Chapter Pandas-II Python Pandas(II)

"DataFrame--- A dataframe is a two-dimensional labeled array like, pandas data structure that stores an ordered collection columns that can store data of different types.

" DataFrames are both, value-mutable and size-mutable, i.e we can change both its value and size".

Characteristics

- It has two indexes or we can say that two axes-a row index(axis=0) and a column index(axis=1).
- Conceptually it is like spreadsheet where each value is identifiable with the combination of row index and column index. The row index is known as **index in general and the column index is called the column-name.**
- The indexes can be of numbers or letters or string.
- There is no condition of having all data of same type across columns; its columns can have data of different types .
- We can easily change its values, i.e it is value-mutable.
- We can add or delete rows/columns in a DataFrame. In other words, it is size-mutable.

Creating and Displaying a DataFrame

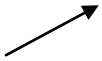
```
In [1]: import pandas as pd
In [2]: population=pd.Series([1234,5674,7867,9834],
index=['Delhi','guwahati','kolkata','chennai'])
In [3]:
avgincome=pd.Series([21,27,34,26],index=['Delhi','guwahati','kolkata','che
nnai'])
In [4]: percpita=avgincome/population
In [5]:
dict1={'population':population,'avgincome':avgincome,'percpita':percpita}
In [7]: df=pd.DataFrame(dict1)
In [8]: df
Out[8]:
population avgincome percpita
Delhi      1234 21   0.017018
guwahati   5674 27   0.004759
kolkata    7867 34   0.004322
chennai    9834 26   0.002644
```

Creating and Displaying a DataFrame

How to create DataFrame from Series

Program

```
import pandas as pd

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

df=pd.DataFrame(s)
```



Output **(Default Column as 0)**

Creating and Displaying a DataFrame

How to create DataFrame from Dictionary

Program

```
import pandas as pd

1=[{'Name':'Sachin', 'SriName':'Bhardwaj'},

    {'Name':'Vinod','SriName':'Verma'},

    {' Name':'Rajesh','SriName':'Mishra''}]

Df1=pd.DataFrame(1)
```

	Name	SirName
0	Sachin	Bhardwaj
1	Vinod	Verma
2	Rajesh	Mishra

output

Creating and Displaying a DataFrame (iterrows())

Program

```
import  pandas as pd
1=[{'Name':'Sachin', 'SriName':'Bhardwaj'},
    {'Name':'Vinod','SriName':'Verma'},
    {' Name':'Rajesh','SriName':'Mishra''}]
Df1=pd.DataFrame(1)
for(row_index,row_value) in df1.iterrows():
   print('\n Row index is ::',row_index)
   print('Row Value is::')
   print(row_value)
```

Output

	Name	SirName
0	Sachin	Bhardwaj
1	Vinod	Verma

Row index is::0

Row value is::

Name Sachin

SirName Bhardwaj

Name:0, dtype:object

Creating and Displaying a DataFrame (iteritems())

Program

```
import  pandas as pd
1=[{'Name':'Sachin', 'SriName':'Bhardwaj'},
    {'Name':'Vinod','SriName':'Verma'},
    {' Name':'Rajesh','SriName':'Mishra''}]
Df1=pd.DataFrame(1)
for(col_name,col_value) in df1.iteritems():
   print('\n )
   print ('Column Name is::',col_name)
   print('Column Values are::')
```

Output

	Name	SirName
0	Sachin	Bhardwaj
1	Vinod	Verma

Column Name is ::Name

Column value are::

0 Sachin

1 Vinod

Name: Name, dtype:object

Select operation in data frame

To access the column data, we can mention the column name as subscript.
Eg.-**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)
```

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

>>>**df.empid or df['empid']**

0	101
1	102
2	103
3	104
4	105
5	106

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 Column 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

```
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']
```

Output-

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

Output

0

1

>>del df[‘List3’] We can simply delete a column by passing column name in subscript with df

>>df.pop(‘List2’) we can simply delete a column by passing column name in pop method.

Output-

	List1
0	10
1	15

To Delete a Column in data frame

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

	List 1	List2
0	10	40
1	20	40
2	30	40
3	40	40

After deletion::

	List 1
0	10
1	20
2	30
3	40

After row deletion::

	List 1
0	10
1	20

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.

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)
print(df)

print(df.loc[:, 'TCS'])

print(df.loc[ :, 'TCS':'WIPRO'])
```

	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

print(df.loc[: , 'TCS'])

print(df.loc[: , 'TCS' : 'WIPRO'])

Qtr1 2500

Qtr2 2000

Qtr3 3000

Qtr4 2000

	TCS	WIPRO
Qtr1	2500	2800
Qtr2	2000	2400
Qtr3	3000	3600

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)
print(df)
```

print(df.iloc[0:2 , 1: 2]) To access first two Rows and Second column

	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
Qtr3	3000	3600

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

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

In [2]: ssg={'cdtid':[101,102,103,104,105,106],
   ...: 'ename':['ram','mohan','sohan','sonal','Raju','lakhan'],
   ...: 'house':['lohit','lachit','ud','abhi','ani','bha'] }


In [3]: sainik=pd.DataFrame(ssg)


In [4]: print(sainik)


In [5]: print(sainik)
cdtid ename house
0 101 ram lohit
1 102 mohan lachit
2 103 sohan ud
3 104 sonal abhi
4 105 Raju ani
5 106 lakhan bha


In [6]: print(sainik.head(2))
cdtid ename house
0 101 ram lohit
1 102 mohan lachit
```

```
In [7]: print(sainik.head())
cdtid ename house
0 101 ram lohit
1 102 mohan lachit
2 103 sohan ud
3 104 sonal abhi
4 105 Raju ani

In [8]: print(sainik.tail())
cdtid ename house
1 102 mohan lachit
2 103 sohan ud
3 104 sonal abhi
4 105 Raju ani
5 106 lakhan bha

In [9]: print(sainik.tail(3)
   ...: )
cdtid ename house
3 104 sonal abhi
4 105 Raju ani
5 106 lakhan bha
```

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.

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

In [2]: ssg={'cdtid':[101,102,103,104,105,106],
   ...: ...: 'ename':['ram','mohan','sohan','sonal','Raju','lakhan'],
   ...: ...: 'house':['lohit','lachit','ud','abhi','ani','bha'] }

In [3]: sainik=pd.DataFrame(ssg)

In [4]: print(sainik)
cdtid ename house
0 101 ram lohit
1 102 mohan lachit
2 103 sohan ud
3 104 sonal abhi
4 105 Raju ani
```

```
5 106 lakhan bha
In [11]:
booli=pd.DataFrame(ssg,index=[True,False,False,False,True,False,])

In [12]: print(booli)
cdtid ename house
True  101 ram lohit
False 102 mohan lachit
False 103 sohan ud
False 104 sonal abhi
True  105 Raju ani
False 106 lakhan bha
In [15]: print(booli)
cdtid ename house
True  101 ram lohit
False 102 mohan lachit
False 103 sohan ud
False 104 sonal abhi
True  105 Raju ani
False 106 lakhan bha
In [16]: print(booli.loc[True])
cdtid ename house
True 101 ram lohit
True 105 Raju ani
In [17]: print(booli.loc[False])
cdtid ename house
False 102 mohan lachit
False 103 sohan ud
False 104 sonal abhi
False 106 lakhan bha
```

To do 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.

```
In [2]: import pandas as pd

In [3]: dict1={'id':['1','2'],
'value1':['A','B'],'Value2':['B','D']}

In [4]: dict2={'id':['2','3'],
'value1':['k','m'],'Value2':['l','n']}

In [5]: df1=pd.DataFrame(dict1)

In [6]: df2=pd.DataFrame(dict2)

In [7]: df3=pd.concat([df1,df2],axis=1)

In [8]: print(df3)
id value1 Value2 id value1 Value2
0 1 A B 2 k l
1 2 B D 3 m n

In [9]: df3=pd.concat([df1,df2])

_In [10]: df3
Out[10]:
id value1 Value2
0 1 A B
1 2 B D
0 2 k l
1 3 m n

In [11]: df3=pd.concat([df1,df2],axis=1)
```

Merge operation in data frame

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

In [2]:
ic1={'id':['1','2','3','4','5'],'value1':['A','C','E','G','I'],
   ...: 'VALUE1':['B','D','F','H','J']}

In [3]:
dic2={'id':['2','3','6','7','8'],'value1':['K','M','O','Q','S'],
   ...: 'VALUE2':['L','N','P','R','T']}

In [4]: df1=pd.DataFrame(ic1)

In [5]: df2=pd.DataFrame(dic2)

In [6]:
dic2={'id':['2','3','6','7','8'],'value1':['A','M','O','G','S'],
   ...: 'VALUE2':['L','N','P','R','T']}

In [7]: df2=pd.DataFrame(dic2)
In [8]:
dic3={'id':['2','3','6','7','8'],'value1':['A','M','O','G','S'],
   ...: 'VALUE2':['L','N','P','R','T']}

In [9]: df3=pd.DataFrame(dic2)

In [10]: df3=pd.DataFrame(dic3)

In [11]: df4=pd.merge(df1,df3, on="value1")

In [12]: df4
Out[12]:
id_x value1 VALUE1 id_y VALUE2
0 1 A B 2 L
1 4 G H 7 R

In [13]: df1
Out[13]:
id value1 VALUE1
0 1 A B
1 2 C D
2 3 E F
3 4 G H
4 5 I J
```

```
In [14]: df3
Out[14]:
id value1 VALUE2
0 2 A L
1 3 M N
2 6 O P
3 7 G R
4 8 S T

In [15]: df5=pd.merge(df1,df3, on="id")

In [16]: df5=pd.merge(df1,df3, on="value1")
In [18]: df5
Out[18]:
id_x value1 VALUE1 id_y VALUE2
0 1 A B 2 L
1 4 G H 7 R

In [19]: df6=pd.merge(df1,df3, on="value1")

In [20]: df6
Out[20]:
id_x value1 VALUE1 id_y VALUE2
0 1 A B 2 L
1 4 G H 7 R

In [21]: df1
Out[21]:
id value1 VALUE1
0 1 A B
1 2 C D
2 3 E F
3 4 G H
4 5 I J

In [22]: df2
Out[22]:
id value1 VALUE2
0 2 A L
1 3 M N
2 6 O P
3 7 G R
4 8 S T
```

```
In [23]: df3
Out[23]:
id value1 VALUE2
0 2 A L
1 3 M N
2 6 O P
3 7 G R
4 8 S T

In [24]: df7=pd.merge(df1,df3, on="value1")

In [25]: df7
Out[25]:
id_x value1 VALUE1 id_y VALUE2
0 1 A B 2 L
1 4 G H 7 R
```

Join operation in data frame

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.

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

In [2]:
dic1={ 'id':['1','2','3','4','5'],'Value1':['A','C','E','G','I'],'Value2':
['B','D','F','H','J']}

In [3]:
dic2={ 'id':['2','3','6','7','8'],'Value1':['K','M','O','Q','S'],'Value2':
['L','N','P','R','T']}

In [4]: df1=pd.DataFrame(dic1)

In [5]: df2=pd.DataFrame(dic2)

In [6]: df3=pd.merge(df1,df2, on='id',how='outer')
```

```
In [7]: df1
Out[7]:
id Value1 Value2
0 1 A B
1 2 C D
2 3 E F
3 4 G H
4 5 I J

In [8]: df2
Out[8]:
id Value1 Value2
0 2 K L
1 3 M N
2 6 O P
3 7 Q R
4 8 S T

In [9]: df3
Out[9]:
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
```

Join operation in data frame

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.

```
In [2]:
dic1={ 'id':['1','2','3','4','5'],'Value1':['A','C','E','G','I'],'Value2':
['B','D','F','H','J']}


In [3]:
dic2={ 'id':['2','3','6','7','8'],'Value1':['K','M','O','Q','S'],'Value2':
['L','N','P','R','T']}

In [4]: df1=pd.DataFrame(dic1)

In [5]: df2=pd.DataFrame(dic2)
```

```
In [6]: df3=pd.merge(df1,df2, on='id',how='inner')
```

Output

	Id	value_x	Value2_x	Value1_y	Value2_y
0	2	C	D	K	L
1	3	E	F	M	N

Join operation in data frame

RIGHT Join:- 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

```
In [2]:
dic1={ 'id':['1','2','3','4','5'],'Value1':['A','C','E','G','I'],'Value2':
['B','D','F','H','J']}
```

```
In [3]:
dic2={ 'id':['2','3','6','7','8'],'Value1':['K','M','O','Q','S'],'Value2':
['L','N','P','R','T']}
```

```
In [4]: df1=pd.DataFrame(dic1)
In [5]: df2=pd.DataFrame(dic2)
```

```
In [6]: df3=pd.merge(df1,df2, on='id',how='RIGHT')
```

Output

	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

Join operation in data frame

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.

```
In [2]:
dic1={ 'id':['1','2','3','4','5'],'Value1':['A','C','E','G','I'],'Value2':
['B','D','F','H','J']}
```

```
In [3]:
dic2={ 'id':['2','3','6','7','8'],'Value1':['K','M','O','Q','S'],'Value2':
['L','N','P','R','T']}
```

```
In [4]: df1=pd.DataFrame(dic1)
In [5]: df2=pd.DataFrame(dic2)
```

```
In [6]: df3=pd.merge(df1,df2, on='id',how='Left')
print(df3)
```

Output

	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

Joining on Index operation in data frame

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.

```
In [2]:
dic1={ 'id':['1','2','3','4','5'],'Value1':['A','C','E','G','I'],'Value2':
['B','D','F','H','J']}
```

```
In [3]:
dic2={ 'id':['2','3','6','7','8'],'Value1':['K','M','O','Q','S'],'Value2':
['L','N','P','R','T']}

In [4]: df1=pd.DataFrame(dic1)

In [5]: df2=pd.DataFrame(dic2)

In [6]: df3=pd.merge(df1,df2, right_index=True, left_index=True)
print(df3)
```

Output

	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

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.

Descriptive Statistics

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

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)

In [4]: df
Out[4]:
```

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

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [6]: print(df['WIPRO'].max())
3600

In [7]: print(df['L&T'].max())
35000

In [8]: print(df.max(axis=0))
TCS 3000
WIPRO 3600
L&T 35000
dtype: int64

In [9]: print(df.max(axis=1))
Qtr1 5000
Qtr2 5700
Qtr3 35000
Qtr4 2100
dtype: int64
```

Descriptive Statistics

It returns the maximum 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.max(axis=1) returns the minimum value of every row

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)

In [4]: df
Out[4]:
TCS WIPRO L&T
```

```
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].min())
1800

In [11]: print(df['TCS'].min())
2000

In [12]: print(df.min(axis=0))
TCS 2000
WIPRO 1800
L&T 2100
dtype: int64

In [13]: print(df.min(axis=1))
Qtr1 2500
Qtr2 2000
Qtr3 3000
Qtr4 1800
dtype: int64
```

Descriptive Statistics

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.max(axis=1) returns the number of value of every row

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)
```

```
In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].count())
4
print(df.count(axis=0))
TCS   4
WIPRO 4
L& T  4
dtype: int64
```

Descriptive Statistics

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.max(axis=1) returns the number of value of every row

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)

In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
```

```
In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].count())
4
print(df.count(axis=0))
TCS   4
WIPRO 4
L& T  4
dtype: int64
```

Descriptive Statistics

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.max(axis=1) returns the number of value of every row

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)

In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
```

```
In [10]: print(df['WIPRO'].count())
4
print(df.count(axis=0))
TCS   4
WIPRO 4
L& T  4
dtype: int64
```

Descriptive Statistics

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

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)

In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].mean())
2650.0

print(df.mean(axis=1))

Qtr1      3433.333333
Qtr2      3366.666667
Qtr3     13866.666667
Qtr4      1966.666667
```

Dtype:float64

Descriptive Statistics

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

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)


In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].sum())
10600

 print(df.sum(axis=0))

TCS   9500
WIPRO 10600
L&T   47800
Dtype:int64
```

Descriptive Statistics

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 value of every row

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)

In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
```

output

```
In [10]: print(df['WIPRO'].median())
2600.0

print(df.median(axis=0))
```

TCS 2250.0
WIPRO 2600.0
L&T 5350.0
Dtype:int64

Descriptive Statistics

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 value of every column

Or

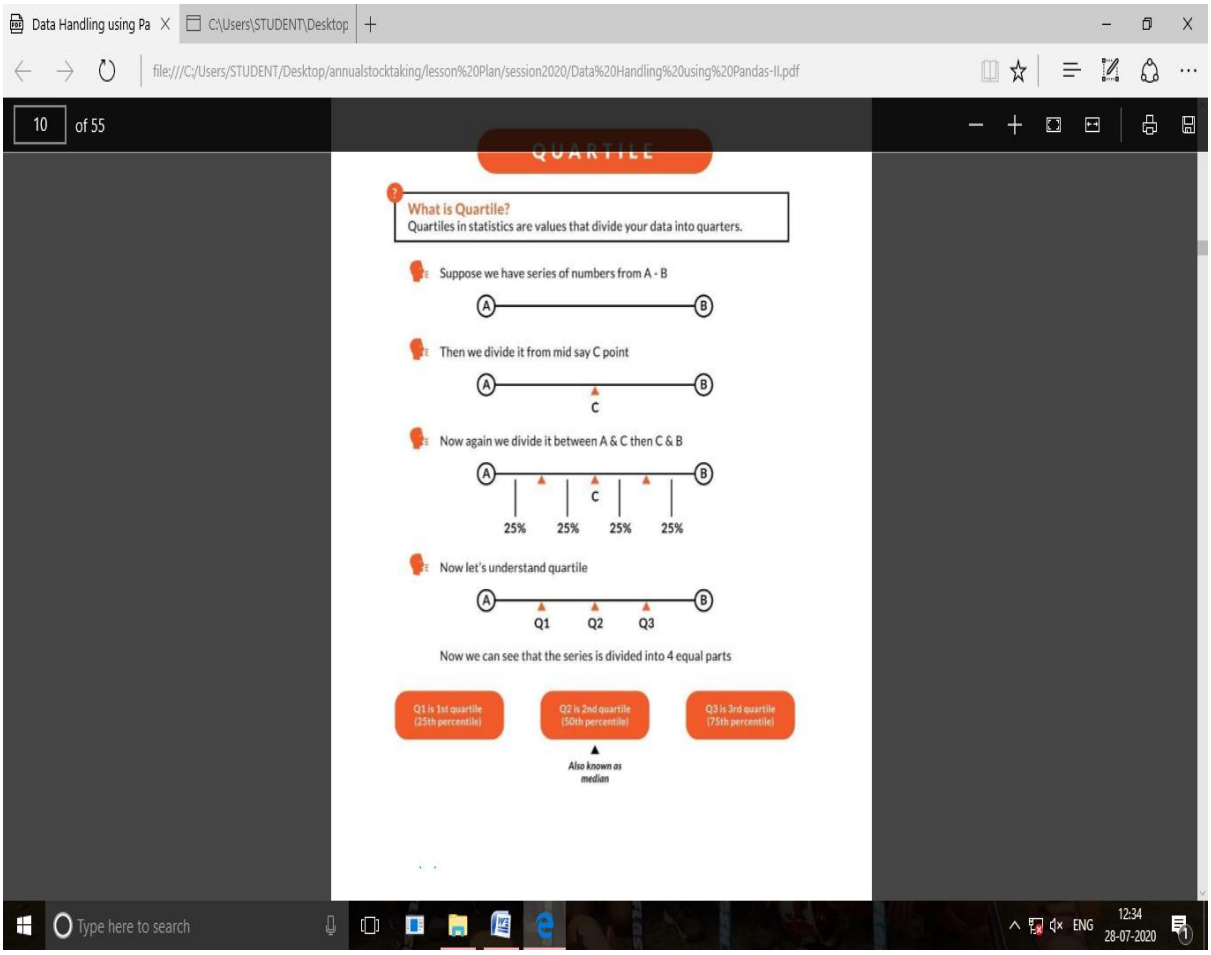
df.mode(axis=1) returns the mode value of every row

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)


In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].mode())
0 2400

print (df.mode(axis=0))
   TCS    WIPRO  L&T
0  2000   2400   2100
```

Descriptive Statistics

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.



Descriptive Statistics

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 every row

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

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)

In [4]: df
Out[4]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100

In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].var())
320000.0

print (df. var(axis=0))
TCS       2.291667e+05
WIPRO     3.200000e+05
L&T       2.541025e+08
Dtype :float64
```

Descriptive Statistics

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 every row

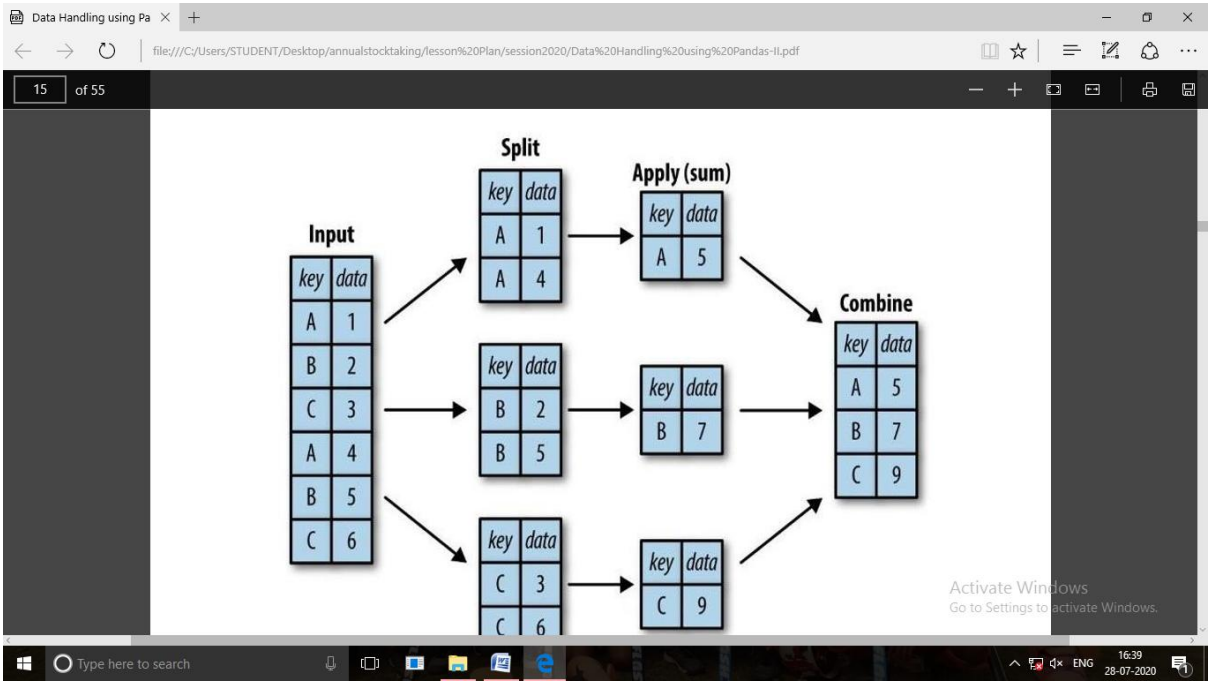
```
In [1]: import pandas as pd

In [2]: 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}}

In [3]: df=pd.DataFrame(Runs)


In [5]: df
Out[5]:
TCS WIPRO L&T
Qtr1 2500 2800 5000
Qtr2 2000 2400 5700
Qtr3 3000 3600 35000
Qtr4 2000 1800 2100
output
In [10]: print(df['WIPRO'].std())
565.685424949238

print (df. std(axis=0))
TCS       478.713554
WIPRO     565.685425
L&T       15940.592837
Dtype :float64
```

Descriptive Statistics

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)

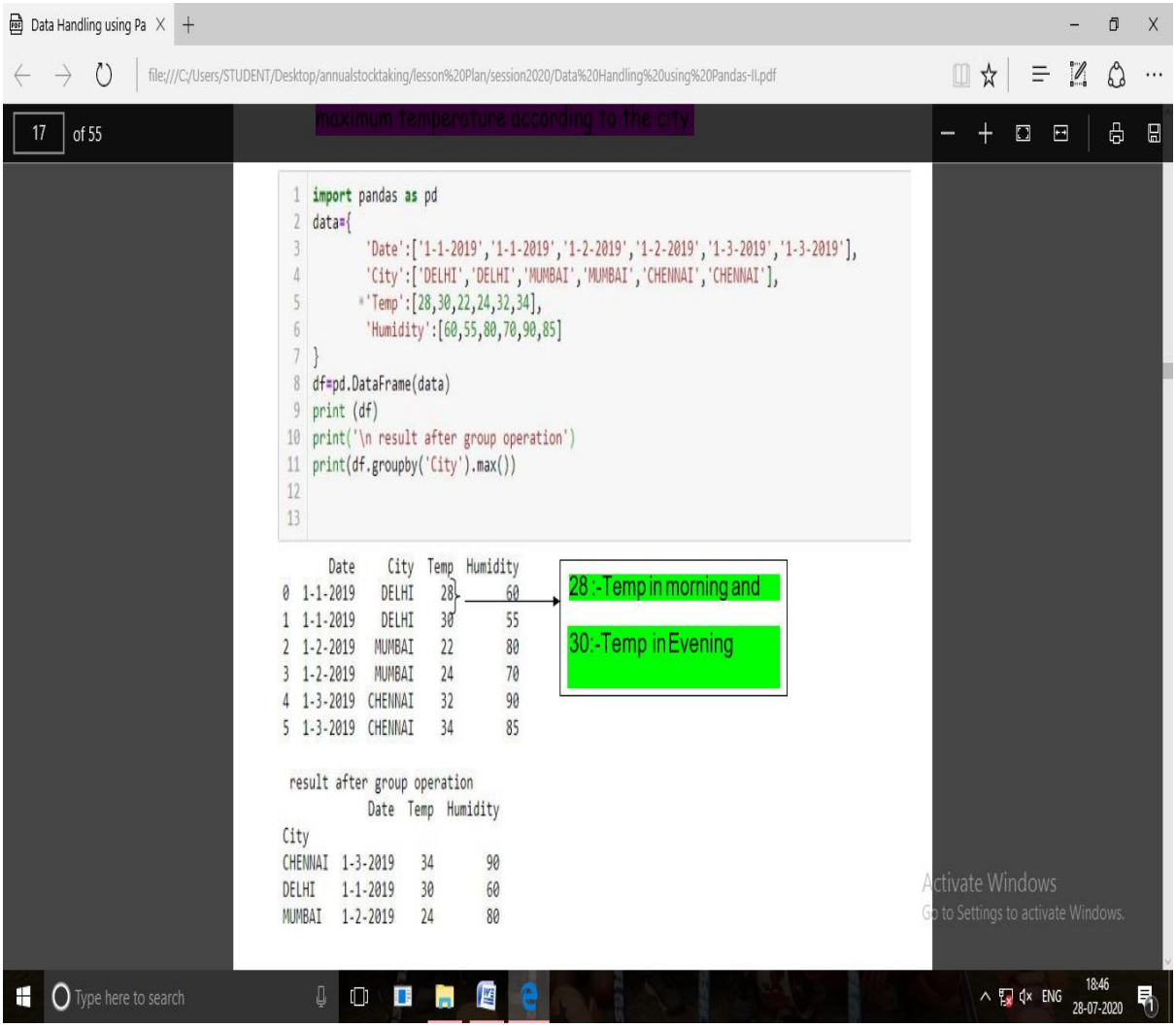
3. Combining the result




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

Descriptive Statistics

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



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.

Sorting (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.

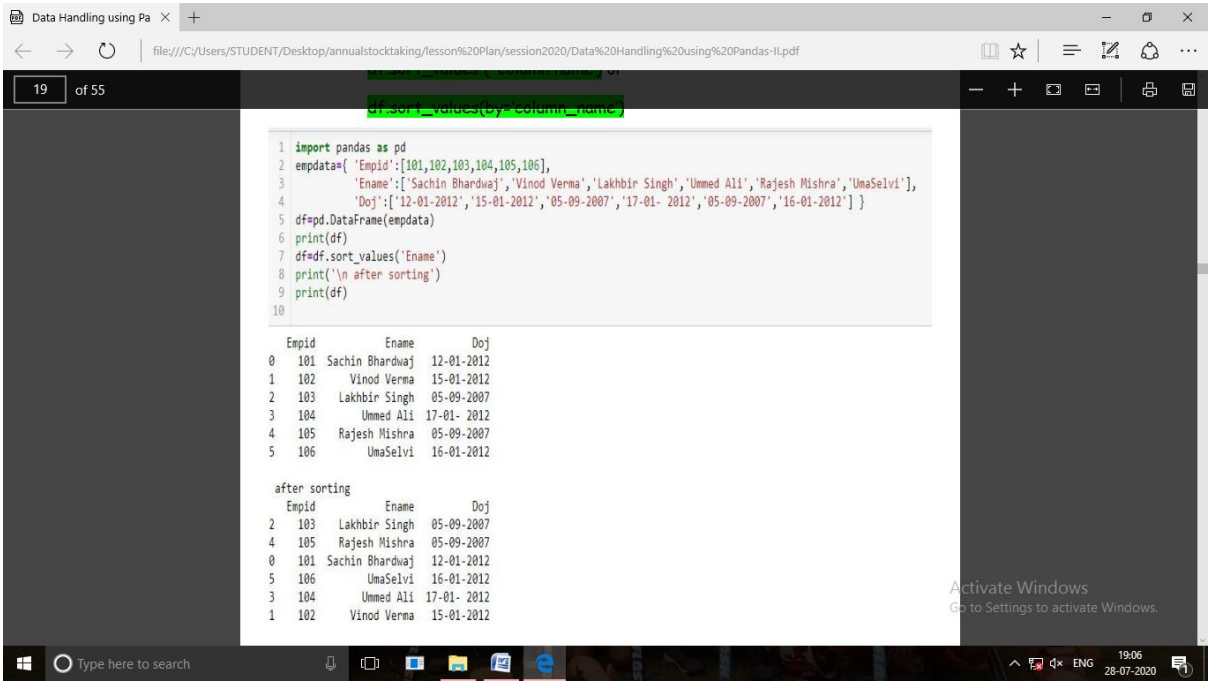
Ascending : By default ascending is true.

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')
```



Sorting (sort_index())

To sort the data based on index Value
Syntax: df.sort_index(by=None, ascending=True or False, inplace =True or False)

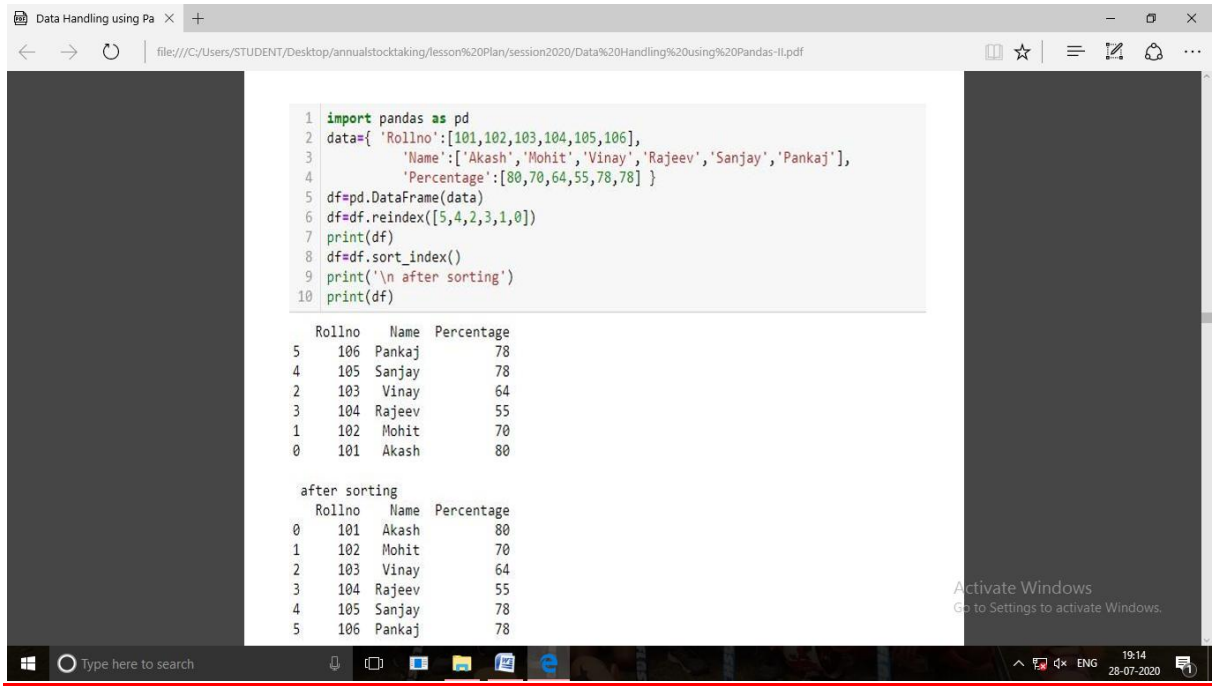
by: Give column name on which you want to perform sorting.

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.

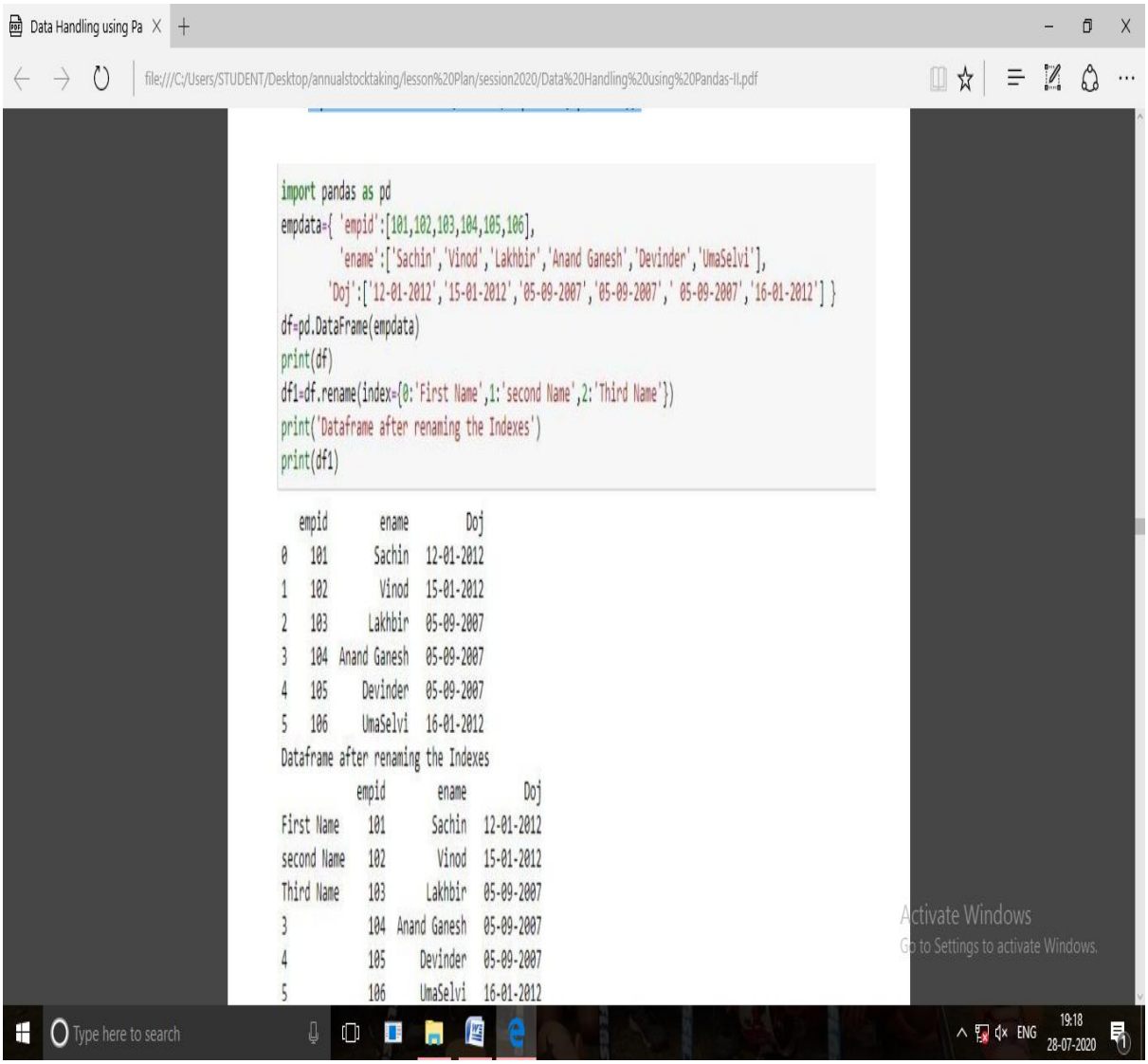
Example 1:- To sort the data frame based on index in ascending order



Renaming index

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

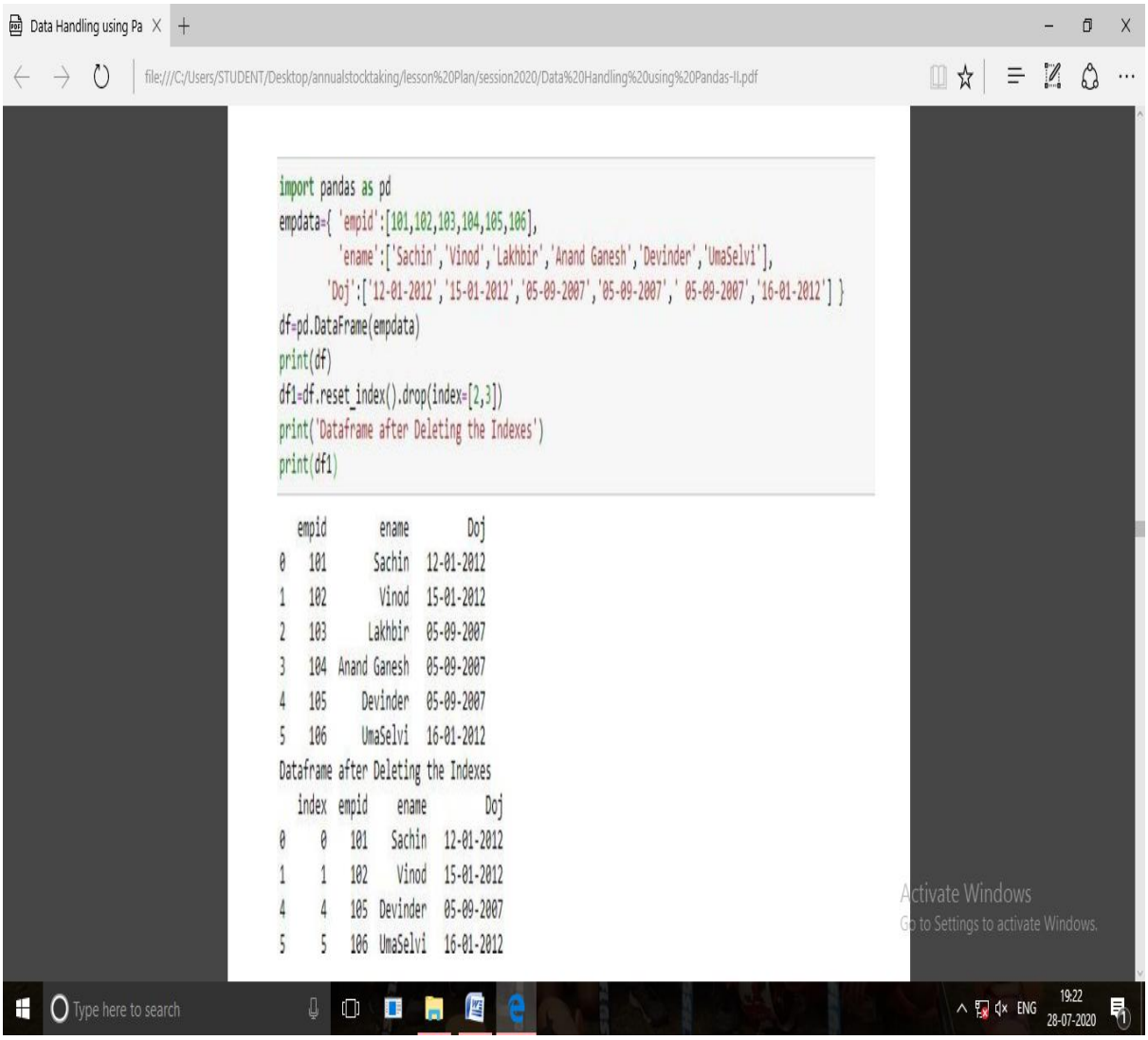
Syntax- df.rename (index, inplace (optional))



Deleting index

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

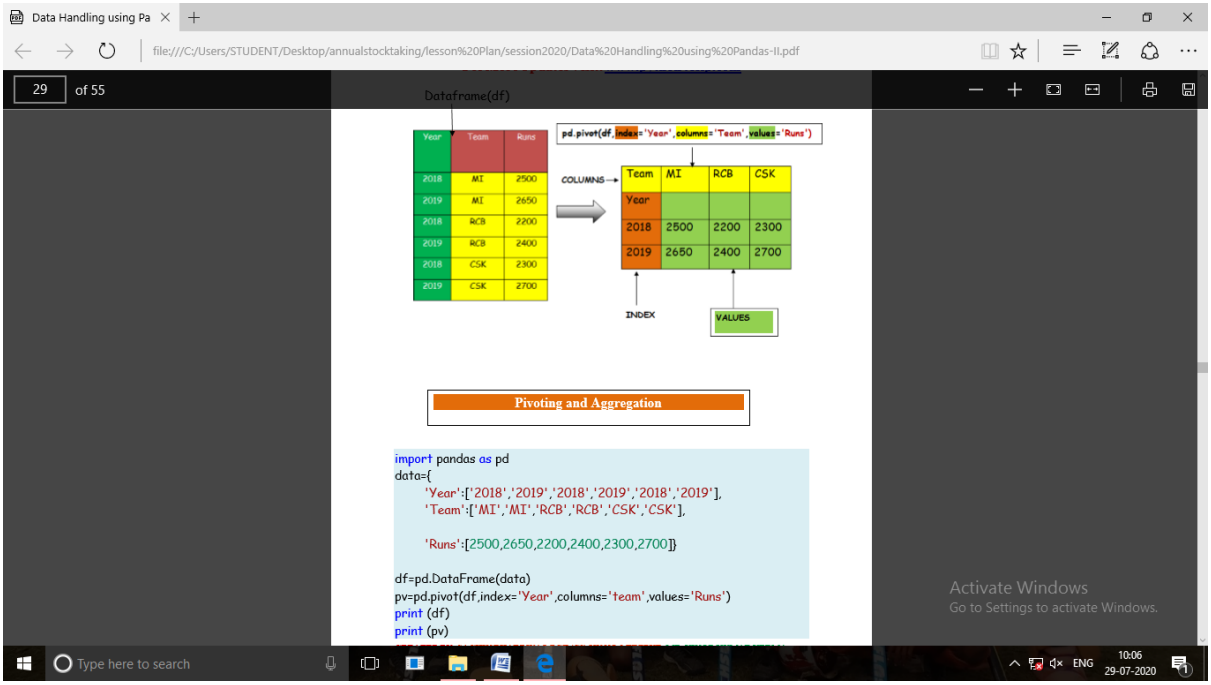
Syntax- df. reset index().drop(index, inplace (optional))

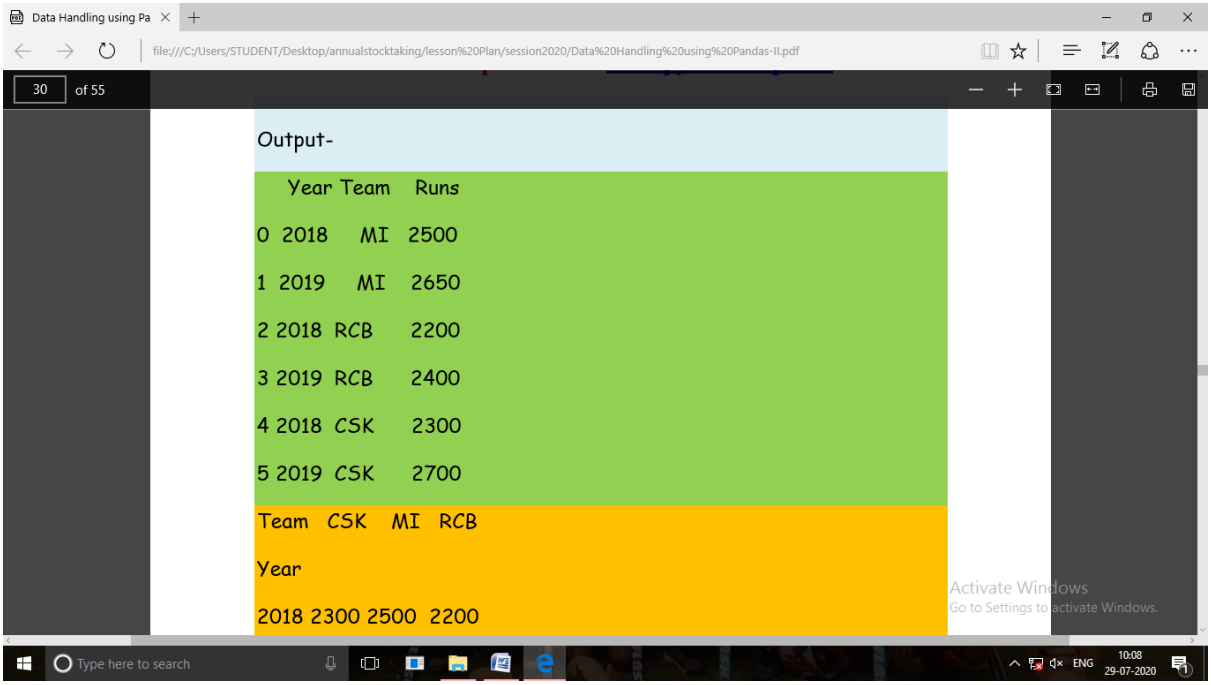


Pivoting

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.

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).

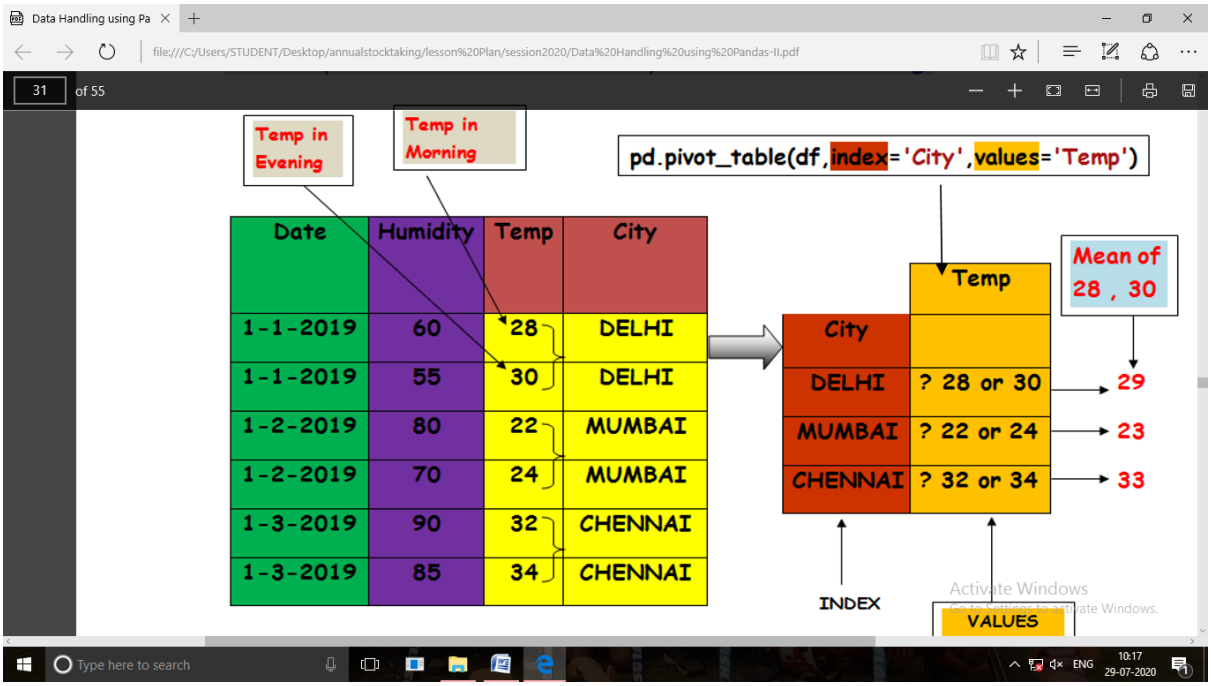

Data Handling using Pa
file:///C:/Users/STUDENT/Desktop/annualstocktaking/lesson%20Plan/session2020/Data%20Handling%20using%20Pandas-II.pdf
29 of 55
Dataframe(df)
pd.pivot(df,index='Year',columns='Team',values='Runs')
COLUMNS
INDEX
VALUES
Pivoting and Aggregation
import pandas as pd
data={
'Year':['2018','2019','2018','2019','2018','2019'],
'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)
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
10:06
29-07-2020

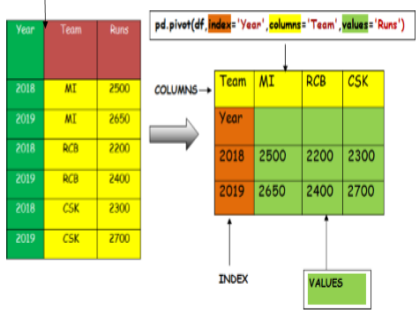

Data Handling using Pa
file:///C:/Users/STUDENT/Desktop/annualstocktaking/lesson%20Plan/session2020/Data%20Handling%20using%20Pandas-II.pdf
30 of 55
Output-
Year Team Runs
0 2018 MI 2500
1 2019 MI 2650
2 2018 RCB 2200
3 2019 RCB 2400
4 2018 CSK 2300
5 2019 CSK 2700
Team CSK MI RCB
Year
2018 2300 2500 2200
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
10:08
29-07-2020

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.





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)
```

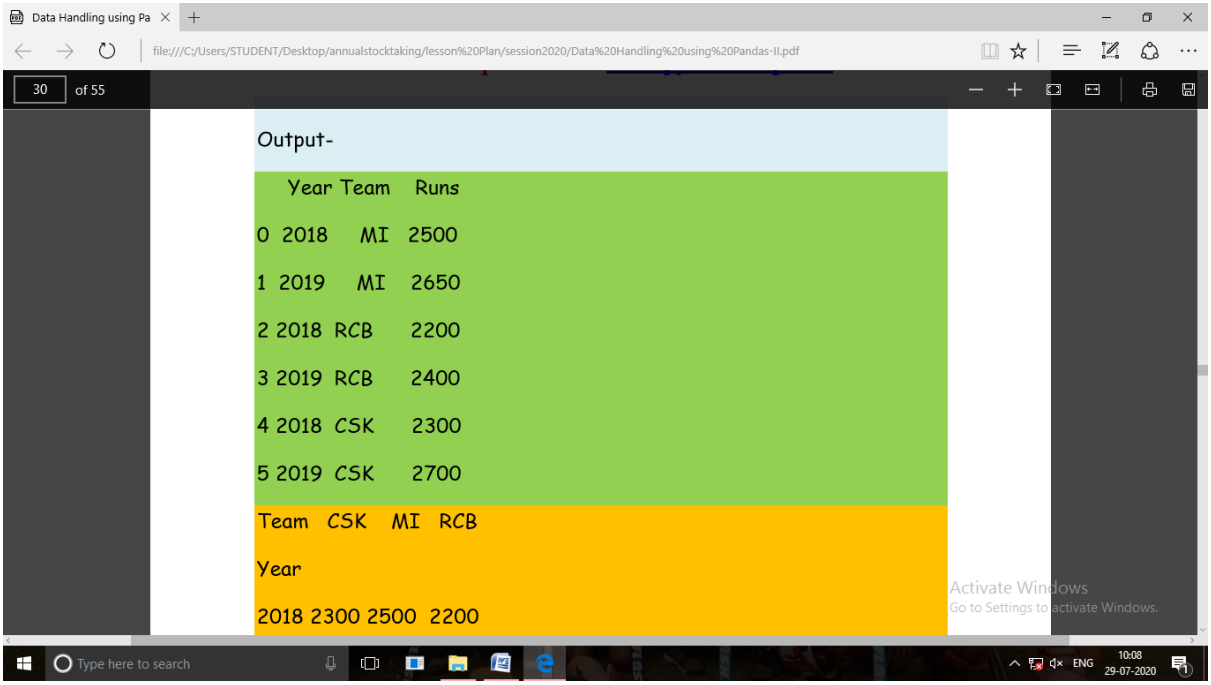


```
    'City':['DELHI','DELHI','MUMBAI','MUMBAI','CHENNAI','CH
    ENNAI'],

    '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-

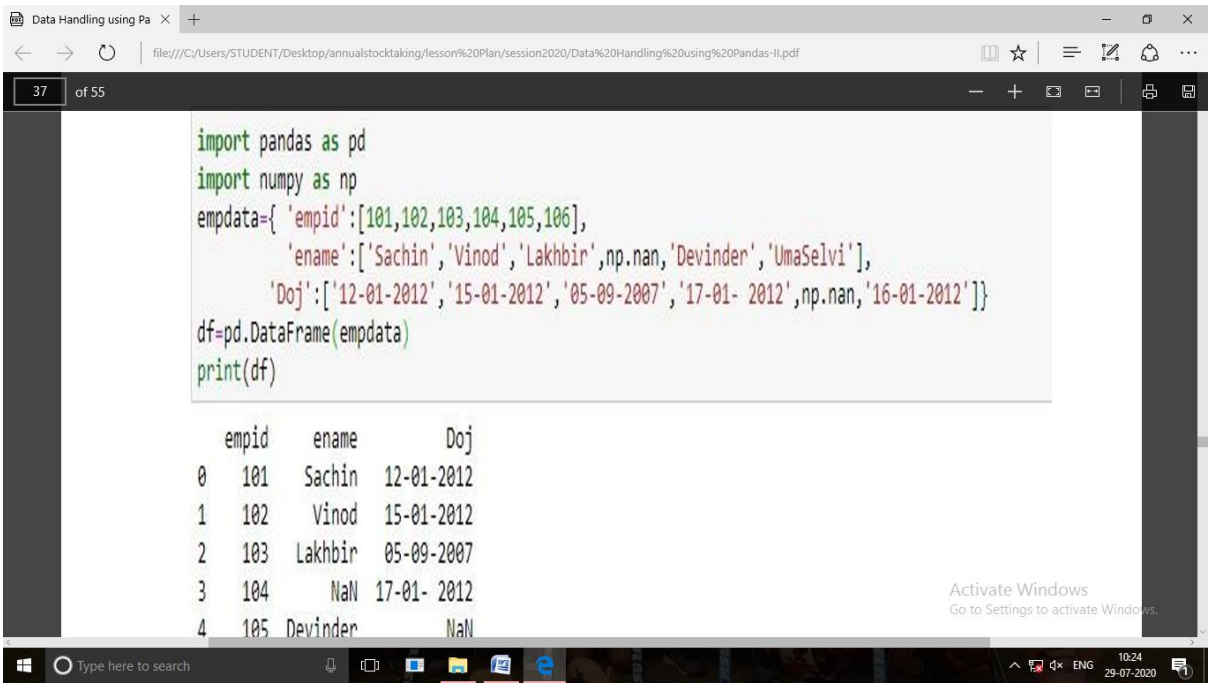
	Date	Humidity	Temp	City
0	1-1-2019	60	28	DELHI
1	1-1-2019	55	30	DELHI
2	1-2-2019	80	22	MUMBAI
3	1-2-2019	70	24	MUMBAI
4	1-3-2019	90	32	CHENNAI
5	1-3-2019	85	34	CHENNAI

City	Temp
CHENNAI	33
DELHI	29
MUMBAI	23



Handling Missing Values- filling & Dropping

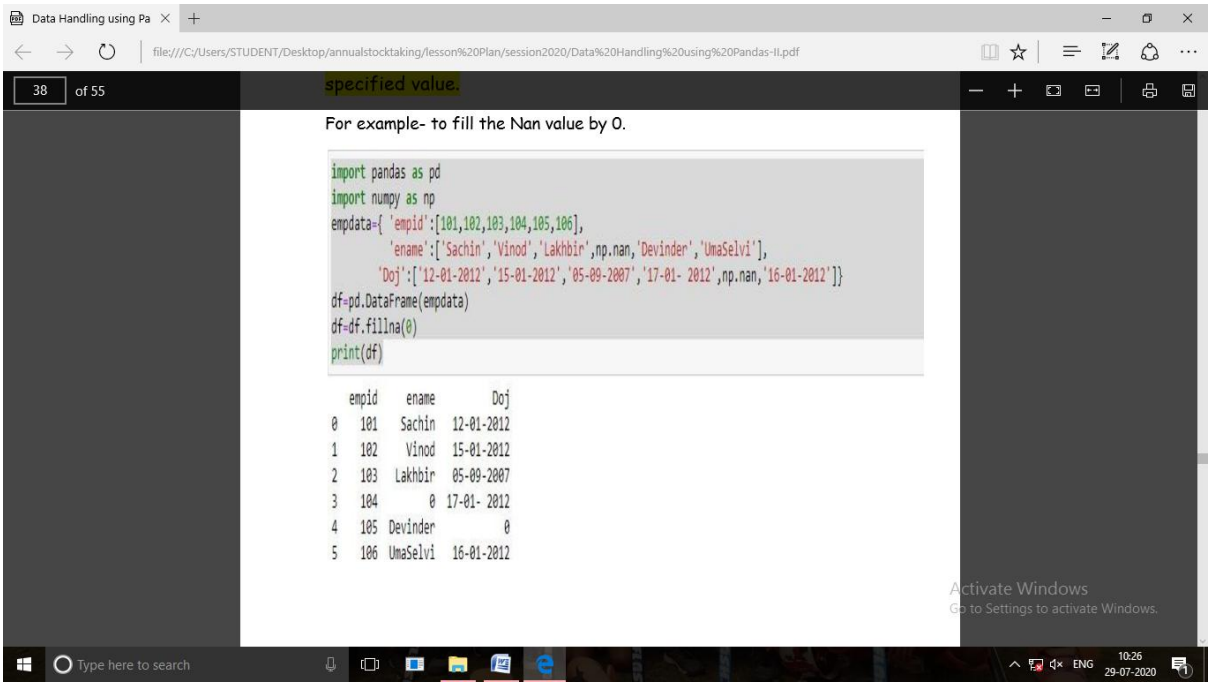
In many cases, the data that we receive from many sources may not be perfect. That means there may be some missing data. For example- in the given program where employee name is missing in one row and date of joining is missing in other row.



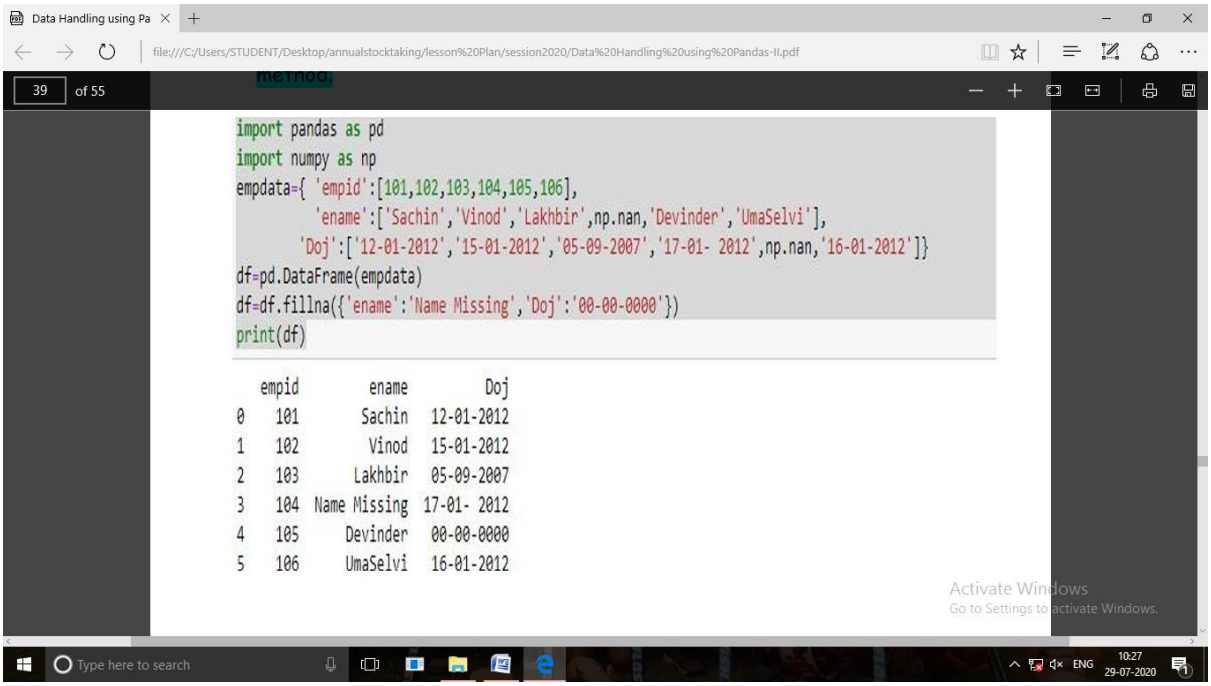
When we convert the data into data frame, the missing data is represented by NaN (Not a Number). NaN is a default marker for the missing value.

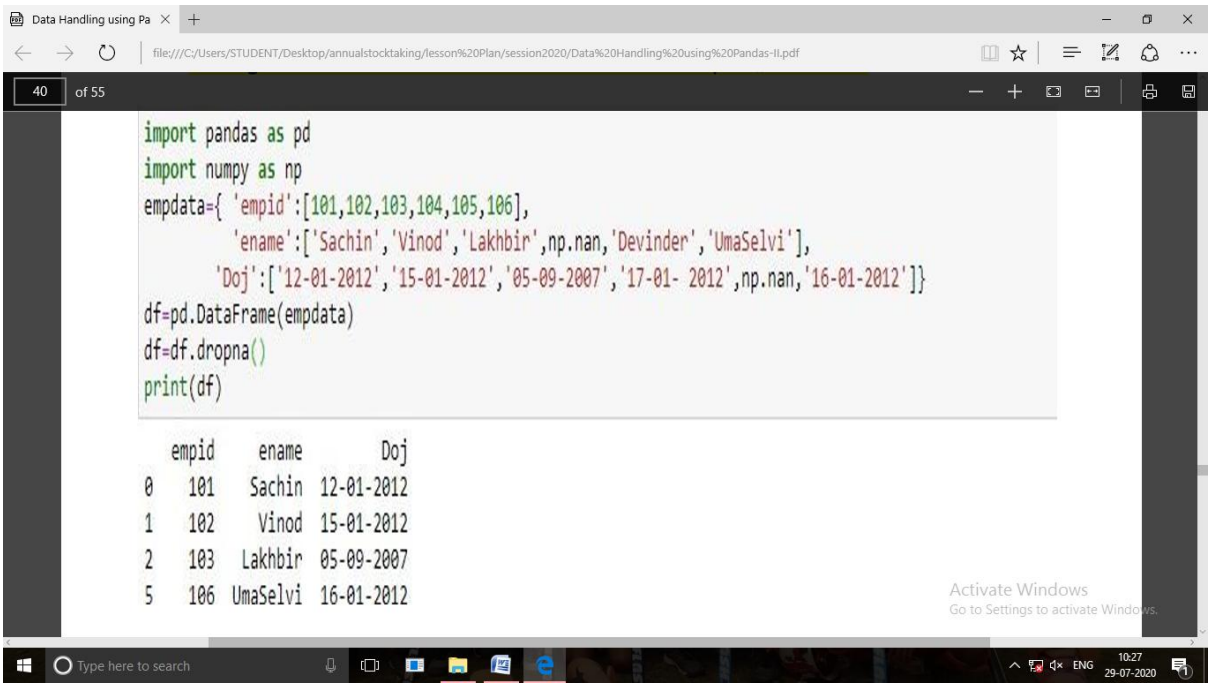
Consider the following Data Frame- We can use fillna() method to replace NaN or Na value by a specified value.

For example- to fill the Nan value by 0.



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.


Data Handling using Pa
file:///C:/Users/STUDENT/Desktop/annualstocktaking/lesson%20Plan/session2020/Data%20Handling%20using%20Pandas-II.pdf
39 of 55
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
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
10:27
29-07-2020


Data Handling using Pa
file:///C:/Users/STUDENT/Desktop/annualstocktaking/lesson%20Plan/session2020/Data%20Handling%20using%20Pandas-II.pdf
40 of 55
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
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
10:27
29-07-2020