematics that deals with collecting, interpreting, organization and interpretation of data. Descriptive statistics involves summarizing and organizing the data so that it can be easily understood.

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

max()

It returns the maximum value from a column of a data frame or series.

Syntax-

df['columnname'].max()

Or

df.max(axis=0) ⟶ returns the maximum value of every column

Or

df.max(axis=1) ⟶ returns the maximum value of every row

```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

        'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':1800},

        'L&T': { 'Qtr1':5000,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df['WIPRO'].max())
print(df.max(axis=0))
```

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   5000
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   1800   2100
3600
TCS       3000
WIPRO     3600
L&T      35000
dtype: int64
```

min()

It returns the minimum value from a column of a data frame or series.

Syntax-

df['columnname'].min()

Or

df.min (axis=0) ⟶ returns the minimum value of every column

Or

df.min(axis=1) ⟶ returns the minimum value of every row

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':1800},
5
6          'L&T': { 'Qtr1':5000,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df['WIPRO'].min())
10 print(df.min(axis=0))
```

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   5000
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   1800   2100
1800
TCS       2000
WIPRO     1800
L&T       2100
dtype: int64
```

3-count()

It returns the number of values present in a column of a data frame or series.

Syntax-

df['columnname'].count()

Or

df.count(axis=0) ⟶ returns the number of value in each column

Or

df.count(axis=1) ⟶ returns the number of value in each row

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':1800},
5
6          'L&T': { 'Qtr1':5000,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df['WIPRO'].count())
10 print(df.count(axis=0))
```

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   5000
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   1800   2100
4
TCS      4
WIPRO    4
L&T      4
dtype: int64
```

4- mean()

It is used to return the arithmetic mean of a given set of numbers, mean of a data frame, mean of a column, mean of rows.

Syntax-

df['columnname'].mean()

Or

df.mean(axis=0) ⟶ returns the mean of each column

Or

df.mean(axis=1) ⟶ returns the mean of each row

```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

       'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':1800},

       'L&T': { 'Qtr1':5000,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df['WIPRO'].mean())
print(df.mean(axis=1))
```

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   5000
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   1800   2100
2650.0
Qtr1     3433.333333
Qtr2     3366.666667
Qtr3    13866.666667
Qtr4     1966.666667
dtype: float64
```

5- sum()

It is used to return the addition of all the values of a particular column of a data frame or a series .

Syntax-

df['columnname'].sum()

Or

df.sum (axis=0) ⟶ returns the sum of each column

Or

df.sum (axis=1) ⟶ returns the sum of each row

```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

       'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':1800},

       'L&T': { 'Qtr1':5000,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df['WIPRO'].sum())
print(df.sum(axis=0))
```

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   5000
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   1800   2100
10600
TCS       9500
WIPRO    10600
L&T      47800
dtype: int64
```

6- median()

It is used to return the middle value or median of a given set of numbers, median of a data frame, median of a column, median of rows.

Syntax-

df['columnname'].median()

Or

df.median(axis=0) ⟶ returns the median of each column

Or

df.median(axis=1) ⟶ returns the median of each row

```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

        'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':1800},

        'L&T': { 'Qtr1':5000,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df['WIPRO'].median())
print(df.median(axis=0))
```

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   5000
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   1800   2100
2600.0
TCS      2250.0
WIPRO    2600.0
L&T      5350.0
dtype: float64
```

7- mode()

It is used to return the mode or most repeated value of a given set of numbers, mode of a data frame, mode of a column, mode of rows.

Syntax-

df['columnname'].mode()

Or

df.mode(axis=0) ⟶ returns the mode of each column

Or

df.mode(axis=1) ⟶ returns the mode of each row

```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

       'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},

       'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df['WIPRO'].mode())
print(df.mode(axis=0))
```

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
0    2400
dtype: int64
    TCS  WIPRO   L&T
0  2000   2400  2100
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

8- quartile()

The word "quartile" is taken from the word "quantile" and the word "quantile" taken from the "quantity". Let us understand this by taking an example-

The 0.35 quantile states that 35% of the observations in the dataset are below a given line. It also states that there are 65% remaining observations are above the line.

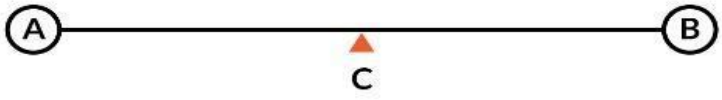
QUARTILE

What is Quartile?

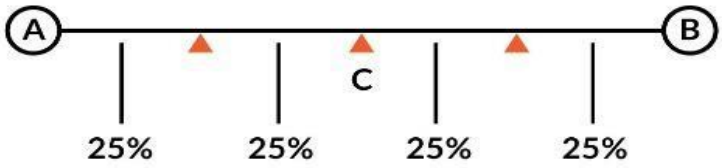
Quartiles in statistics are values that divide your data into quarters.

Suppose we have series of numbers from A - B

Then we divide it from mid say C point



Now again we divide it between A & C then C & B



Now let's understand quartile

Now we can see that the series is divided into 4 equal parts

Q1 is 1st quartile (25th percentile)

Q2 is 2nd quartile (50th percentile)

Also known as median

Q3 is 3rd quartile (75th percentile)

Method to find Quartiles?

Let us take an example: suppose we have numbers- 1,3,4,7,8,8,9

Step 1: Arrange the data in ascending order (already in ascending order)

Step 2: Count total number of observation, say n=7

Step 3: Find out first quartile i.e. Q1 (25%) say 0.25 also called 25^{TH} percentile

Step 4: Now calculate Q1=round (.25(n+1))= round (.25(7+1))
= round (.25(8)) = 2.0 it means 2^{ND} Observation i.e. 3

Step 5: Calculate second quartile i.e. Q2 (50%) = 0.50 or 50^{TH} percentile
= round (.50(7+1)) = 4^{TH} observation i.e. 7

Step 6: Calculate third Quartile i.e. Q3 (75%) =0.75 or 75^{TH} percentile = round (.75(7+1)) = 6^{TH} observation=8

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Program to Find Quartile-

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df.quantile([0.25,0.50,0.75,1.0],axis=0))
10
```

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
        TCS   WIPRO      L&T
0.25  2000.0  2400.0   2100.0
0.50  2250.0  2600.0   3900.0
0.75  2625.0  3000.0  13025.0
1.00  3000.0  3600.0  35000.0
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

9- Variance

It is used to return the variance of a given set of numbers, a data frame, column, rows.

Syntax-

df['columnname'].var()

Or

df.var(axis=0) ⟶ returns the variance of each column

Or

df.var(axis=1) ⟶ returns the variance of each row

```
import pandas as pd
Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},

        'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},

        'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
df=pd.DataFrame(Runs)
print(df)
print(df['WIPRO'].var())
print(df.var(axis=0))
```

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
320000.0
TCS       2.291667e+05
WIPRO     3.200000e+05
L&T       2.541025e+08
dtype: float64
```

10- Standard deviation

It is used to return the standard deviation of a given set of numbers, a data frame, column, rows.

Syntax-

df['columnname'].std()

Or

df.std(axis=0) ⟶ returns the standard deviation of each column

Or

df.std(axis=1) ⟶ returns the standard deviation of each row

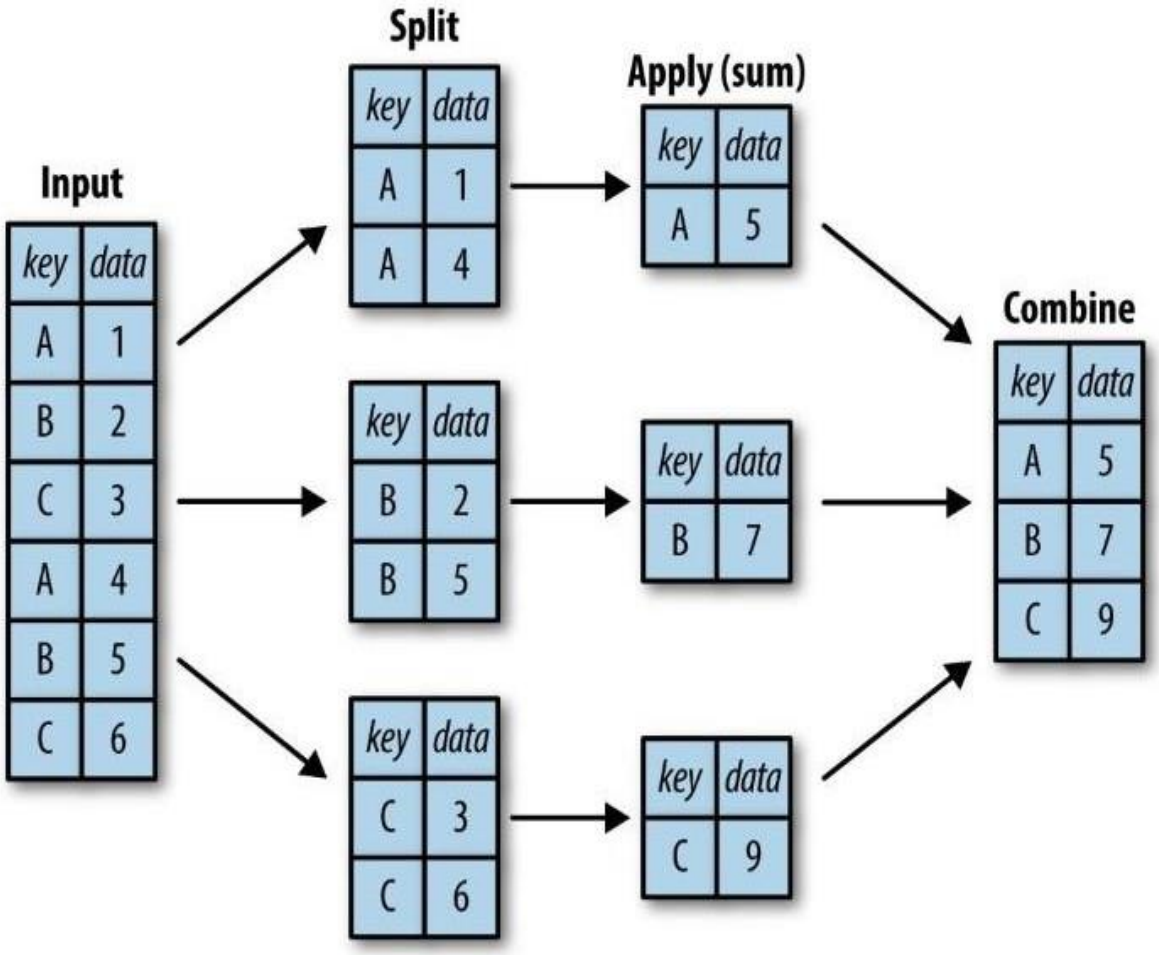
```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df['WIPRO'].std())
10 print(df.std(axis=0))
```

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
565.685424949238
TCS         478.713554
WIPRO       565.685425
L&T       15940.592837
dtype: float64
```

Groupby()

A groupby() function involves one of the following operations on the data frame –

1. Splitting the data frame
2. Applying a function (usually an aggregate function)
3. Combining the result



CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

```
1 import pandas as pd
2 dic={ 'key':['A','B','C','A','B','C'],
3       'data':[1,2,3,4,5,6]}
4 df=pd.DataFrame(dic)
5 print(df)
6 print(df.groupby('key').sum())
```

```
  key  data
0   A     1
1   B     2
2   C     3
3   A     4
4   B     5
5   C     6
     data
key
A       5
B       7
C       9
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Example:- Program to group the data- city wise and find out maximum temperature according to the city.

```
import pandas as pd
data={
        'Date':['1-1-2019','1-1-2019','1-2-2019','1-2-2019','1-3-2019','1-3-2019'],
        'City':['DELHI','DELHI','MUMBAI','MUMBAI','CHENNAI','CHENNAI'],
        'Temp':[28,30,22,24,32,34],
        'Humidity':[60,55,80,70,90,85]
}
df=pd.DataFrame(data)
print (df)
print('\n result after group operation')
print(df.groupby('City').max())
```

```
      Date     City  Temp  Humidity
0  1-1-2019    DELHI    28        60
1  1-1-2019    DELHI    30        55
2  1-2-2019   MUMBAI    22        80
3  1-2-2019   MUMBAI    24        70
4  1-3-2019  CHENNAI    32        90
5  1-3-2019  CHENNAI    34        85
```

28 :-Temp in morning and

30:-Temp in Evening

```
 result after group operation
             Date  Temp  Humidity
City
CHENNAI  1-3-2019    34        90
DELHI    1-1-2019    30        60
MUMBAI   1-2-2019    24        80
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Sorting

Sorting in data frame can be done row wise or column wise. By default sorting is done row wise.

Pandas provide two types of sort functions-

1. sort_values(): To sort the data of a given column in ascending or descending order.
2. sort_index(): To sort the data based on index value.

sort_values() : To sort the data of a given column in ascending or descending order.

Syntax:-

df.sort_values(by='col_name', ascending=True or False, inplace =True or False)

by: Give column name on which you want to perform sorting.

Ascending : By default ascending is true.

Inplace : By default inplace is false. It means if you do not want to create a new data frame then set its value as True.

Example 1- to sort a data frame in ascending order of a column.

For performing sorting in ascending order we do-

df.sort_values ('column name') or

df.sort_values(by='column_name')

```
import pandas as pd
empdata={ 'Empid':[101,102,103,104,105,106],
          'Ename':['Sachin Bhardwaj','Vinod Verma','Lakhbir Singh','Ummed Ali','Rajesh Mishra','UmaSelvi'],
          'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
df=pd.DataFrame(empdata)
print(df)
df=df.sort_values('Ename')
print('\n after sorting')
print(df)

```

```
   Empid           Ename         Doj
0    101  Sachin Bhardwaj  12-01-2012
1    102      Vinod Verma  15-01-2012
2    103    Lakhbir Singh  05-09-2007
3    104        Ummed Ali  17-01- 2012
4    105    Rajesh Mishra  05-09-2007
5    106         UmaSelvi  16-01-2012

 after sorting
   Empid           Ename         Doj
2    103    Lakhbir Singh  05-09-2007
4    105    Rajesh Mishra  05-09-2007
0    101  Sachin Bhardwaj  12-01-2012
5    106         UmaSelvi  16-01-2012
3    104        Ummed Ali  17-01- 2012
1    102      Vinod Verma  15-01-2012
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Example 2- To sort a data frame in descending order of a column.

For performing sorting in descending order we do-

df.sort_values ('column name', ascending=False or (0))

```
1  import pandas as pd
2  empdata={ 'Empid':[101,102,103,104,105,106],
3            'Ename':['Sachin Bhardwaj','Vinod Verma','Lakhbir Singh','Ummed Ali','Rajesh Mishra','UmaSelvi'],
4            'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
5  df=pd.DataFrame(empdata)
6  print(df)
7  df=df.sort_values('Ename', ascending=False)
8  print('\n after sorting')
9  print(df)
10
```

```
   Empid           Ename          Doj
0    101  Sachin Bhardwaj   12-01-2012
1    102      Vinod Verma   15-01-2012
2    103    Lakhbir Singh   05-09-2007
3    104        Ummed Ali  17-01- 2012
4    105    Rajesh Mishra   05-09-2007
5    106         UmaSelvi   16-01-2012

 after sorting
   Empid           Ename          Doj
1    102      Vinod Verma   15-01-2012
3    104        Ummed Ali  17-01- 2012
5    106         UmaSelvi   16-01-2012
0    101  Sachin Bhardwaj   12-01-2012
4    105    Rajesh Mishra   05-09-2007
2    103    Lakhbir Singh   05-09-2007
```

Example 3- To sort a data frame based on multiple column.

For performing sorting based on multiple column we do-

df.sort_values (by=['col1', 'col2'], ascending=[(True or False), (True or False)]

```
1 import pandas as pd
2 data={ 'Rollno':[101,102,103,104,105,106],
3           'Name':['Akash','Mohit','Vinay','Rajeev','Sanjay','Pankaj'],
4           'Percentage':[80,70,64,55,78,78] }
5 df=pd.DataFrame(data)
6 print(df)
7 df=df.sort_values(by=['Percentage','Rollno'], ascending=[True,False])
8 print('\n after sorting')
9 print(df)
```

```
   Rollno    Name  Percentage
0     101   Akash          80
1     102   Mohit          70
2     103   Vinay          64
3     104  Rajeev          55
4     105  Sanjay          78
5     106  Pankaj          78
```

```
 after sorting
   Rollno    Name  Percentage
3     104  Rajeev          55
2     103   Vinay          64
1     102   Mohit          70
5     106  Pankaj          78
4     105  Sanjay          78
0     101   Akash          80
```

As we are sorting the data in ascending order of Percentage so when two values in Percentage are same then data frame will be sorted in descending order of Roll Number.

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Example 4- If you do not want to modify your data frame after sorting.

For this we do-

df.sort_values (by= 'column name', ascending=False or True, inplace=True)

By default inplace is False.

If you do not want to create a new data frame.

```
1 import pandas as pd
2 data={ 'Rollno':[101,102,103,104,105,106],
3           'Name':['Akash','Mohit','Vinay','Rajeev','Sanjay','Pankaj'],
4           'Percentage':[80,70,64,55,78,78] }
5 df=pd.DataFrame(data)
6 print(df)
7 df=df.sort_values(by=['Percentage','Rollno'], ascending=[True,False],inplace=True)
8 print('\n after sorting')
9 print(df)
```

```
   Rollno    Name  Percentage
0     101   Akash          80
1     102   Mohit          70
2     103   Vinay          64
3     104  Rajeev          55
4     105  Sanjay          78
5     106  Pankaj          78

 after sorting
None
```

sort_index()

To sort the data based on index Value.

Syntax:

df.sort_index(by=None, ascending=True or False, inplace =True or False)

by: Give column name on which you want to perform sorting.

Ascending : By default ascending is true.

Inplace : By default inplace is false. It means if you do not want to create a new data frame then set its value as True.

Example 1:- To sort the data frame based on index in ascending order

```
import pandas as pd
data={ 'Rollno':[101,102,103,104,105,106],
          'Name':['Akash','Mohit','Vinay','Rajeev','Sanjay','Pankaj'],
          'Percentage':[80,70,64,55,78,78] }
df=pd.DataFrame(data)
df=df.reindex([5,4,2,3,1,0])
print(df)
df=df.sort_index()
print('\n after sorting')
print(df)
```

```
   Rollno    Name  Percentage
5     106  Pankaj          78
4     105  Sanjay          78
2     103   Vinay          64
3     104  Rajeev          55
1     102   Mohit          70
0     101   Akash          80

 after sorting
   Rollno    Name  Percentage
0     101   Akash          80
1     102   Mohit          70
2     103   Vinay          64
3     104  Rajeev          55
4     105  Sanjay          78
5     106  Pankaj          78
```

Example 2:- To sort the data frame based on index in descending order

```
import pandas as pd
data={ 'Rollno':[101,102,103,104,105,106],
          'Name':['Akash','Mohit','Vinay','Rajeev','Sanjay','Pankaj'],
          'Percentage':[80,70,64,55,78,78] }
df=pd.DataFrame(data)
df=df.reindex([5,4,2,3,1,0])
print(df)
df=df.sort_index(ascending=False)
print('\n after sorting')
print(df)
```

```
   Rollno    Name  Percentage
5     106  Pankaj          78
4     105  Sanjay          78
2     103   Vinay          64
3     104  Rajeev          55
1     102   Mohit          70
0     101   Akash          80

 after sorting
   Rollno    Name  Percentage
5     106  Pankaj          78
4     105  Sanjay          78
3     104  Rajeev          55
2     103   Vinay          64
1     102   Mohit          70
0     101   Akash          80
```

Renaming index

rename () method is used to rename the indexes in a data frame.

Syntax- df.rename (index, inplace (optional))

```
import pandas as pd
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anand Ganesh','Devinder','UmaSelvi'],
        'Doj':['12-01-2012','15-01-2012','05-09-2007','05-09-2007',' 05-09-2007','16-01-2012'] }
df=pd.DataFrame(empdata)
print(df)
df1=df.rename(index={0:'First Name',1:'second Name',2:'Third Name'})
print('Dataframe after renaming the Indexes')
print(df1)
```

```
   empid         ename         Doj
0    101        Sachin   12-01-2012
1    102         Vinod   15-01-2012
2    103       Lakhbir   05-09-2007
3    104  Anand Ganesh   05-09-2007
4    105      Devinder   05-09-2007
5    106      UmaSelvi   16-01-2012
Dataframe after renaming the Indexes
             empid         ename         Doj
First Name     101        Sachin   12-01-2012
second Name    102         Vinod   15-01-2012
Third Name     103       Lakhbir   05-09-2007
3              104  Anand Ganesh   05-09-2007
4              105      Devinder   05-09-2007
5              106      UmaSelvi   16-01-2012
```

Deleting index

reset_index().drop() method is used to delete the indexes in a data frame.

Syntax- df. reset_index().drop(index, inplace (optional))

```
import pandas as pd
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anand Ganesh','Devinder','UmaSelvi'],
        'Doj':['12-01-2012','15-01-2012','05-09-2007','05-09-2007',' 05-09-2007','16-01-2012'] }
df=pd.DataFrame(empdata)
print(df)
df1=df.reset_index().drop(index=[2,3])
print('Dataframe after Deleting the Indexes')
print(df1)
```

```
   empid         ename         Doj
0    101        Sachin   12-01-2012
1    102         Vinod   15-01-2012
2    103       Lakhbir   05-09-2007
3    104  Anand Ganesh   05-09-2007
4    105      Devinder   05-09-2007
5    106      UmaSelvi   16-01-2012
Dataframe after Deleting the Indexes
   index  empid     ename         Doj
0      0    101    Sachin   12-01-2012
1      1    102     Vinod   15-01-2012
4      4    105  Devinder   05-09-2007
5      5    106  UmaSelvi   16-01-2012
```

PIVOTING AND AGGREGATION

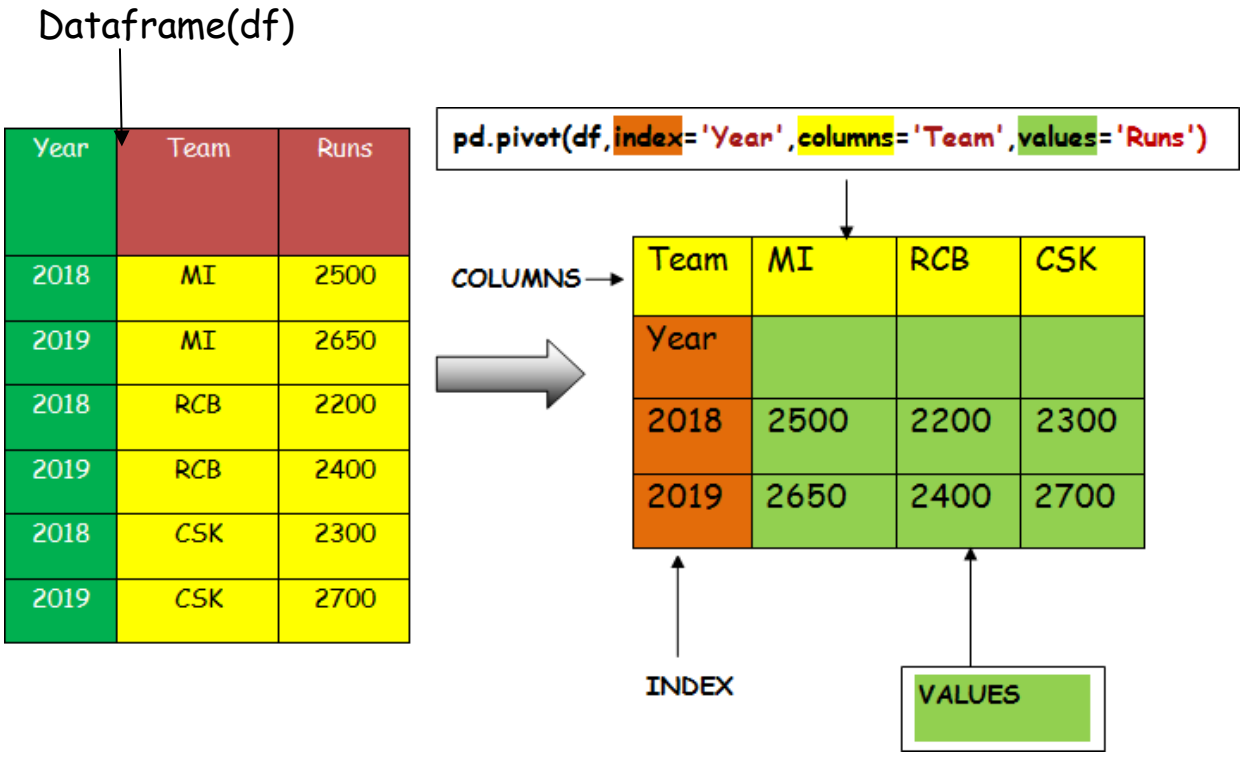
Pivoting- Pivoting is one of the important aspect of data analyst. It is used to summarize large amount of data and permit us to access important records from a large dataset.

Python Pandas provide two functions for pivoting.

1. pivot()
2. pivot-table()

pivot()

pivot()- pivot() allows us to transform or reshape the data frame based on the column values according to our perspective. It takes 3 arguments – (index, columns and values).



Pivoting and Aggregation

```
import pandas as pd
data={
      'Year':['2018','2019','2018','2019','2018','2019'],
      'Team':['MI','MI','RCB','RCB','CSK','CSK'],

      'Runs':[2500,2650,2200,2400,2300,2700]}

df=pd.DataFrame(data)
pv=pd.pivot(df,index='Year',columns='team',values='Runs')
print (df)
print (pv)
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Output-

```
   Year Team   Runs
0  2018    MI   2500
1  2019    MI   2650
2 2018  RCB    2200
3 2019  RCB    2400
4 2018  CSK    2300
5 2019  CSK    2700
```

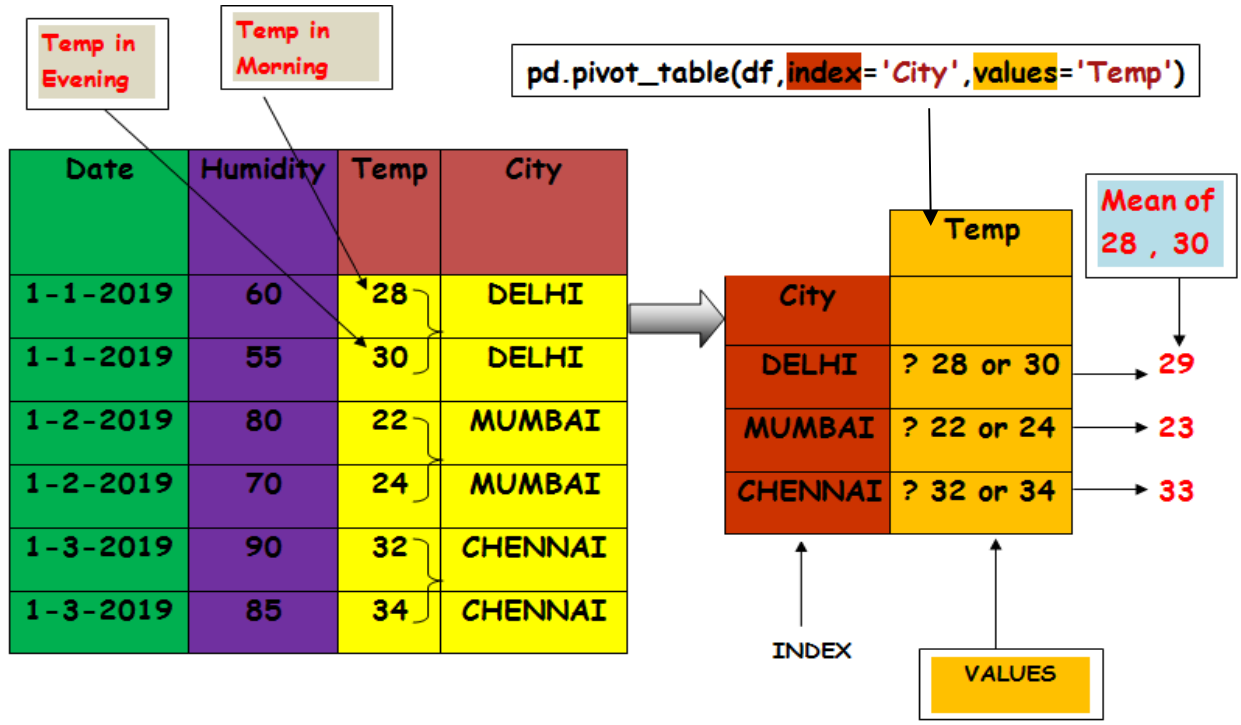
```
Team   CSK   MI   RCB
Year
2018 2300 2500  2200
2019 2700 2650  2400
```

pivot_table()

pivot_table() :- we know that pivot() method takes at least 2 column names as parameters - the index and the columns named parameters. What will happen if we have multiple rows with the same values for these columns.

The pivot_table() method comes to solve this problem. It works like pivot, but it aggregates the values from rows with duplicate entries for the specified columns (means apply aggregate function specify by us).

By default pivot_table() apply mean() to aggregate the values from rows with duplicate entries for the specified columns. E.g.



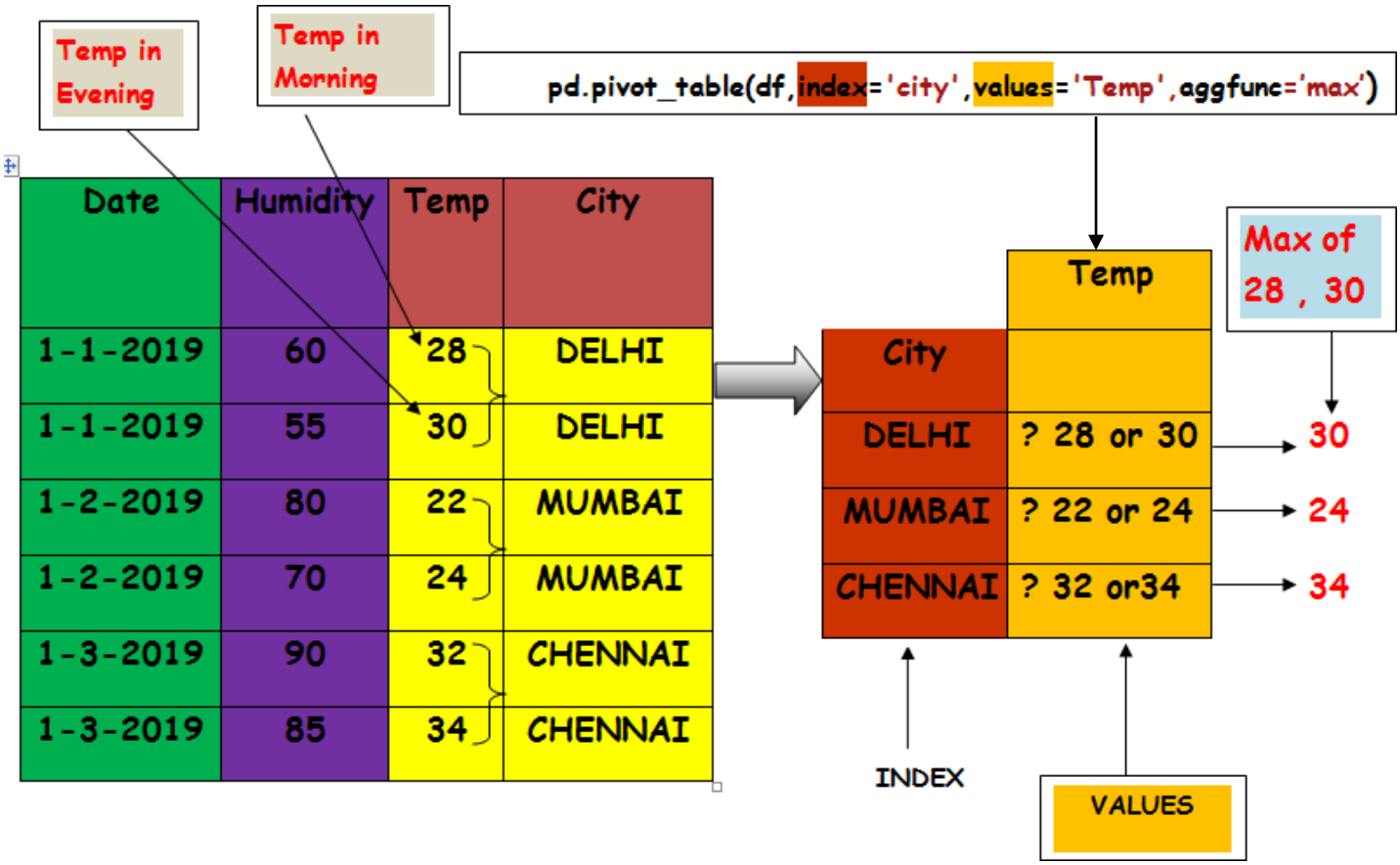

```
#program to Find City wise temperature
Program-
import pandas as pd
data={
      'Date':['1-1-2019','1-1-2019','1-2-2019','1-2-2019','1-3-2019','1-
      3-2019'],
```

```
        'City':['DELHI','DELHI','MUMBAI','MUMBAI','CHENNAI','CH
        ENNAI'],

        'Temp':[28,30,22,24,32,34],
        'Humidity':[60,55,80,70,90,85]
        }
df=pd.DataFrame(data)
print (df)
pv=pd.pivot_table(df,index='City',values='Temp')
print (pv)
```

Output-

```
      Date  Humidity  Temp       City
0  1-1-2019        60       28     DELHI
1  1-1-2019        55       30     DELHI
2  1-2-2019        80       22    MUMBAI
3  1-2-2019        70       24    MUMBAI
4  1-3-2019        90       32  CHENNAI
5  1-3-2019        85       34  CHENNAI
                Temp
City
CHENNAI         33
DELHI           29
MUMBAI          23
```




```
#Program to find City Wise Maximum temperature

import pandas as pddata={

    'Date':['1-1-2019','1-1-2019','1-2-2019','1-2-2019','1-3-2019','1-3-2019'],


    'city':['DELHI','DELHI','MUMBAI','MUMBAI','CHENNAI','CHENNAI'],

    'Temp':[28,30,22,24,32,34],

    'Humidity':[60,55,80,70,90,85]
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

```
      }
df=pd.DataFrame(data)
```

```
print (df)
pv=pd.pivot_table(df,index='city',values='Temp', aggfunc='max')
print (pv)
```

Output-

```
        Date  Humidity  Temp       city
0  1-1-2019        60     28      DELHI
1  1-1-2019        55     30      DELHI
2  1-2-2019        80     22     MUMBAI
3  1-2-2019        70     24     MUMBAI
4  1-3-2019        90     32    CHENNAI
5  1-3-2019        85     34    CHENNAI
```

```
              Temp
city
DELHI           30
MUMBAI          24
CHENNAI         34
```

```
#Program to print data frame on date index and city column
Example 3-
import pandas as pd
data={
            'Date':['1-1-2019','1-1-2019','1-2-2019','1-2-2019','1-3-
            2019','1-3-2019'],

            'city':['DELHI','DELHI','MUMBAI','MUMBAI','CHENNAI
            ','CHENNAI'],
           'Temp':[28,30,22,24,32,34],

            'Humidity':[60,55,80,70,90,85]

      }

df=pd.DataFrame(data)

print (df)

print(pd.pivot_table(df,index='Date',columns='city'))
```

Output-

	Date	Humidity	Temp	city
0	1-1-2019	60	28	DELHI
1	1-1-2019	55	30	DELHI
2	1-2-2019	80	22	MUMBAI
3	1-2-2019	70	24	MUMBAI
4	1-3-2019	90	32	CHENNAI
5	1-3-2019	85	34	CHENNAI

	Humidity			Temp		
city	CHENNAI	DELHI	MUMBAI	CHENNAI	DELHI	MUMBAI
Date						
1-1-2019	NaN	57.5	NaN	NaN	29.0	NaN
1-2-2019	NaN	NaN	75.0	NaN	NaN	23.0
1-3-2019	87.5	NaN	NaN	33.0	NaN	NaN

Not a Number or a Missing Value

Handling Missing Values- filling & Dropping

In many cases, the data that we receive from many sources may not be perfect. That means there may be some missing data. For example- in the given program where employee name is missing in one row and date of joining is missing in other row.

```
import pandas as pd
import numpy as np
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir',np.nan,'Devinder','UmaSelvi'],
        'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012',np.nan,'16-01-2012']}
df=pd.DataFrame(empdata)
print(df)
```

```
   empid     ename         Doj
0    101    Sachin  12-01-2012
1    102     Vinod  15-01-2012
2    103   Lakhbir  05-09-2007
3    104       NaN  17-01- 2012
4    105  Devinder         NaN
5    106  UmaSelvi  16-01-2012
```

When we convert the data into data frame, the missing data is represented **by NaN (Not a Number). NaN** is a default marker for the missing value.

Consider the following Data Frame-

We can use fillna() method to replace NaN or Na value by a specified value.

For example- to fill the Nan value by 0.

```
import pandas as pd
import numpy as np
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir',np.nan,'Devinder','UmaSelvi'],
        'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012',np.nan,'16-01-2012']}
df=pd.DataFrame(empdata)
df=df.fillna(0)
print(df)
```

```
   empid     ename         Doj
0    101    Sachin  12-01-2012
1    102     Vinod  15-01-2012
2    103   Lakhbir  05-09-2007
3    104         0  17-01- 2012
4    105  Devinder           0
5    106  UmaSelvi  16-01-2012
```

But this is not useful as it is filling any type of column with 0. We can fill each column with a different value by passing the column name and the value to be used to fill in that column.

For example- to fill 'ename' with 'Name Missing' and 'Doj' wityh '00-00-0000'. We should supply these values as a dictionary inside fillna() method.

```
import pandas as pd
import numpy as np
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir',np.nan,'Devinder','UmaSelvi'],
        'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012',np.nan,'16-01-2012']}
df=pd.DataFrame(empdata)
df=df.fillna({'ename':'Name Missing','Doj':'00-00-0000'})
print(df)
```

```
   empid         ename         Doj
0    101        Sachin  12-01-2012
1    102         Vinod  15-01-2012
2    103       Lakhbir  05-09-2007
3    104  Name Missing  17-01- 2012
4    105      Devinder  00-00-0000
5    106      UmaSelvi  16-01-2012
```

If we do not want any missing data and want to remove those rows having Na or NaN values, then we can use dropna() method.

```
import pandas as pd
import numpy as np
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir',np.nan,'Devinder','UmaSelvi'],
        'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012',np.nan,'16-01-2012']}
df=pd.DataFrame(empdata)
df=df.dropna()
print(df)
```

```
   empid     ename        Doj
0    101    Sachin  12-01-2012
1    102     Vinod  15-01-2012
2    103   Lakhbir  05-09-2007
5    106  UmaSelvi  16-01-2012
```

Importing-Exporting Data between MySql and Python Pandas

For importing and exporting data between Mysql and Python Pandas we need to install mysql connector and mysql client module.

Installing and importing mysql connector, mysql client-

With Anaconda : if we have installed python using Anaconda, then mysql connector and mysql client need to be installed on your computer. We can check this in Anaconda Navigator, by Clicking on not installed in Environment and then scroll down to find mysql connector and mysql client and by clicking on both these, install them in Anaconda.

Steps to import and export data using pandas and Mysql

1. Start Python
2. import mysql.connector package
3. Create or open a database
4. Open and establish a connection to the database
5. Create a cursor object or its instance (required for Pandas to Mysql)
6. Read a sql query for (Mysql to Pandas) and execute a query for(Pandas to Mysql)
7. Commit the transaction for(Pandas to Mysql)

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

8. Close the connection for(Pandas to Mysql)

Exporting Data between Python Pandas & Mysql

Program 1- To insert and Delete record in MySql from Pandas data frame.

Before execution of the program employee table contains no record.

```
mysql> select * from employee;
Empty set (0.00 sec)
```

In [8]:
```
import mysql.connector
import pandas as pd
con=mysql.connector.connect(host="localhost",user="root",passwd="root",database="sachin")
print(con)
c=con.cursor()
print(df)
c.execute("delete from employee")
con.commit()
for(row,rs) in df.iterrows():
    empid=str(int(rs[0]))
    ename=rs[1]
    Doj=(rs[2])
    c.execute("insert into employee values("+ empid +",'"+ ename +"','"+ Doj +"')")
con.commit()
c.close()
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi'],
          'Doj':['2012-01-12','2012-01-15','2007-09-05','2012-01-17','2007-09-05','2012-01-16'] }
df=pd.DataFrame(empdata)
print("Dta transfer Successfully")
```

For extracting data from data frame into different columns

For casting integer to string

```
<mysql.connector.connection.MySQLConnection object at 0x000001F78BC5A828>
   empid     ename        Doj
0    101    Sachin  2012-01-12
1    102     Vinod  2012-01-15
2    103   Lakhbir  2007-09-05
3    104      Anil  2012-01-17
4    105  Devinder  2007-09-05
5    106  UmaSelvi  2012-01-16
Dta transfer Successfully
```

After the execution of the program the records in employee table are-

```
mysql> select * from employee;
+-------+----------+------------+
| empid | ename    | Doj        |
+-------+----------+------------+
|   101 | Sachin   | 2012-01-12 |
|   102 | Vinod    | 2012-01-15 |
|   103 | Lakhbir  | 2007-09-05 |
|   104 | Anil     | 2012-01-17 |
|   105 | Devinder | 2007-09-05 |
|   106 | UmaSelvi | 2012-01-16 |
+-------+----------+------------+
6 rows in set (0.05 sec)
```

Example 2-

To perform Update operation in MySql from Pandas data frame.

```
In [18]: import mysql.connector
         import pandas as pd
         con=mysql.connector.connect(host="localhost",user="root",passwd="root",database="sachin")
         print(con)
         c=con.cursor()
         q="update employee set ename= 'Sachin Bhardwaj' where empid=101"
         c.execute(q)
         con.commit()
         c.close()
         print('\n Update Operation Performed Successfully')
```

```
<mysql.connector.connection.MySQLConnection object at 0x000001F78C01EE80>

 Update Operation Performed Successfully
```

After the execution of the program the record in employee table got updated from Sachin to Sachin Bhardwaj-

```
mysql> select * from employee;
+-------+----------+------------+
| empid | ename    | Doj        |
+-------+----------+------------+
|   101 | Sachin   | 2012-01-12 |
|   102 | Vinod    | 2012-01-15 |
|   103 | Lakhbir  | 2007-09-05 |
|   104 | Anil     | 2012-01-17 |
|   105 | Devinder | 2007-09-05 |
|   106 | UmaSelvi | 2012-01-16 |
+-------+----------+------------+
6 rows in set (0.00 sec)

mysql> select * from employee;
+-------+-----------------+------------+
| empid | ename           | Doj        |
+-------+-----------------+------------+
|   101 | Sachin Bhardwaj | 2012-01-12 |
|   102 | Vinod           | 2012-01-15 |
|   103 | Lakhbir         | 2007-09-05 |
|   104 | Anil            | 2012-01-17 |
|   105 | Devinder        | 2007-09-05 |
|   106 | UmaSelvi        | 2012-01-16 |
+-------+-----------------+------------+
6 rows in set (0.03 sec)

mysql>
```

Importing Data between Python Pandas & Mysql

Example 1- To retrieve column empid and Doj from employee table into data frame emp.

In [9]:
```
import mysql.connector
import pandas as pd
con=mysql.connector.connect(host="localhost",user="root",passwd="root",database="sachin")
print(con)
emp=pd.read_sql_query("select empid,Doj from employee",con)
emp
```

```
<mysql.connector.connection.MySQLConnection object at 0x000001F789469940>
```

Out[9]:

	empid	Doj
0	101	2012-01-12
1	102	2012-01-15
2	103	2007-09-05
3	104	2012-01-17
4	105	2007-09-05
5	106	2012-01-16

Example -2

To retrieve all the tables from database sachin into data frame emp.

```
In [1]: import pandas as pd
```

```
In [2]: import mysql.connector
```

```
In [5]: con=mysql.connector.connect(host="localhost",user="root",passwd="root",database="sachin")
        print(con)
```

```
<mysql.connector.connection.MySQLConnection object at 0x0000000009C1D7F0>
```

```
In [6]: emp=pd.read_sql_query("show tables from sachin",con)
        emp
```

Out[6]:

	Tables_in_sachin
0	employee

Importing-Exporting Data between MySql and Python Pandas USING Sqlalchemy

Sqlalchemy is a database manipulation tool for python which can be used as standalone library to manipulate relational databases. Sqlalchemy provide core python based sql expressions and object oriented python based ORM (Object Relational Mapper). it also provide high level declarative syntax for ORM for simplicity.

Sqlalchemy follow data mapper pattern and inspired from java hibernate. To work with sqlalchemy first of all we need to install following library:

1. Slalchemy (-m pip install sqlalchemy)
2. PyMySQL (-m pip install PyMySQL)

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Importing Data between Python Pandas & MySQL using sqlalchemy

```
import pandas as pd
import sqlalchemy
con=sqlalchemy.create_engine('mysql+pymysql://root:123@localhost/sachin')
df=pd.read_sql("record",con)
print(df)
```

```
    id       empname        dob
0  101        Sachin 1987-08-17
1  102          Anil 1987-08-19
2  103  Anand Ganesh 1980-02-10
```

In Above program-

User Name of MYSQL is- root

Password of MYSQL is- 123

Database in MYSQL is- sachin

Table from which records are fetched is- record

that is already created in MYSQL with 3 records.

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Importing Data between Python Pandas & MySQL using sqlalchemy based on specific Columns

```
import pandas as pd
import sqlalchemy
con=sqlalchemy.create_engine('mysql+pymysql://root:123@localhost/sachin')
df=pd.read_sql("record",con,columns=['empname'])
print(df)
```

```
        empname
0        Sachin
1          Anil
2  Anand Ganesh
```

Importing Data between Python Pandas & MySQL using sqlalchemy based on specific Condition

```
import pandas as pd
import sqlalchemy
con=sqlalchemy.create_engine('mysql+pymysql://root:123@localhost/sachin')
df=pd.read_sql("select * from record where empname='Sachin'",con)
print(df)
```

```
    id empname         dob
0  101  Sachin  1987-08-17
```

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

Exporting Data between Python Pandas & Mysql using sqlalchemy

```
import pandas as pd
import sqlalchemy
con=sqlalchemy.create_engine('mysql+pymysql://root:123@localhost/sachin')
df = pd.DataFrame({"Name":['Hardik Pandya','Virat Kohli','K L Rahul','Rohit Sharma'],
                   "IPLTeam":['MI','RCB','XI PUNJAB','MI'],
                   "Runs":[1500,4500,2400,4450]})
print(df)
df.to_sql("ipl",con=con,if_exists="replace")
```

```
            Name    IPLTeam  Runs
0  Hardik Pandya         MI  1500
1    Virat Kohli        RCB  4500
2      K L Rahul  XI PUNJAB  2400
3   Rohit Sharma         MI  4450
```

The to_sql() function is used to write the records stored in a DataFrame to a SQL Table.

After the execution of the above program

MYSQL database sachin looks like:

CREATED BY: SACHIN BHARDWAJ, PGT (CS) KV NO.1 TEZPUR, MR. VINOD KUMAR VERMA, PGT (CS) KV OEF KANPUR

```
mysql> show tables;
+------------------+
| Tables_in_sachin |
+------------------+
| ipl              |
| record           |
+------------------+
2 rows in set (0.00 sec)

mysql> select * from ipl;
+-------+---------------+-----------+------+
| index | Name          | IPLTeam   | Runs |
+-------+---------------+-----------+------+
|     0 | Hardik Pandya | MI        | 1500 |
|     1 | Virat Kohli   | RCB       | 4500 |
|     2 | K L Rahul     | XI PUNJAB | 2400 |
|     3 | Rohit Sharma  | MI        | 4450 |
+-------+---------------+-----------+------+
4 rows in set (0.00 sec)
```

Example-2

```
import pandas as pd
import sqlalchemy
con=sqlalchemy.create_engine('mysql+pymysql://root:123@localhost/sachin')
df = pd.DataFrame({"Name":['Hardik Pandya','Virat Kohli','K L Rahul','Rohit Sharma'],
                   "IPLTeam":['MI','RCB','XI PUNJAB','MI'],
                   "Runs":[1500,4500,2400,4450]})
print(df)
df.to_sql("ipl",con=con,if_exists="append",index=False)
```

```
           Name    IPLTeam  Runs
0  Hardik Pandya         MI  1500
1    Virat Kohli        RCB  4500
2      K L Rahul  XI PUNJAB  2400
3   Rohit Sharma         MI  4450
```

After the execution of the above program

MYSQL ipl tables looks like:

```
mysql> select * from ipl;
+-------+---------------+-----------+------+
| index | Name          | IPLTeam   | Runs |
+-------+---------------+-----------+------+
|     0 | Hardik Pandya | MI        | 1500 |
|     1 | Virat Kohli   | RCB       | 4500 |
|     2 | K L Rahul     | XI PUNJAB | 2400 |
|     3 | Rohit Sharma  | MI        | 4450 |
|  NULL | Hardik Pandya | MI        | 1500 |
|  NULL | Virat Kohli   | RCB       | 4500 |
|  NULL | K L Rahul     | XI PUNJAB | 2400 |
|  NULL | Rohit Sharma  | MI        | 4450 |
+-------+---------------+-----------+------+
8 rows in set (0.00 sec)
```