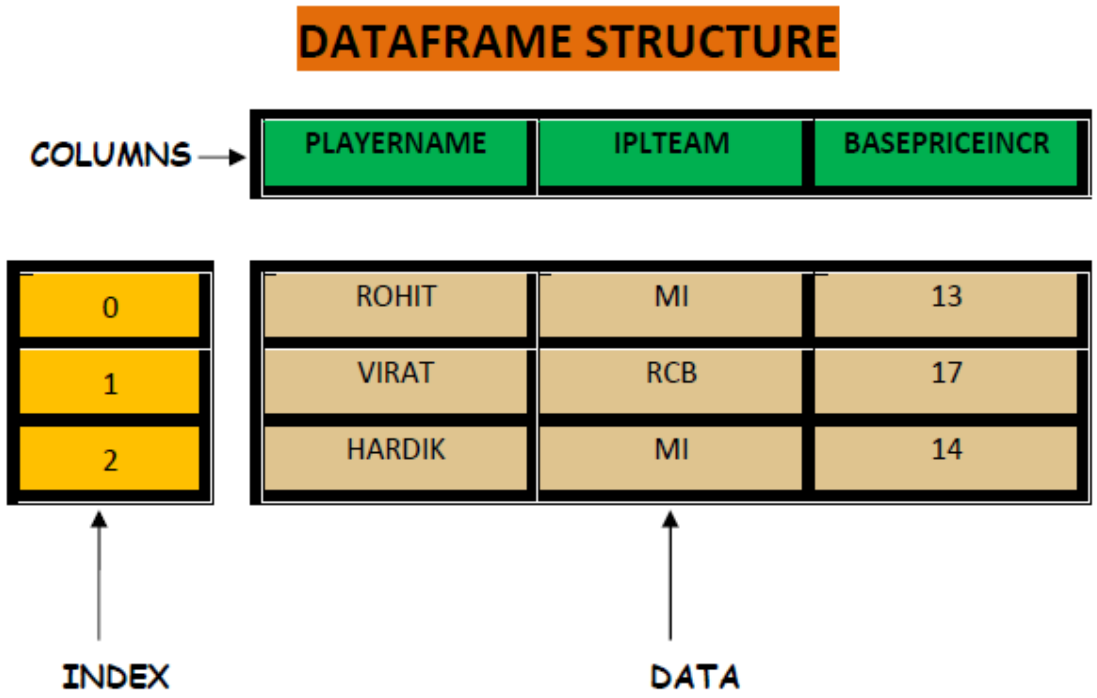
CLASS : XII

SUBJECT: INFORMATICS PRACTICES

UNIT :1 (PYTHON PANDAS- I (Part-3-DATAFRAMES)-) NOTES

DATA HANDLING USING PANDAS DATA FRAMES

INTRODUCTION TO DATAFRAME



A DataFrame is a 2D labeled heterogeneous, data-mutable and size-mutable array which is widely used and is one of the most important data structures. The data in DataFrame is aligned in a tabular fashion in rows and columns therefore has both a row and column labels. Each column can have a different type of value such as numeric, string, boolean, etc.

For example:

ID	NAME	DEPT	SEX	EXPERIENCE
101	JOHN	ENT	M	12
104	SMITH	ORTHOPEDIC	M	5
107	GEORGE	CARDIOLOGY	M	10
109	LARA	SKIN	F	3
113	GEORGE	MEDICINE	F	9
115	JOHNSON	ORTHOPEDIC	M	10

The above table describes data of doctors in the form of rows and columns. Here vertical subset are columns and horizontal subsets are rows. The column labels are Id, Name, Dept, Sex and Experience and row labels are 101, 104, 107, 109, 113 and 115.

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.

Difference between DataFrames and Numpy Array

The basic difference between a Numpy array and a Dataframe is that ***Numpy array contains homogenous data*** while ***Dataframe contains heterogenous data.***

CREATING DATAFRAME

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

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

SYNTAX:
import pandas as pd
pd.DataFrame(data, index, column)

where
data: takes various forms like series, list, constants/scalar values, dictionary, another dataframe

index: specifies index/row labels to be used for resulting frame. They are unique and hashable with same length as data. Default is np.arrange(n) if no index is passed.

column: specifies column labels to be used for resulting frame. They are unique and hashable with same length as data. Default is np.arrange(n) if no index is passed.

Creating An Empty Dataframe

An empty dataframe can be created as follows:

EXAMPLE 1

```
import pandas as pd
dfempty = pd.DataFrame()
print (dfempty)
```

OUTPUT:

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

Creating a DataFrame from List of Dictionaries

A dataframe can be created from a list of dictionaries.

```
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
```

EXAMPLE 2

```
import pandas as pd

l1=[{101:"Amit",102:"Binay",103:"Chahal"}, {102:"Arjun",103:"Fazal"}]
df=pd.DataFrame(l1)
print(df)
```

OUTPUT:

```
       101      102     103
0     Amit     Binay  Chahal
1     NaN     Arjun   Fazal
```

Explanation:

Here, the dictionary **keys are treated as column labels** and **row labels take default values starting from zero.** The values corresponding to each key are treated as rows. The number of rows is equal to the number of dictionaries present in the list. There are two rows in the above dataframe as there are two dictionaries in the list. In the second row the value corresponding to key 101 is NaN because 101 key is missing in the second dictionary.

Note: If the value for a particular key is missing NaN (Not a Number) is inserted at that place.

Creating a DataFrame from List of Lists

A dataframe can be created from a list of lists.

EXAMPLE 3

```
import pandas as pd
a=[[1,"Amit"],[2,"Chetan"],[3,"Rajat"],[4,"Vimal"]]
b=pd.DataFrame(a)
print(b)
```

OUTPUT:

```
     0       1
0    1    Amit
1    2   Chetan
2    3   Rajat
3    4   Vimal
```

Here, ***row and column labels take default values***. Lists form the rows of the dataframe. Column labels and row index can also be changed in the following way:

b=pd.DataFrame(a,index=['I','II','III','IV'],columns=['Rollno','Name'])
Now the output will be:
OUTPUT:

	Rollno	Name
I	1	Amit
II	2	Chetan
III	3	Rajat
IV	4	Vimal

Creating a DataFrame from Dictionary of Lists

A dataframe can be created from a dictionary of lists.

EXAMPLE 4

```
import pandas as pd

d1={"Rollno":[1,2,3], "Total":[350.5,400,420],
"Percentage":[70,80,84]}
df1=pd.DataFrame(d1)
print(df1)
```

OUTPUT

	Rollno	Total	Percentage
0	1	350.5	70
1	2	400.0	80
2	3	420.0	84

Explanation:
Here, the ***dictionary keys are treated as column labels*** and row labels take default values starting from zero.

Creating a DataFrame from Dictionary of Series

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

EXAMPLE 5

```
import pandas as pd

d2={"Rollno":pd.Series([1,2,3,4]), "Total":pd.Series([350.5,400,420]),
"Percentage":pd.Series([70,80,84,80])}

df2=pd.DataFrame(d2)

print(df2)
```

OUTPUT

	Rollno	Total	Percentage
0	1	350.5	70
1	2	400.0	80
2	3	420.0	84
3	4	NaN	80

CUSTOMIZING LABELS/ CHANGING INDEX AND COLUMN LABELS

The index and columns parameters can be used to change the default row and column labels. Consider a dataframe created in example 3 above:

	0	1
0	1	Amit
1	2	Chetan
2	3	Rajat
3	4	Vimal

Suppose we want to have “roll No.” and “Name” as the column labels and numbers from 1 to 4 as the row labels.

EXAMPLE 6:

```
import pandas as pd

a=[[1,"Amit"],[2,"Chetan"],[3,"Rajat"],[4,"Vimal"]]

b=pd.DataFrame(a, index=[1,2,3,4], columns=["Roll No.", "Name"])

print(b)
```

OUTPUT:

	Roll No.	Name
1	1	Amit
2	2	Chetan
3	3	Rajat
4	4	Vimal

EXPLANATION:

Here you can see that using **index** parameter the row labels are changed to 1,2,3 and 4. Similarly using **columns** parameter the column labels are changed to "Roll No." and "Name".

One can easily change the default order of row labels to user defined row labels using the index parameter. It can be used to select only desired rows instead of all rows. Also, the columns parameter in the DataFrame() method can be used to change the sequence of DataFrame columns or to display only selected columns.

EXAMPLE 7

```
import pandas as pd
d2={"Rollno":pd.Series([1,2,3,4],index=[1,2,3,4]),

                    "Total":pd.Series([350.5,400,420],index=[1,2,4]),

                    "Percentage":pd.Series([70,80,84,80],index=[1,2,3,4])
    }

df3=pd.DataFrame(d2)
print(df3)
```

OUTPUT:

	Rollno	Total	Percentage
1	1	350.5	70
2	2	400.0	80
3	3	NaN	84
4	4	420.0	80

EXPLANATION

Here, the dictionary keys are treated as column labels and the row labels are a union of all the series indexes passed to create the DataFrame. Every DataFrame column is a Series object.

CREATING DATAFRAME FROM A DATAFRAME OBJECT

EXAMPLE 8

```
>>>df4=pd.DataFrame(d2, columns=["Rollno","Total"])
>>>df4
```

OUTPUT

```
       Rollno      Total
1           1      350.5
2           2      400.0
3           3      NaN
4           4      420.0
```

EXPLANATION

Here, only two columns Rollno and Total arethere in df4. All rows are displayed.

EXAMPLE 9

```
>>>df5=pd.DataFrame(d2, index=[1,2,3], columns=["Rollno","Percentage"])
>>>df5
```

OUTPUT:

```
       Rollno      Percentage
1           1              70
2           2              80
3           3              84
```

EXPLANATION

Here, **in df5** dataframe **only two columns Rollno and Total are stored** and displayed. Only rows with index values 1,2 and 3 are displayed.

DATAFRAME ATTRIBUTES

The dataframe attribute is defined as any information related to the dataframe object such as size, datatype. etc. Below are some of the attributes about the dataframe object (Consider the ***dataframe df1 defined below*** for all the examples):

	Rollno	Total	Percentage
1	1	350.5	70
2	2	400.0	80
3	3	420.0	84
4	4	356.0	80
5	5	434.0	87
6	6	398.0	79

Attributes

1. df1.size

Return an int representing the number of elements in given dataframe.

```
print(df1.size)
```

OUTPUT

```
18
```

EXPLANATION:

6 rows X 3 columns =18

2. df1.shape

Return a tuple representing the dimensions of the DataFrame.

```
print(df1.shape)
```

OUPUT

```
(6, 3)
```

3. df1.axes

Return a list representing the axes of the DataFrame.

```
print(df1.axes)
```

OUTPUT

```
[Int64Index([1, 2, 3, 4, 5, 6], dtype='int64'), Index(['Rollno', 'Total',
'Percentage'], dtype='object')]
```

4. df1.ndim

Return an int representing the number of axes / array dimensions

```
print(df1.ndim)
```

OUTPUT

```
2
```

5. df1.columns

The column labels of the DataFrame

```
print(df1.columns)
```

OUTPUT

```
Index(['Rollno', 'Total', 'Percentage'], dtype='object')
```

6. df1.values

Return a Numpy representation of the DataFrame.

```
print(df1.values)
```

OUTPUT

```
[[  1.  350.5  70. ]
 [  2.  400.   80. ]
 [  3.  420.   84. ]
 [  4.  356.   80. ]
 [  5.  434.   87. ]
 [  6.  398.   79. ]]
```

7. df1.empty

Indicator whether DataFrame is empty.

```
print(df1.empty)
```

OUTPUT:

```
False
```

ROW/COLUMN OPERATIONS

SELECTING A PARTICULAR COLUMN

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','U
          maSelvi'],
          '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
```

Now if we want to select/display a particular column empid then we will write it as follows:

```
>>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
```

ADDING NEW ROW/COLUMN

Consider **the dataframe df1**(defined above) containing Rollno, Total and Percentage. To add a new column Grade, we write the following statement:

Syntax: DFobject.columnname=new value
Or DFobject[column name]=new value

EXAMPLE 10

```
>>> df1["Grade"] = ["C","B","A"]
>>> df1
```

OUTPUT

```
        Rollno   Total     Percentage   Grade

0            1     350.5            70        C

1            2     400.0            80        B

2            3     420.0             84        A
```

Note: If the column already exists by the same label then the values of that column are updated by the new values.

Using *loc()* to add row or column/access/display row or column

IMPORTANT:

- A column can also be added using loc() method.
- New row can be added to a dataframe using loc() method.

Syntax: DF object.loc[:,column name]=new value
DF object.loc[rowname,:]=new value

EXAMPLE 11

```
>>> df1.loc[:,"Grade"] = ["C","B","A"]
>>> df1
```

OUTPUT

```
   Rollno  Total  Percentage  Grade
0       1  350.5          70      C
1       2  400.0          80      B
2       3  420.0          84      A
```

EXAMPLE 12

```
>>> df1.loc[3]=[4,480.0,96,"A"]
>>> df1
```

OUTPUT

```
   Rollno  Total  Percentage  Grade
0       1  350.5          70      C
1       2  400.0          80      B
2       3  420.0          84      A
3       4  480.0          96      A
```

Note: If the row exists by the same index then the values of that row are updated by the new values.

DELETING ROW/COLUMN

The ***DataFrame.drop()*** method is used to delete row or column from a DataFrame. This method takes names of row/column labels and axis as parameters.

Important : For rows, axis is set to 0 and for columns, axis is set to 1.

Consider dataframe df1, to delete the column Grade the command would be:

EXAMPLE 13

```
>>> df1.drop("Grade", axis=1)
```

OUTPUT

```
   Rollno  Total  Percentage
0       1  350.5          70
1       2  400.0          80
2       3  420.0          84
3       4  480.0          96
```

To delete the row with index 2, the command would be:

EXAMPLE 14

```
>>> df1.drop(2, axis=0)
```

OUTPUT

```
   Rollno  Total  Percentage  Grade
0       1  350.5          70      C
1       2  400.0          80      B
3       4  480.0          96      A
```

Multiple rows can also be deleted. See some examples below:

EXAMPLE 15: To delete rows with index 1 and 2

```
>>> df1.drop([1,2] , axis=0)
```

```
OUTPUT

         Rollno     Total      Percentage Grade

0           1         350.5           70         C

3           4         480.0           96          A
```

RENAMING ROW/COLUMN

The ***DataFrame.rename()*** method is used to rename the labels of rows and columns in a DataFrame.

Syntax: DF.rename(index={names dictionary},columns={names dictionary},inplace=true)
Or DF.rename({names dictionary},axis='index' /'columns')

index : this argument is for row labels, to rename rows only
columns: this argument is for the columns. If you want to rename columns only.
inplace: specify inplace as ***True*** if you want to rename the rows/columns in the same dataframe, if you skip this argument then a new dataframe is created with changed indexes/columns
Consider dataframe df2 given below :

```
df2:
            Rollno    Total   Percentage

Amit          1       350.5          70

Bunty         2       400.0          80

Chetan        3       420.0          84

Reena         4       356.0          80
```

To rename all the columns the command would be:

EXAMPLE 16

```
>>>df2=df2.rename({'Rollno':'Roll', 'Total':'Tot', 'Percentage':'Per'}, axis
= 'columns')
or
 >>>df2=df2.rename(columns={'Rollno':'Roll', 'Total':'Tot',
'Percentage':'Per'})

OUTPUT
          Roll     Tot    Per

Amit        1      350.5   70
Bunty       2      400.0   80
Chetan      3      420.0   84
Reena       4      356.0   80
```

To rename all the rows labels the command would be:

EXAMPLE 17

```
>>> df2=df2.rename({'Amit':'R1', 'Bunty':'R2', 'Chetan':'R3','Reena':'R4'},
axis = 'index')
```

OUTPUT

```
          Rollno  Total     Percentage

Amit         1    350.5          70

Bunty        2    400.0          80

Chetan       3    420.0          84

Reena        4    356.0          80
```

The parameter axis='index' specifies that the row label is to be changed.

Note: If new label is not given corresponding to the old label, the old label is left as it is. Also if extra values are given in the rename() method they are simply ignored.

ACCESSING DATAFRAMES

The data present in the DataFrames can be accessed using **indexing** and **slicing**.

INDEXING

Indexing in DataFrames is of two types:

- Label based Indexing
- Boolean Indexing.

Label based Indexing

The **loc() method** is used for label based indexing. It accepts row/column labels as parameters.

SYNTAX

```
dataframe_name.loc[startrow:endrow,startcolumn:end column]
```

To access a row: just give the row name/label : DF.loc[row label,:]
Make sure not to miss the COLON AFTER COMMA.

EXAMPLE: To display the row with row label 2, the command is:

```
df1.loc[2,:]
```

or

```
df1.loc[2]
```

where the symbol : indicates all columns.

EXAMPLE 18

```
>>> df1.loc[2]
```

OUTPUT

```
Rollno            3
Total             420
Percentage        84
Grade             A
Name: 2, dtype: object
```

Here, you can see that single row label passed, returns that particular row as series. Similarly, single column label passed will return that particular column as series. For example, to display the column with column label Rollno the command is:

To access columns: use:
DF.loc[:,start column:end column]
Make sure not to miss the COLON BEFORE COMMA.

EXAMPLE 19

```
>>> df1.loc[:,"Rollno"]
 OUTPUT
0    1
1    2
2    3
Name: Rollno, dtype: int64
```

Here, : indicates all rows of the specified column label.

Slicing is used to extract a subset of a dataframe. Some of the examples are as follows:

(Consider the dataframe df1 defined below for all the examples):

```
      Rollno    Total         Per
1         1       350.5          70
2         2       400.0          80
3         3       420.0          84
4         4       356.0          80
5         5       434.0          87
6         6       398.0          79
```

SLICING ROWS

```
df1[2:4]
   Rollno  Total          Per
2       2  400.0          80
3       3  420.0          84
df1[:4]
   Rollno  Total         Per
1       1  350.5          70
2       2  400.0          80
3       3  420.0          84
df1[::3]
   Rollno  Total          Per
1       1  350.5          70
4       4  356.0          80
df1[:: -3]
   Rollno  Total          Per
6       6  398.0          79
3       3  420.0          84
df1[3:]
```

```
   Rollno  Total          Per
3       3  420.0          84
4       4  356.0          80
5       5  434.0          87
6       6  398.0          79
```

SLICING COLUMNS

```
df1['Total']
```

or

```
df1.Total
1    350.5
2    400.0
3    420.0
4    356.0
5    434.0
6    398.0
Name: Total, dtype: float64
```

```
df1[['Rollno','Total']]
```
(**Note the use of nested list to specify multiple columns.)**

```
   Rollno  Total
1       1  350.5
2       2  400.0
3       3  420.0
4       4  356.0
5       5  434.0
6       6  398.0
```

loc() and iloc()

We can easily retrieve a slice of rows and columns using **loc() and iloc()** methods.

The **loc[] method is used to retrieve the group of rows and columns by labels** or a boolean array present in the DataFrame. It takes only index labels, and if it exists in the called DataFrame, it returns the rows, columns, or DataFrame.

SYNTAX:

dataframe_name.loc[start_row : end_row, start_column : end_column]

EXAMPLE 22

```
>>> df1.loc[1:2]
```

OUTPUT

```
     Rollno  Total  Percentage
1         2     400.0          80
2         3     420.0          84
```

Here, rows with labels 1 to 2 are displayed.

EXAMPLE 23

```
>>> df1.loc[:,"Rollno":"Total"]
```

OUTPUT

```
      Rollno  Total
0         1   350.5
1         2   400.0
2         3   420.0
```

Here, columns with labels **Rollno to Total** are displayed.

EXAMPLE 24

```
>>>df2.loc['Amit':'Chetan']
```

OUTPUT

```
           Rollno  Total  Percentage
Amit          1     350.5          70
Bunty         2     400.0          80
Chetan        3     420.0          84
```

EXAMPLE 25

```
>>> df2.loc['Amit':'Chetan','Rollno':'Total']
```

OUTPUT

```
                Rollno    Total
      Amit         1      350.5
      Bunty        2      400.0
      Chetan       3      420.0
```

EXAMPLE 26

```
    >>> df2.loc['Amit':'Chetan',['Rollno','Percentage']]
```

OUTPUT

```
               Rollno  Percentage
   Amit         1            70
   Bunty        2            80
   Chetan       3            84
```

Here both end_row and end_column are included in the output.

The .iloc[] method is used retrieve the group of rows and columns at particular positions in the index so it only takes integers. It is used when the index label of the DataFrame is other than numeric series of 0,1,2,....,n, or in the case when the user does not know the index label.

SYNTAX
dataframe_name.iloc[start_row : end_row, start_column : end_column]

Here, end_row and end_column are not included in the output.
Consider the following dataframe df1:

	Rollno	Total	Percentage
Amit	1	350.5	70
Bimal	2	400.0	80
Chetan	3	420.0	84
Harshit	4	356.0	80
Seema	5	434.0	87
Ravi	6	398.0	79

Here, rows from Amit to Chetan and columns from Rollno to Percentage are displayed.
Here, row 1 and columns 1 and 2 are displayed
Command

```
df1.loc['Amit':'Chetan']
```

OUTPUT

	Rollno	Total	Percentage
Amit	1	350.5	70
Bimal	2	400.0	80
Chetan	3	420.0	84

```
df1.loc[:,"Rollno":"Total"]
```

OUTPUT

```
             Rollno   Total
Amit            1      350.5
Bimal           2      400.0
Chetan          3      420.0
Harshit         4      356.0
Seema           5      434.0
Ravi            6      398.0
```

Here, all rows and columns with labels Rollno to Total are displayed.

```
df1.loc['Amit':'Chetan','Rollno':'Percentage']
```

OUTPUT

```
            Rollno  Total  Percentage
Amit           1     350.5          70
Bimal          2     400.0          80
Chetan         3     420.0          84
```

```
df1.loc['Amit':'Chetan',['Rollno','Percentage']]
```

OUTPUT

```
               Rollno  Percentage
Amit            1          70
Bimal           2          80
Chetan          3          84
```

Here, rows from Amit to Chetan and columns Rollno and Percentage are displayed.

```
df1.iloc[1:3]
```

OUTPUT

```
            Rollno  Total  Percentage
Bimal         2    400.0          80
```

```
Chetan        3    420.0          84
```

Here, rows 1 and 2 are displayed. Row 3 is not displayed.

```
df1.iloc[[0,2,4]]
```

OUTPUT

```
            Rollno   Total      Percentage
Amit          1       350.5          70
Chetan        3       420.0          84
Seema         5       434.0          87
```

Here, list of rows i.e. 0,2,4 is displayed.

```
df1.iloc[:,1:3]
```

OUTPUT

```
               Total    Percentage
Amit          350.5         70
Bimal         400.0         80
Chetan        420.0         84
Harshit       356.0         80
Seema         434.0         87
Ravi          398.0         79
```

Here, all rows and 1,2 columns are displayed. Column 3 is not displayed.

```
df1.iloc[1:2,1:3]
```

OUTPUT

```
              Total     Percentage
Bimal         400.0          80
```

NOTE: Column/row labels/index numbers separated by comma and enclosed in a list are used to display specific rows and columns. While Column/row labels/index numbers separated by colon are used to display a range of rows and columns.