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.

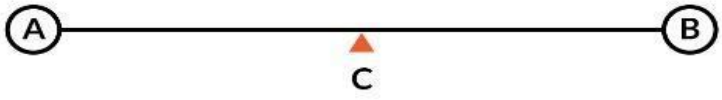
QUARTILE

What is Quartile?

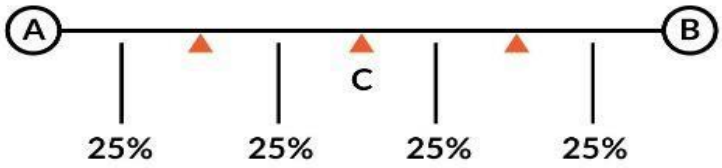
Quartiles in statistics are values that divide your data into quarters.

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

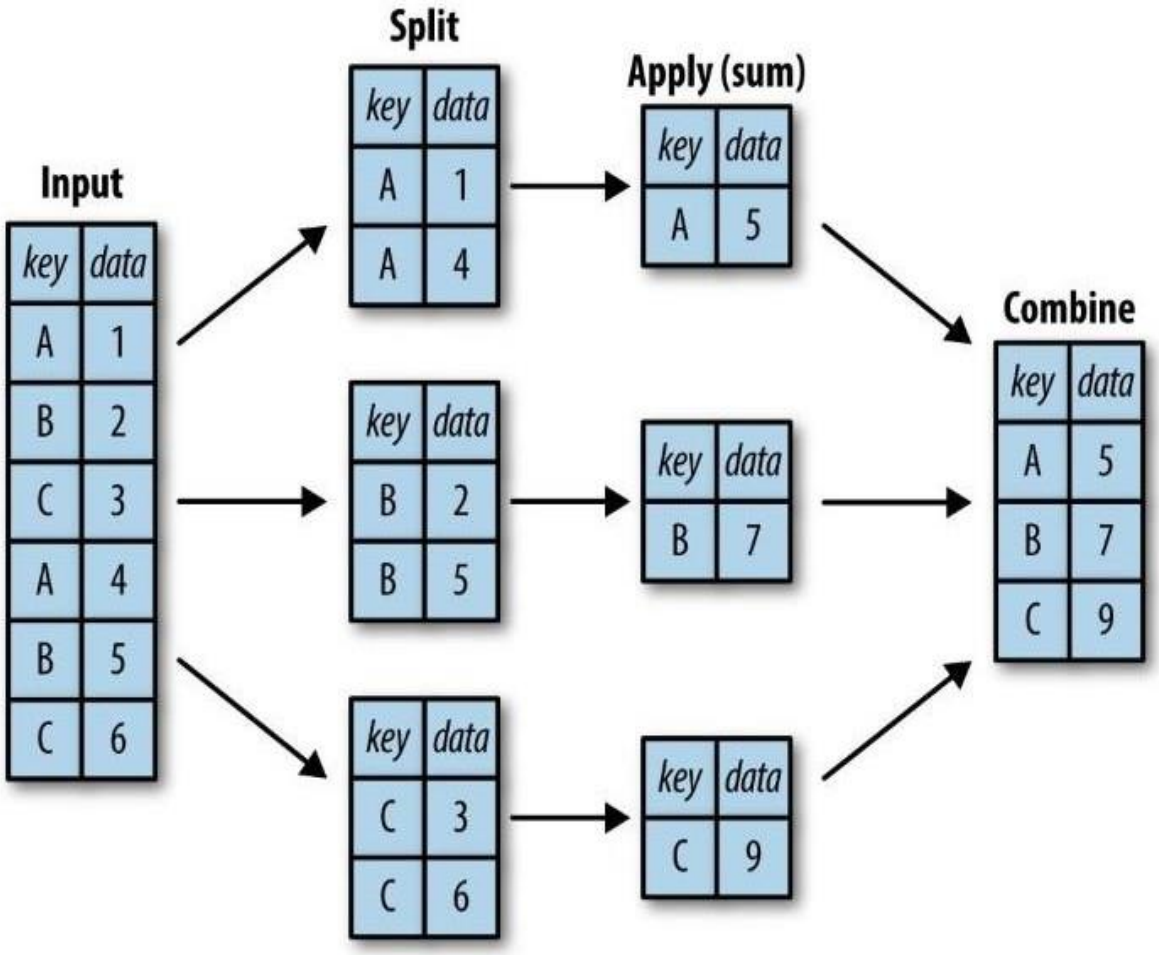
```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df['WIPRO'].std())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

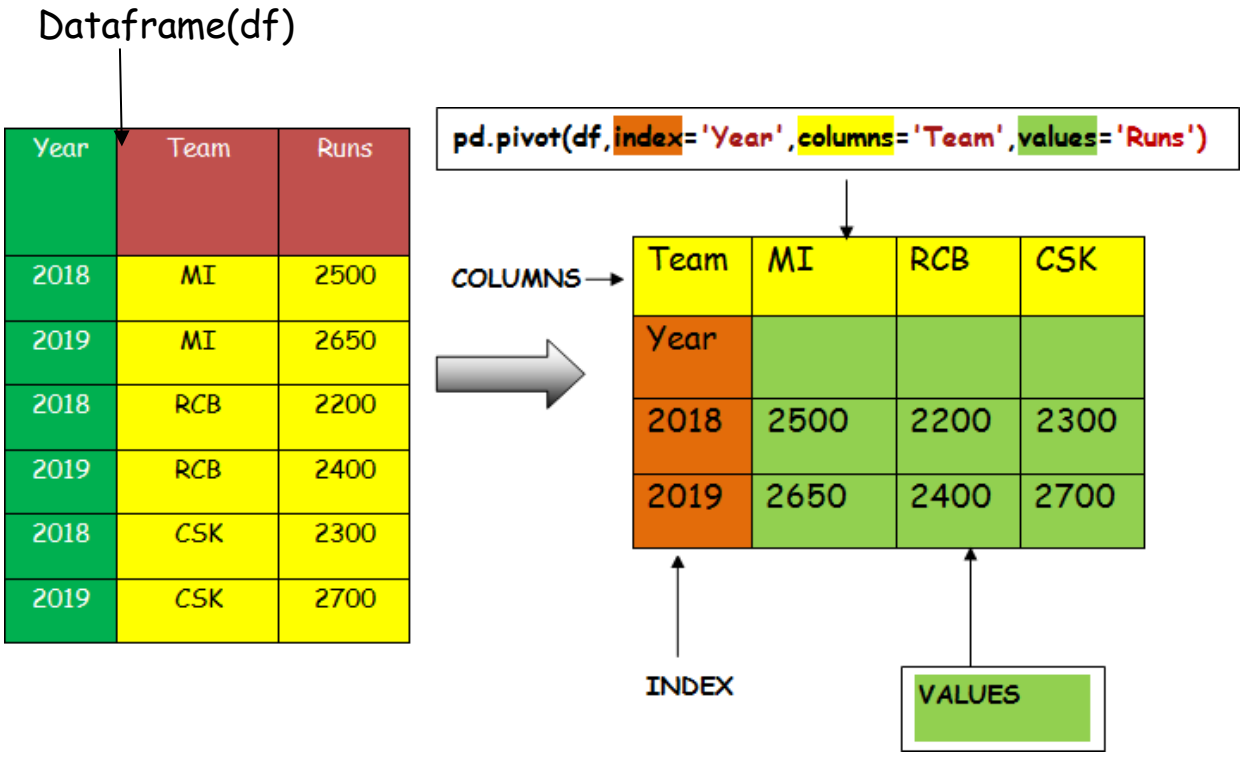
Pivoting- Pivoting is one of the important aspect of data analyst. It is used to summarize large amount of data and permit us to access important records from a large dataset.

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
```

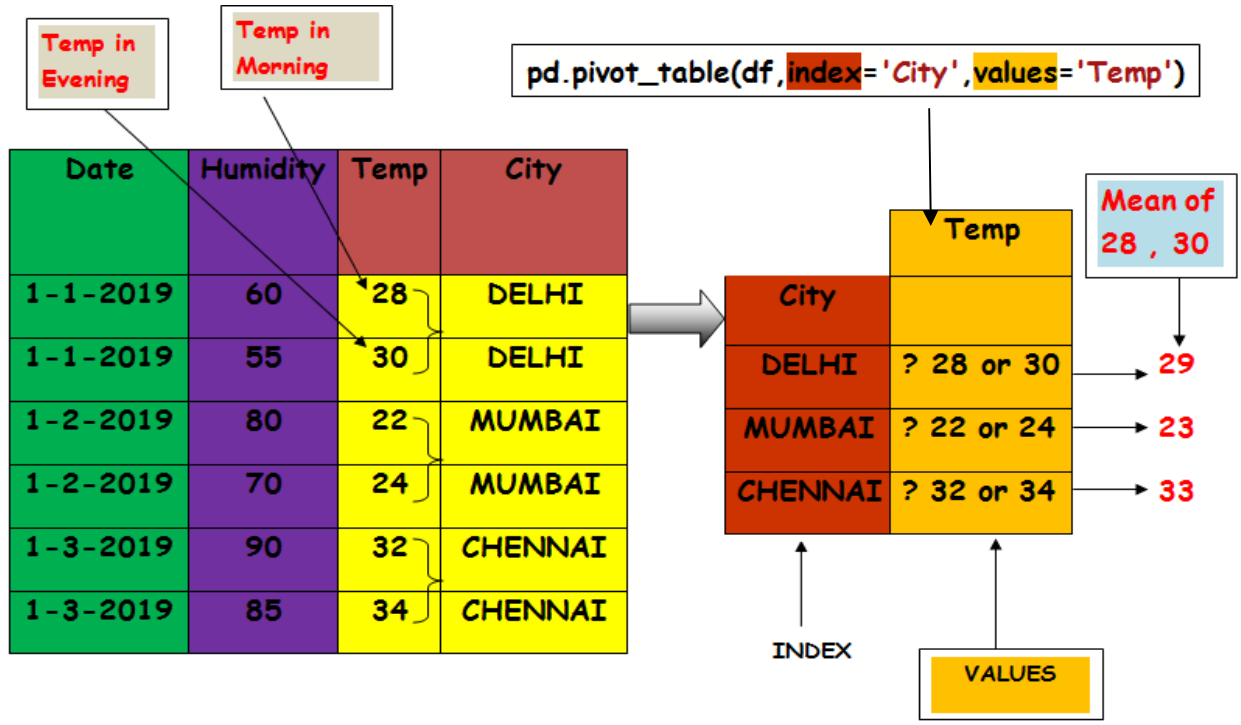
```
Team   CSK   MI   RCB
Year
2018 2300 2500  2200
2019 2700 2650  2400
```

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.



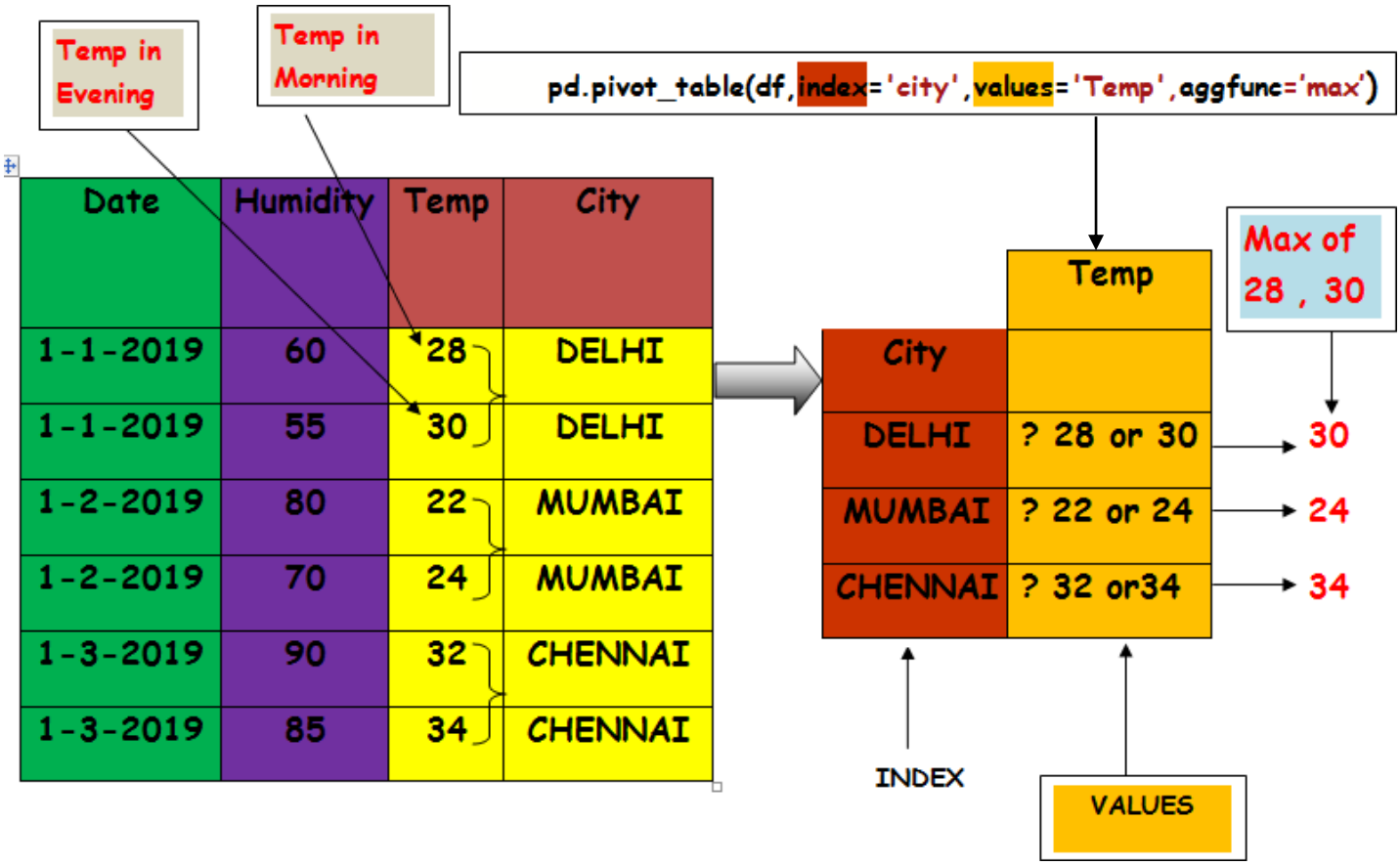

```
#program to Find City wise temperature
Program-
import pandas as pd
data={
      'Date':['1-1-2019','1-1-2019','1-2-2019','1-2-2019','1-3-2019','1-
      3-2019'],
```

```
        '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)
```