MRA DAV PUBLIC SCHOOL

INFORMATICS PRACTICES

Code No-065

CLASS-XII

Blue Print:

Unit No	Unit Name	Marks
1	Data Handling using Pandas and Data Visualization	30
2	Database Query using SQL	25
3	Introduction to Computer Networks	7
4	Societal Impacts	8
	Practical	30
	Total	100

Data Handling Using Pandas - I

By

Rajesh Verma

DATA FRAMES

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.

We can create Data Frame in many ways, such as-

(i) Two dimensional dictionaries

(ii) Two dimensional ndarrays(NumPy arrays)

(iii) Series type object

(iv) Another Dataframe object

(v) Text/CSV files

CREATE DATAFRAME

DataFrames can be created in a number of ways.

SYNTAX

pandas.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 :

```
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.

EXAMPLE

```
import pandas as pd
l1=[{101:"Ansh",102:“Moonal",103:“Devansh"}, {102:“Rashi",103:“Pranjal"}]
df=pd.DataFrame(l1)
print(df)
```

OUTPUT

	101	102	103
0	Ansh	Moonal	Devansh
1	NaN	Rashi	Pranjal

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.

Creating a DataFrame from List of Lists

A dataframe can be created from a list of lists.

EXAMPLE :

```
 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.

Creating a DataFrame from Dictionary of Lists

A dataframe can be created from a dictionary of lists.

EXAMPLE :

```
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

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

```
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

Create Dataframe from Text/CSV Files

Text/CSV files:

We can Create Dataframe from Text/CSV Files by using read_csv() function.

Syntax:

<data frame name>

=pandas.read_csv(filepath_or_buffer, sep=',', delimiter=None, header='infer', names=None, index_col=None, usecols=None, …)

Create Dataframe from Text/CSV Files

Example:

```
import pandas as pd
data = pd.read_csv('iris_data_sample.csv')
df = data.head(3)
print(df)
```

Output:

```
   Unnamed: 0 SepalLengthCm  SepalWidthCm PetalLengthCm  PetalWidthCm  \
0           1           5.1           3.5           1.4           0.2
1           2           4.9           NaN           1.4           0.2
2           3           4.7           3.2           1.3           0.2

       Species
0  Iris-setosa
1          NaN
2  Iris-setosa
```

CUSTOMIZING LABELS

The **index** and **columns** parameters can be used to change the default row and column labels.

Example:

```
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

DATAFRAME ATTRIBUTES

Attributes	Meaning
size	Return an int representing the number of elements in given dataframe. Ex: df4.size
shape	Return a tuple representing the dimensions of the DataFrame. Ex: df4.shape
axes	Return a list representing the axes of the DataFrame. Ex: df4.axes
ndim	Return an int representing the number of axes / array dimensions Ex: df4.ndim
columns	The column labels of the DataFrame Ex: df4.columns
values	Return a Numpy representation of the DataFrame. Ex: df4.values
empty	Indicator whether DataFrame is empty. Ex: df4.empty

Example of Dataframe Attribute

```
import pandas as pd
data = {'RollNo': [1,2,3,4,5,6,7],
        'Name':['Arnab','Amit','Vimal','Aryan','Krish','Kanav','Ayush'],
         'Percentage':[65,85,88,90,98,75,78]}
df4 = pd.DataFrame(data, index = [1,2,3,4,5,6,7])
print("Return an int representing the number of elements in given dataframe:-", df4.size)
print("Return a tuple representing the dimensions of the DataFrame:-", df4.shape)
print("Return a list representing the axes of the DataFrame:-", df4.axes)
print("Return an int representing the number of axes / array dimensions:-", df4.ndim)
print("The column labels of the DataFrame:-", df4.columns)
print("Return a Numpy representation of the DataFrame:-", df4.values)
print("Indicator whether DataFrame is empty:-", df4.empty)
```

Example of DataFrame Attribute

Output

```
Return an int representing the number of elements in given dataframe:- 21
Return a tuple representing the dimensions of the DataFrame:- (7, 3)
Return a list representing the axes of the DataFrame:- [Int64Index([1, 2, 3, 4, 5, 6, 7], dtype='int64'), Index(['RollNo', 'Nam
e', 'Percentage'], dtype='object')]
Return an int representing the number of axes / array dimensions:- 2
The column labels of the DataFrame:- Index(['RollNo', 'Name', 'Percentage'], dtype='object')
Return a Numpy representation of the DataFrame:- [[1 'Arnab' 65]
 [2 'Amit' 85]
 [3 'Vimal' 88]
 [4 'Aryan' 90]
 [5 'Krish' 98]
 [6 'Kanav' 75]
 [7 'Ayush' 78]]
Indicator whether DataFrame is empty:- False
```

Operations on rows and columns in DataFrames

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

1. Adding a new Column to a DataFrame

2. Adding a New Row to DataFrame

3. Deleting Rows or Columns from a DataFrame

Adding a new Column to a DataFrame

Example

```
import pandas as pd
data =
{'Rollno':[1,2,3,4],'Name':['Arnab','Amit','Shikha','Sonali'],
      'Total':[350,400,490,450],'Percentage':[70,80,98,90]}
df5 = pd.DataFrame(data)
print('Old Data Frame:-\n',df5)
#Adding New column in a DataFrame
df5['Grade']=['C','B','A','A']
print('New Data Frame:-\n',df5)
```

Output:

```
Old Data Frame:-
    Rollno    Name  Total  Percentage
0       1   Arnab    350          70
1       2    Amit    400          80
2       3  Shikha    490          98
3       4  Sonali    450          90
New Data Frame:-
    Rollno    Name  Total  Percentage Grade
0       1   Arnab    350          70     C
1       2    Amit    400          80     B
2       3  Shikha    490          98     A
3       4  Sonali    450          90     A
```

Note: If the column already exists by the same label then the values of that column are updated by the new values. New column is added as the last column in the dataframe.

Add new column applying Mathematical Operations

Example:

```
#Adding New Columns applying Mathematical Operation
import pandas as pd
import numpy as np
bill = {'Item Name': ['Pen','Copy','Eraser'],
        'Quantity':[2,4,5],
         'Price': [10,20,5]}
B= pd.DataFrame(bill, index = [1,2,3])
B['Amount'] = B['Quantity'] * B['Price']
print(B)
```

Output:

	Item Name	Quantity	Price	Amount
1	Pen	2	10	20
2	Copy	4	20	80
3	Eraser	5	5	25

New Column add at desired position

Example:

```
#Adding Columns on desired position
import pandas as pd
import numpy as np
bill = {'Item Name': ['Pen','Copy','Eraser'],
        'Quantity':[2,4,5],
         'Price': [10,20,5]}
B= pd.DataFrame(bill, index = [1,2,3])
B.insert(1,"Amount",[20,80,25])
print(B)
```

Output:

	Item Name	Amount	Quantity	Price
1	Pen	20	2	10
2	Copy	80	4	20
3	Eraser	25	5	5

Here, column amount is inserted at the position 1.

To add column using loc method

Example

```
import pandas as pd
import numpy as np
bill = {'Item Name': ['Pen','Copy','Eraser'],
        'Quantity':[2,4,5],
         'Price': [10,20,5]}
B= pd.DataFrame(bill, index = [1,2,3])
B['Amount'] = B['Quantity'] * B['Price']
B.loc [:, "Sale Price"] = [9,18,4]
print(B)
```

Output:

	Item Name	Quantity	Price	Amount	Sale Price
1	Pen	2	10	20	9
2	Copy	4	20	80	18
3	Eraser	5	5	25	4

To add Row using loc method

Example:

```
import pandas as pd
import numpy as np
bill = {'Item Name': ['Pen','Copy','Eraser'],
        'Quantity':[2,4,5],
         'Price': [10,20,5]}
B= pd.DataFrame(bill, index = [1,2,3])
B['Amount'] = B['Quantity'] * B['Price']
B.loc[4] = ['Pencil',9,4,36]
print(B)
```

Output:

	Item Name	Quantity	Price	Amount
1	Pen	2	10	20
2	Copy	4	20	80
3	Eraser	5	5	25
4	Pencil	9	4	36

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

```
#deleting Columns in a Dataframe with label
import pandas as pd
bill =
{'itemName':['Pen','Copy','Eraser','Book','Pencil','Ink',
'Color'],
        'Quantity':[2,10,5,6,4,1,2],
         'Price':[10,20,4,50,6,10,20]}
B = pd.DataFrame(bill, index = [1,2,3,4,5,6,7])
B['Amount'] = B['Quantity'] * B['Price']
B.loc[:,'Sale Price'] = [9,6,5,4,3,2,1]
print(B)
C= B.drop('Sale Price', axis = 1)

print(C)
```

Output:

	itemName	Quantity	Price	Amount	Sale Price
1	Pen	2	10	20	9
2	Copy	10	20	200	6
3	Eraser	5	4	20	5
4	Book	6	50	300	4
5	Pencil	4	6	24	3
6	Ink	1	10	10	2
7	Color	2	20	40	1

	itemName	Quantity	Price	Amount
1	Pen	2	10	20
2	Copy	10	20	200
3	Eraser	5	4	20
4	Book	6	50	300
5	Pencil	4	6	24
6	Ink	1	10	10
7	Color	2	20	40

Deleting Rows

Example:

```
# Deleting row in a Dataframe with label

import pandas as pd
bill =
{'itemName':['Pen','Copy','Eraser','Book','Pencil','Ink',
'Color'],
        'Quantity':[2,10,5,6,4,1,2],
         'Price':[10,20,4,50,6,10,20]}
B = pd.DataFrame(bill, index = [1,2,3,4,5,6,7])
B['Amount'] = B['Quantity'] * B['Price']
B.loc[:,'Sale Price'] = [9,6,5,4,3,2,1]
print(B)
C= B.drop(3, axis = 0)

print(C)
```

Output:

	itemName	Quantity	Price	Amount	Sale Price
1	Pen	2	10	20	9
2	Copy	10	20	200	6
3	Eraser	5	4	20	5
4	Book	6	50	300	4
5	Pencil	4	6	24	3
6	Ink	1	10	10	2
7	Color	2	20	40	1

	itemName	Quantity	Price	Amount	Sale Price
1	Pen	2	10	20	9
2	Copy	10	20	200	6
4	Book	6	50	300	4
5	Pencil	4	6	24	3
6	Ink	1	10	10	2
7	Color	2	20	40	1

Deleting Multiple Rows in DataFrame

Example:

```
# Deleting Multiple rows in a Dataframe

import pandas as pd
bill = {'itemName':['Pen','Copy','Eraser','Book','Pencil','Ink',
'Color'],
        'Quantity':[2,10,5,6,4,1,2],
         'Price':[10,20,4,50,6,10,20]}
B = pd.DataFrame(bill, index = [1,2,3,4,5,6,7])
B['Amount'] = B['Quantity'] * B['Price']
B.loc[:,'Sale Price'] = [9,6,5,4,3,2,1]
print(B)
C= B.drop([1,3], axis = 0)
print(C)
```

Output

```
   itemName  Quantity  Price  Amount  Sale Price
1       Pen         2     10      20           9
2      Copy        10     20     200           6
3    Eraser         5      4      20           5
4      Book         6     50     300           4
5    Pencil         4      6      24           3
6       Ink         1     10      10           2
7     Color         2     20      40           1
   itemName  Quantity  Price  Amount  Sale Price
2      Copy        10     20     200           6
4      Book         6     50     300           4
5    Pencil         4      6      24           3
6       Ink         1     10      10           2
7     Color         2     20      40           1
```

UPDATING DATAFRAME

The **dataframe.at()** method is used to update a single value for a row/column label pair. It raises a key error if 'label' does not exist in DataFrame.

Example:

```
# UPDATING ROWS/COLUMNS USING at method
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)
df2.at[2,'Total']=500
print(df2)
```

Output:

```
   Rollno  Total  Percentage
0       1  350.5          70
1       2  400.0          80
2       3  500.0          84
3       4    NaN          80
```

UPDATING DATAFRAME

To ***update the entire column*** we use the following statement:
Dataframe[column label]= [values] Or Dataframe[:,column label]=[values]

Example:

```
# Updating entire Column

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)
df2['Total']=[450,400,480,500]
print(df2)
```

Output:

	Rollno	Total	Percentage
0	1	400	70
1	2	450	80
2	3	480	84
3	4	500	80

Update rows using loc method

Example:

```
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)
df2.loc[1] = [410,450,480]
print(df2)
```

Output:

	Rollno	Total	Percentage
0	1	350.5	70
1	410	450.0	480
2	3	420.0	84
3	4	NaN	80

RENAMING ROW/COLUMN

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

```
#Renaming Columns in DataFrame
import pandas as pd
bill =
{'itemName':['Pen','Copy','Eraser','Book','Pencil','Ink',
'Color'],
        'Quantity':[2,10,5,6,4,1,2],
         'Price':[10,20,4,50,6,10,20]}
B = pd.DataFrame(bill, index = [1,2,3,4,5,6,7])
print("Old DataFrame:-\n",B)
df3= B.rename({'itemName':'IName', 'Quantity':'Qty',
'Price':'Rs.'}, axis = 'columns')
print("Renamed Column Data:-\n", df3)
```

Output:

```
Old DataFrame:-
   itemName  Quantity  Price
1      Pen         2     10
2     Copy        10     20
3   Eraser         5      4
4     Book         6     50
5   Pencil         4      6
6      Ink         1     10
7    Color         2     20
Renamed Column Data:-
     IName  Qty  Rs.
1     Pen    2   10
2    Copy   10   20
3  Eraser    5    4
4    Book    6   50
5  Pencil    4    6
6     Ink    1   10
7   Color    2   20
```

RENAMING ROW/COLUMN

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

```
#Renaming Rows in DataFrame
import pandas as pd
bill =
{'itemName':['Pen','Copy','Eraser','Book','Pencil','Ink',
'Color'],
        'Quantity':[2,10,5,6,4,1,2],
         'Price':[10,20,4,50,6,10,20]}
B = pd.DataFrame(bill, index = [1,2,3,4,5,6,7])
print("Old DataFrame:-\n",B)
df3= B.rename({1:'R1', 2:'R2', 3:'R3',4:'R4', 5:'R5',
6:'R6',7:'R7'}, axis = 'index')
print("Renamed Row :-\n", df3)
```

Output:

```
Old DataFrame:-
   itemName  Quantity  Price
1      Pen         2     10
2     Copy        10     20
3   Eraser         5      4
4     Book         6     50
5   Pencil         4      6
6      Ink         1     10
7    Color         2     20
Renamed Row :-
    itemName  Quantity  Price
R1      Pen         2     10
R2     Copy        10     20
R3   Eraser         5      4
R4     Book         6     50
R5   Pencil         4      6
R6      Ink         1     10
R7    Color         2     20
```

ACCESSING DATAFRAMES

The data present in the DataFrames can be accessed using indexing and slicing. There are three ways of accessing data:

Method	Explanation	Example
using Dataframe[ ]		
DF["col1"] or DF.col1	To display single column col1	print(df3.itemName)
DF[["col1","col2"]]	To display multiple columns col1 and col2	print(df3[['itemName','Quantity']])

ACCESSING DATAFRAMES

Method	Explanation	Example
Using DataFrame.loc() method		
DF.loc["row1"]	To display single row row1 and all columns	print("Single Row\n",B.loc[1])
DF.loc[["row1" , "row2"]]	To display Specified rows row1 and row2 and all columns	Print(B.loc[[1,2]])
DF.loc["row1":"row4"]	To display range of rows from row1 to row4 and all columns	print(B.loc[1:4])
DF.loc[:,"col1"]	To display single column col1 and all rows	Print(B.loc[:,'Quantity'])
DF.loc[:,["col1","col2"]]	To display multiple columns col1 and col2 and all rows	Print(B.loc[:,['Quantity','Price']])
DF.loc[:,"col1":"col4"]	To display range of columns col1 to col4 and all rows	Print(B.loc[:,'Quantity':'Price'])

ACCESSING DATAFRAMES

Method	Explanation	Example
Using DataFrame.loc() method		
DF.loc[["row1","row2"], ["col1","col2"]]	To display row1 and row2 and col1 and col2	B.loc[[1,4],['itemName','Quantity']]
DF.loc["row1":"row4","col1":"col4"]	To display range of rows from row1 to row4 and range of columns from col1 to col4.	B.loc[1:4,'itemName':'Quantity']
DF.loc["row1":"row4",["col1","col4"]]	To display range of rows from row1 to row4 and multiple columns col1 and col4.	B.loc[1:4,['itemName','Quantity']]
DF.loc[::,::]	To display all rows and all columns	

ACCESSING DATAFRAMES

Method	Explanation	Example
Using DataFrame.iloc() method		
DF.iloc[1]	To display single row at position 1	**B.iloc[1]**
DF.iloc[[1,2]]	To display multiple rows at position 1 and 2	B.iloc[[1,2]]
DF.iloc[1:4]	To display range of rows from row at position 1,2 and 3. Row at position 4 is not displayed.	B.iloc[1:4]
DF.iloc[:,2]	To display single column at position 2	B.iloc[:,2]
DF.iloc[:,[1,2]]	To display multiple columns at position 1 and 2	B.iloc[:,[1,2]]
DF.iloc[:,1:4]	To display range of columns from columns at position 1,2 and 3.	B.iloc[:,1:4]
DF.iloc[[1,2],[2,3]]	To display multiple rows at position 1 and 2 and multiple columns at position 2 and 3.	B.iloc[[0,2],[1,2]]
DF.iloc[1:3,2:4]	To display range of rows from row at position 1 to 2 and range of columns from column at position 2 to 3.	B.iloc[0:2,1:2]

INDEXING

Indexing in DataFrames is of two types: Label based Indexing and 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[row_label, column_label]

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

df1.loc[2,:]

or

df1.loc[2]

where the symbol **:** indicates all columns.

Here, you can see that single row label passed, returns that particular row as series.

To display the column with Column RollNo, the command is:

df1.loc[:,"Rollno"]

Here, you can see that single column label passed, returns that particular column as series.

BOOLEAN INDEXING

Instead of selecting data on the basis of row or column labels we can also select the data based on their values present in the dataframe. **Boolean indexing is a type of indexing which uses actual values of the data in the Series.** Using Boolean indexing we can filter data by applying certain condition on data using relational operators like ==, >, <, <=, >= and logical operators like ~(not), &(and) and |(or).

Example:

```
import pandas as pd
sales = {'Revenue':[25000,22000,25000,30000,28000,35000,40000,27000,24000,26000,30000,35000],
        'Expenses':[12000,11500,15000,18000,16000,12000,14000,18000,17500,13870,15800,20500]}
df = pd.DataFrame(sales,index=['Jan','Feb','Mar','April','May','June','July','Aug','Sep','Oct','Nov','Dec'])
df['Profit'] = df['Revenue']-df['Expenses']
print(df)
print(df['Profit']>20000)
```

SLICING

Slicing is used to extract a subset of a dataframe.

Example:

```
import pandas as pd
salelist = [{'Revenue':250000,'Expenses':30000},
          {'Revenue':500000, 'Expenses':20000},
           {'Revenue':400000, 'Expenses':200000},
           {'Revenue':350000, 'Expenses':200000},
          {'Revenue':450000, 'Expenses':250000},
          {'Revenue':410000, 'Expenses':205000}]
data =
pd.DataFrame(salelist,index=['Jan','Feb','Mar','Apr','May','J
un'])
data['Profit'] = data['Revenue']-data['Expenses']
print(data)
```

print(data[2:4])

```
     Revenue  Expenses  Profit
Mar   400000    200000  200000
Apr   350000    200000  150000
```

print(data[:4])

```
     Revenue  Expenses  Profit
Jan   250000     30000  220000
Feb   500000     20000  480000
Mar   400000    200000  200000
Apr   350000    200000  150000
```

print(data[::4])

```
     Revenue  Expenses  Profit
Jan   250000     30000  220000
May   450000    250000  200000
```

ITERATING OVER DATAFRAMES

There are 3 ways to iterate over Pandas dataframes. These are-

iteritems(): Helps to iterate over each element of the set, column-wise.

iterrows(): Helps to iterate over each element of the set, row-wise.

itertuple(): Helps to iterate over each row and form a tuple out of them.

Iterating with iteritems()

The function iteritems() makes our code snippet runs through each and every element in all the columns of the dataset.

Example:

```
import pandas as pd
saleser = {'Revenue':pd.Series([2000,2500,1800],
index=['Jan','Feb','Mar']),
          'Expenses':pd.Series([600,700,1000],
index=['Jan','Feb','Mar'])}
df2 = pd.DataFrame(saleser)
df2['Profit'] = df2['Revenue']-df2['Expenses']
print(df2)
for key,values in df2.iteritems():
   print("Itemwise:-\n",key,"\n",values)
```

```
     Revenue  Expenses  Profit
Jan     2000       600    1400
Feb     2500       700    1800
Mar     1800      1000     800
Itemwise:-
 Revenue
 Jan    2000
Feb    2500
Mar    1800
Name: Revenue, dtype: int64
Itemwise:-
 Expenses
 Jan     600
Feb     700
Mar    1000
Name: Expenses, dtype: int64
Itemwise:-
 Profit
 Jan    1400
Feb    1800
Mar     800
```

Iterating with iterrows()

With iterrows() we can visit all the elements of a dataset, row-wise. It returns an iterator containing index of each row and the data in each row as a Series.

Example:

```
import pandas as pd
salelist = [{'Revenue':250000,'Expenses':30000},
          {'Revenue':500000, 'Expenses':20000},
           {'Revenue':400000, 'Expenses':200000},
           {'Revenue':350000, 'Expenses':200000},
          {'Revenue':450000, 'Expenses':250000},
          {'Revenue':410000, 'Expenses':205000}]
data = pd.DataFrame(salelist,index=['Jan','Feb','Mar','Apr','May','Jun'])
data['Profit'] = data['Revenue']-data['Expenses']
print(data)
for row_index, row in data.iterrows():
   print("Row-wise data sets\n",row_index, row)
```

```
     Revenue  Expenses  Profit
Jan   250000     30000  220000
Feb   500000     20000  480000
Mar   400000    200000  200000
Apr   350000    200000  150000
May   450000    250000  200000
Jun   410000    205000  205000
Row-wise data sets
 Jan Revenue     250000
Expenses      30000
Profit       220000
Name: Jan, dtype: int64
Row-wise data sets
 Feb Revenue     500000
Expenses      20000
Profit       480000
Name: Feb, dtype: int64
```

Iterating with itertuple()

The function itertuples() creates a tuple for every row in the dataset. Thus, iterating over it would give us a tuple of the rows present.

Example:

```
import pandas as pd
sales =
{'Revenue':[25000,22000,25000,30000,28000,35000,40000,27000,24000,
26000,30000,35000],

'Expenses':[12000,11500,15000,18000,16000,12000,14000,18000,17500,
13870,15800,20500]}
df =
pd.DataFrame(sales,index=['Jan','Feb','Mar','April','May','June','July','Aug'
,'Sep','Oct','Nov','Dec'])
df['Profit'] = df['Revenue']-df['Expenses']
for row in df.itertuples():
    print(row)
```

```
Pandas(Index='Jan', Revenue=25000, Expenses=12000, Profit=13000)
Pandas(Index='Feb', Revenue=22000, Expenses=11500, Profit=10500)
Pandas(Index='Mar', Revenue=25000, Expenses=15000, Profit=10000)
Pandas(Index='April', Revenue=30000, Expenses=18000, Profit=12000)
Pandas(Index='May', Revenue=28000, Expenses=16000, Profit=12000)
Pandas(Index='June', Revenue=35000, Expenses=12000, Profit=23000)
Pandas(Index='July', Revenue=40000, Expenses=14000, Profit=26000)
Pandas(Index='Aug', Revenue=27000, Expenses=18000, Profit=9000)
Pandas(Index='Sep', Revenue=24000, Expenses=17500, Profit=6500)
Pandas(Index='Oct', Revenue=26000, Expenses=13870, Profit=12130)
Pandas(Index='Nov', Revenue=30000, Expenses=15800, Profit=14200)
Pandas(Index='Dec', Revenue=35000, Expenses=20500, Profit=14500)
```

HEAD AND TAIL FUNCTIONS

HEAD FUNCTION: The ***head function*** returns a specified number of rows from the beginning of a dataframe.
SYNTAX : **DataFrame.head(n)**
It gives the first n rows in the DataFrame. The first 5 rows of the DataFrame are returned if the parameter n is not specified.

```
import pandas as pd
sales =
{'Revenue':[25000,22000,25000,30000,28000,35000,40000,27000,24000,26000,30
000,35000],

'Expenses':[12000,11500,15000,18000,16000,12000,14000,18000,17500,13870,15
800,20500]}
df =
pd.DataFrame(sales,index=['Jan','Feb','Mar','April','May','June','July','Aug','Sep','Oct
','Nov','Dec'])
df['Profit'] = df['Revenue']-df['Expenses']
print(df.head())
Print(df.head(10))
```

Output

```
       Revenue  Expenses  Profit
Jan      25000     12000   13000
Feb      22000     11500   10500
Mar      25000     15000   10000
April    30000     18000   12000
May      28000     16000   12000
       Revenue  Expenses  Profit
Jan      25000     12000   13000
Feb      22000     11500   10500
Mar      25000     15000   10000
April    30000     18000   12000
May      28000     16000   12000
June     35000     12000   23000
July     40000     14000   26000
Aug      27000     18000    9000
Sep      24000     17500    6500
Oct      26000     13870   12130
```

HEAD AND TAIL FUNCTIONS

TAIL FUNCTION: The ***tail function*** returns a specified number of rows from the end of a dataframe.

SYNTAX **DataFrame.tail(n)**

gives the last n rows in the DataFrame. If the parameter n is not specified it gives the last 5 rows of the DataFrame.

```
import pandas as pd
sales =
{'Revenue':[25000,22000,25000,30000,28000,35000,40000,27000,24000,26000,30
000,35000],

'Expenses':[12000,11500,15000,18000,16000,12000,14000,18000,17500,13870,15
800,20500]}
df =
pd.DataFrame(sales,index=['Jan','Feb','Mar','April','May','June','July','Aug','Sep','Oct
','Nov','Dec'])
df['Profit'] = df['Revenue']-df['Expenses']
print(df..tail())
Print(df.tail10))
```

Output

	Revenue	Expenses	Profit
Aug	27000	18000	9000
Sep	24000	17500	6500
Oct	26000	13870	12130
Nov	30000	15800	14200
Dec	35000	20500	14500

	Revenue	Expenses	Profit
Mar	25000	15000	10000
April	30000	18000	12000
May	28000	16000	12000
June	35000	12000	23000
July	40000	14000	26000
Aug	27000	18000	9000
Sep	24000	17500	6500
Oct	26000	13870	12130
Nov	30000	15800	14200
Dec	35000	20500	14500

JOINING, MERGING AND CONCATENATION

In real life scenarios data comes from multiple source and files. To analyze this data, we may need to bring all the data in one place by some sort of join logic and then start our analysis. We also sometimes need to merge multiple csv files together in a single DataFrame. Joining, merging and concatenating DataFrames are the core processes to start with data analysis and machine learning tasks.

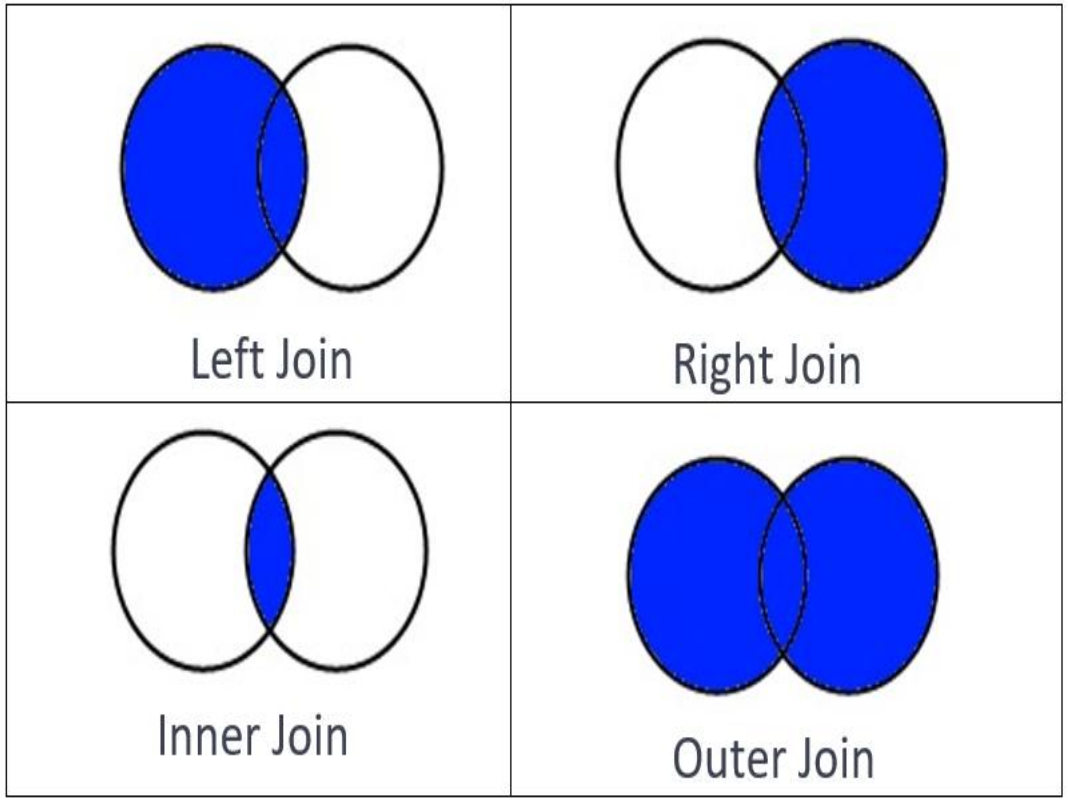
Python provides various methods to combine multiple DataFrames. Here we are discussing join(), merge() and concat() methods to combine DataFrames.

Two DataFrames which hold different kinds of information about the same entity based on the index or some common column(key) can be easily joined using merge() and join() functions of Pandas.

JOINING, MERING AND CONCATENATION

There are basically four types of join:

Types	Description
left	Use keys from left object
right	Use keys from right object
inner	Use intersection of keys
Outer	Use union of keys



JOINING DATAFRAMES

The join() function is used to join two DataFrames.

SYNTAX

DataFrame.join(other, how='left', sort=False)

where:

other : name of the other dataframe

how : specifies type of merging, default is ‘left’. it can be inner, outer, left, right

sort : if it is true the data in the resultant dataframe is sorted

Using join function with left method

```
# Join Dataframe using left method
import pandas as pd
std1 = {'Name':pd.Series(['Arun','Manjit','Shiv','Sunil'], index =
[1,2,3,4]),
        'Address':pd.Series(['Solan','Shimla','Chandigarh','New
Delhi'],index = [1,2,3,4])}
dfstd = pd.DataFrame(std1)
print(dfstd)
std2 = {'English': pd.Series([65,70,74,75], index = [1,3,4,5]),
       'Maths': pd.Series([60,60,65,67], index = [1,3,4,5]),
       'IP': pd.Series([65,60,70,67], index = [1,3,4,5])}
dfmarks = pd.DataFrame(std2)
print(dfmarks)
d3=dfstd.join(dfmarks, how="left")
print(d3)
```

```
     Name     Address
1    Arun       Solan
2  Manjit      Shimla
3    Shiv  Chandigarh
4   Sunil   New Delhi
   English  Maths  IP
1       65     60  65
3       70     60  60
4       74     65  70
5       75     67  67
     Name     Address  English  Maths    IP
1    Arun       Solan     65.0   60.0  65.0
2  Manjit      Shimla      NaN    NaN   NaN
3    Shiv  Chandigarh     70.0   60.0  60.0
4   Sunil   New Delhi     74.0   65.0  70.0
```

Here, data of the DataFrames dfstd and dfmarks is joined based on indexes of the dataframe dfstd. As index 2 is not present in dfmarks thus resultant dataframe has NaN as Eng,Maths,IP for this index number. Index number 5 of dfmarks is ignored as there is no corresponding index number in dfstd.

Using join function with right method

```
# Joinf Dataframe using right method
import pandas as pd
std1 = {'Name':pd.Series(['Arun','Manjit','Shiv','Sunil'],
index = [1,2,3,4]),

'Address':pd.Series(['Solan','Shimla','Chandigarh','New
Delhi'],index = [1,2,3,4])}
dfstd = pd.DataFrame(std1)
print(dfstd)
std2 = {'English': pd.Series([65,70,74,75], index =
[1,3,4,5]),
      'Maths': pd.Series([60,60,65,67], index = [1,3,4,5]),
      'IP': pd.Series([65,60,70,67], index = [1,3,4,5])}
dfmarks = pd.DataFrame(std2)
print(dfmarks)
d3=dfstd.join(dfmarks, how="right")
print(d3)
```

```
     Name     Address
1    Arun       Solan
2  Manjit      Shimla
3    Shiv  Chandigarh
4   Sunil   New Delhi
   English  Maths  IP
1       65     60  65
3       70     60  60
4       74     65  70
5       75     67  67
    Name     Address  English  Maths  IP
1   Arun       Solan       65     60  65
3   Shiv  Chandigarh       70     60  60
4  Sunil   New Delhi       74     65  70
5    NaN         NaN       75     67  67
```

Here, data of the DataFrames dfstd is joined based on indexes of the dataframe dfmarks. As index 5 is not present in dfstd thus resultant dataframe has NaN as Name and address for this index number. Index number 2 of dfstd is ignored as there is no corresponding index number in dfmarks.

Using join function with inner method

```
# Joinf Dataframe using inner method
import pandas as pd
std1 = {'Name':pd.Series(['Arun','Manjit','Shiv','Sunil'],
index = [1,2,3,4]),
       'Address':pd.Series(['Solan','Shimla','Chandigarh','New
Delhi'],index = [1,2,3,4])}
dfstd = pd.DataFrame(std1)
print(dfstd)
std2 = {'English': pd.Series([65,70,74,75], index = [1,3,4,5]),
      'Maths': pd.Series([60,60,65,67], index = [1,3,4,5]),
      'IP': pd.Series([65,60,70,67], index = [1,3,4,5])}
dfmarks = pd.DataFrame(std2)
print(dfmarks)
d3=dfstd.join(dfmarks, how="inner")
print(d3)
```

```
     Name     Address
1    Arun       Solan
2  Manjit      Shimla
3    Shiv  Chandigarh
4   Sunil   New Delhi
   English  Maths  IP
1       65     60  65
3       70     60  60
4       74     65  70
5       75     67  67
    Name     Address  English  Maths  IP
1   Arun       Solan       65     60  65
3   Shiv  Chandigarh       70     60  60
4  Sunil   New Delhi       74     65  70
```

Here, the data corresponding to index numbers which are present in both the DataFrames is visible in the resultant dataframe.

Using join function with outer method

```
# Joinf Dataframe using inner method
import pandas as pd
std1 = {'Name':pd.Series(['Arun','Manjit','Shiv','Sunil'],
index = [1,2,3,4]),
       'Address':pd.Series(['Solan','Shimla','Chandigarh','New
Delhi'],index = [1,2,3,4])}
dfstd = pd.DataFrame(std1)
print(dfstd)
std2 = {'English': pd.Series([65,70,74,75], index = [1,3,4,5]),
      'Maths': pd.Series([60,60,65,67], index = [1,3,4,5]),
      'IP': pd.Series([65,60,70,67], index = [1,3,4,5])}
dfmarks = pd.DataFrame(std2)
print(dfmarks)
d3=dfstd.join(dfmarks, how="outer")
print(d3)
```

```
     Name     Address
1    Arun       Solan
2  Manjit      Shimla
3    Shiv  Chandigarh
4   Sunil   New Delhi
   English  Maths  IP
1       65     60  65
3       70     60  60
4       74     65  70
5       75     67  67
     Name     Address  English  Maths    IP
1    Arun       Solan     65.0   60.0  65.0
2  Manjit      Shimla      NaN    NaN   NaN
3    Shiv  Chandigarh     70.0   60.0  60.0
4   Sunil   New Delhi     74.0   65.0  70.0
5     NaN         NaN     75.0   67.0  67.0
```

Here, the resultant dataframe is the union of the two given DataFrames. Data corresponding to all index numbers is present in the resultant dataframe and the missing values are filled with NaN.

Note: Mainly join function is used exclusively to join the DataFrames based on the index, not on the attribute names, so keep the attributes names different in two DataFrames to be joined.

MERGING DATAFRAMES

merge(<data_frame1>,<data_frame2>, [on=<field_name>], [how='left' |'right' |'inner' |'outer']))

Where

Data_frame1, data_frame2 : data frames to be merged

on : the common column in both the data frames based on which merging is to be performed.

how : specifies type of merging, default is 'inner'

Merge Data Frame

Example:

```
import pandas as pd
df1 =
pd.DataFrame({'key':['b','b','a','c','a','a','b'],'data1':ra
nge(7)})
df2 = pd.DataFrame({'key':['a','b','d'], 'data2' :
range(3)})
x = pd.merge(df1,df2)
print(x)
```

Output

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

Merge Two DataFrames on a Key

Example:

```
import pandas as pd
df1 =
pd.DataFrame({'key':['b','b','a','c','a','a','b'],'data1':range(
7)})
df2 = pd.DataFrame({'key':['a','b','d'], 'data2' : range(3)})
x = pd.merge(df1,df2)
print(x)
x1  = pd.merge(df1,df2, on = 'key')
print(x1)
```

Output

```
   key  data1  data2
0    b      0      1
1    b      1      1
2    b      6      1
3    a      2      0
4    a      4      0
5    a      5      0
   key  data1  data2
0    b      0      1
1    b      1      1
2    b      6      1
3    a      2      0
4    a      4      0
5    a      5      0
```

You may notice that the ‘c’ and ‘d’ values and associated data are missing from the result, By default merge does an ‘inner’ join, the keys in the result are the result are the intersection ot the common set found in both tables.

Merge Two DataFrames on a Key

Example:

```
import pandas as pd
left = pd.DataFrame({
  'id':[1,2,3,4,5],
  'Name': ['Ajay', 'Amit', 'Allen', 'Arick', 'Ayush'],

'subject_id':['Eng','Maths','Physics','Chemistry','IP']
})
right = pd.DataFrame({'id':[1,2,3,4,5],
  'Name': ['Vinay', 'Virat', 'Vivan', 'Viraj', 'Viran'],

'subject_id':['English','Maths','Economics','IP','CS']}
)
print(pd.merge(left,right,on='id'))
```

Output

	id	Name_x	subject_id_x	Name_y	subject_id_y
0	1	Ajay	Eng	Vinay	English
1	2	Amit	Maths	Virat	Maths
2	3	Allen	Physics	Vivan	Economics
3	4	Arick	Chemistry	Viraj	IP
4	5	Ayush	IP	Viran	CS

Merge Two DataFrames on Multiple Keys

Example:

```
import pandas as pd
left = pd.DataFrame({
   'id':[1,2,3,4,5],
  'Name': ['Ajay', 'Amit', 'Allen', 'Arick', 'Ayush'],
   'subject_id':['Eng','Maths','Physics','Chemistry','IP']})
right = pd.DataFrame({'id':[1,2,3,4,5],
    'Name': ['Vinay', 'Virat', 'Vivan', 'Viraj', 'Ayush'],
   'subject_id':['Eng','Maths','Economics','CS','IP']})
print(pd.merge(left,right, on = ['id', 'subject_id']))
```

Output

```
   id Name_x subject_id Name_y
0   1   Ajay        Eng  Vinay
1   2   Amit      Maths  Virat
2   5  Ayush         IP  Ayush
```

Merge Using 'how' Argument with left method)

The **how** argument to merge specifies how to determine which keys are to be included in the resulting table. If a key combination does not appear in either the left or the right tables, the values in the joined table will be NA.

```
import pandas as pd
left = pd.DataFrame({
  'id':[1,2,3,4,5],
 'Name': ['Ajay', 'Amit', 'Allen', 'Arick', 'Ayush'],
  'subject_id':['Eng','Maths','Physics','Chemistry','IP']})
right = pd.DataFrame({'id':[1,2,3,4,5],
   'Name': ['Vinay', 'Virat', 'Vivan', 'Viraj', 'Ayush'],
  'subject_id':['Eng','Maths','Economics','CS','IP']})
print(pd.merge(left,right, on = 'id', how = 'left'))
print(pd.merge(left,right, on = 'subject_id', how = 'left'))
```



```
   id Name_x subject_id_x Name_y subject_id_y
0   1   Ajay          Eng  Vinay          Eng
1   2   Amit        Maths  Virat        Maths
2   3  Allen      Physics  Vivan    Economics
3   4  Arick    Chemistry  Viraj           CS
4   5  Ayush           IP  Ayush           IP
   id_x Name_x subject_id  id_y Name_y
0     1   Ajay        Eng   1.0  Vinay
1     2   Amit      Maths   2.0  Virat
2     3  Allen    Physics   NaN    NaN
3     4  Arick  Chemistry   NaN    NaN
4     5  Ayush         IP   5.0  Ayush
```

The *left join* return joins over the left entries. The output rows now correspond to the entries in the left input. It includes all the rows of your data frame present on left side and only those from the dataframe present on right side that match.

Merge Using 'how' Argument with Right method)

RIGHT JOIN

The *right join* return joins over the right entries. The output rows now correspond to the entries in the right input. It includes all the rows of your data frame present on right side and only those from the dataframe present on left side that match.

```
import pandas as pd
left = pd.DataFrame({
  'id':[1,2,3,4,5],
 'Name': ['Ajay', 'Amit', 'Allen', 'Arick', 'Ayush'],
  'subject_id':['Eng','Maths','Physics','Chemistry','IP']})
right = pd.DataFrame({'id':[1,2,3,4,5],
   'Name': ['Vinay', 'Virat', 'Vivan', 'Viraj', 'Ayush'],
  'subject_id':['Eng','Maths','Economics','CS','IP']})
print(pd.merge(left,right, on = 'id', how = ‘right’))
print(pd.merge(left,right, on = 'subject_id', how = ‘right'))
```

```
   id Name_x subject_id_x Name_y subject_id_y
0   1   Ajay          Eng  Vinay          Eng
1   2   Amit        Maths  Virat        Maths
2   3  Allen      Physics  Vivan    Economics
3   4  Arick    Chemistry  Viraj           CS
4   5  Ayush           IP  Ayush           IP
   id_x Name_x subject_id  id_y Name_y
0   1.0   Ajay        Eng     1  Vinay
1   2.0   Amit      Maths     2  Virat
2   NaN    NaN  Economics     3  Vivan
3   NaN    NaN         CS     4  Viraj
4   5.0  Ayush         IP     5  Ayush
```

Merge Using 'how' Argument with outer method

OUTER JOIN

Outer join can be performed by specifying the how argument as “outer” in the merge function. An *outer join* returns a join over the union of the input columns, and fills in all missing values with NANs. The joined DataFrame will contain all records from both the DataFrames and fill in NaNs for missing matches on either side.

```
import pandas as pd
left = pd.DataFrame({
  'id':[1,2,3,4,5],
 'Name': ['Ajay', 'Amit', 'Allen', 'Arick', 'Ayush'],
  'subject_id':['Eng','Maths','Physics','Chemistry','IP']})
right = pd.DataFrame({'id':[1,2,3,4,5],
   'Name': ['Vinay', 'Virat', 'Vivan', 'Viraj', 'Ayush'],
  'subject_id':['Eng','Maths','Economics','CS','IP']})
print(pd.merge(left,right, on = 'subject_id', how = 'outer'))
```

	id_x	Name_x	subject_id	id_y	Name_y
0	1.0	Ajay	Eng	1.0	Vinay
1	2.0	Amit	Maths	2.0	Virat
2	3.0	Allen	Physics	NaN	NaN
3	4.0	Arick	Chemistry	NaN	NaN
4	5.0	Ayush	IP	5.0	Ayush
5	NaN	NaN	Economics	3.0	Vivan
6	NaN	NaN	CS	4.0	Viraj

Merge Using 'how' Argument with inner method

```
import pandas as pd
left = pd.DataFrame({
  'id':[1,2,3,4,5],
 'Name': ['Ajay', 'Amit', 'Allen', 'Arick', 'Ayush'],
  'subject_id':['Eng','Maths','Physics','Chemistry','IP']})
right = pd.DataFrame({'id':[1,2,3,4,5],
   'Name': ['Vinay', 'Virat', 'Vivan', 'Viraj', 'Ayush'],
  'subject_id':['Eng','Maths','Economics','CS','IP']})
print(pd.merge(left,right, on = 'subject_id', how = 'inner'))
```

INNER JOIN

The INNER JOIN produces only the set of records that match in both DataFrames which are to be merged. Inner join is the default join but we can also pass ***inner*** to the parameter how to perform inner join. To keep only rows that match from the data frames

	id_x	Name_x	subject_id	id_y	Name_y
0	1	Ajay	Eng	1	Vinay
1	2	Amit	Maths	2	Virat
2	5	Ayush	IP	5	Ayush

COCATENATING DATAFRAMES

To concatenate DataFrames, Python provides concat() function. It is used to concatenate DataFrames along the row or column axis.

SYNTAX

pd.concat(data_frame1, data_frame2, [ignore_index=True],[axis=1])

where

data_frame1, data_frame2 : are the DataFrames to be concatenated

ignore_index = True : used to generate row index numbers from 0 to n-1

axis=1 : used to concatenate along the columns. If this is not added, concatenation is done along the rows. For concatenating along the rows, axis=0. It is not mandatory to mention axis=0 as by default the concatenation is done along the rows.

Concatenating The DataFrames Along Rows

```
import pandas as pd
classXIA= {'Rollno':[1,2,3,4],
        'Name':['Ajay','Amit','Allen','Arick'],
        'Marksobt':[450,435,465,350]}
classXIB = {'Rollno':[1,3,4,5],
         'Name':['Virat','Viraj','Vivan','Vinod'],
         'Marksobt':[420,380,400,480]}
cdf = pd.DataFrame(classXIA,index = [1,2,3,4])
cdf1 = pd.DataFrame(classXIB, index = [1,2,3,4])
print(cdf)
print(cdf1)
con = pd.concat([cdf,cdf1], axis = 0,ignore_index = True)
print(con)
```

```
   Rollno   Name  Marksobt
1       1   Ajay       450
2       2   Amit       435
3       3  Allen       465
4       4  Arick       350
   Rollno   Name  Marksobt
1       1  Virat       420
2       3  Viraj       380
3       4  Vivan       400
4       5  Vinod       480
   Rollno   Name  Marksobt
1       1   Ajay       450
2       2   Amit       435
3       3  Allen       465
4       4  Arick       350
1       1  Virat       420
2       3  Viraj       380
3       4  Vivan       400
4       5  Vinod       480
```

Concatenating The DataFrames Along Columns

```
import pandas as pd
classXIA= {'Rollno':[1,2,3,4],
        'Name':['Ajay','Amit','Allen','Arick'],
        'Marksobt':[450,435,465,350]}
classXIB = {'Rollno':[1,3,4,5],
         'Name':['Virat','Viraj','Vivan','Vinod'],
         'Marksobt':[420,380,400,480]}
cdf = pd.DataFrame(classXIA,index = [1,2,3,4])
cdf1 = pd.DataFrame(classXIB, index = [1,2,3,4])
print(cdf)
print(cdf1)
con = pd.concat([cdf,cdf1], axis =1, ignore_index = False)
print(con)
```

```
   Rollno   Name  Marksobt
1       1   Ajay       450
2       2   Amit       435
3       3  Allen       465
4       4  Arick       350
   Rollno   Name  Marksobt
1       1  Virat       420
2       3  Viraj       380
3       4  Vivan       400
4       5  Vinod       480
   Rollno   Name  Marksobt  Rollno   Name  Marksobt
1       1   Ajay       450       1  Virat       420
2       2   Amit       435       3  Viraj       380
3       3  Allen       465       4  Vivan       400
4       4  Arick       350       5  Vinod       480
```

Concatenating Using append

```
import pandas as pd
left = {
  'id':[1,2,3,4,5],
 'Name': ['Ajay', 'Amit', 'Allen', 'Arick', 'Ayush'],
  'subject_id':['Eng','Maths','Physics','Chemistry','IP']}
right = pd.DataFrame({'id':[1,2,3,4,5],
   'Name': ['Vinay', 'Virat', 'Vivan', 'Viraj', 'Ayush'],
  'subject_id':['Eng','Maths','Economics','CS','IP']},index =
pd.Index(['1','2','3','4','5']) )
l1 = pd.DataFrame(left,index = pd.Index(['1','2','3','4','5']))
l1 = l1.reindex()
df1= l1.append(right,sort = True, ignore_index = True)
print(df1)
```

Append() : This method is used to merge two DataFrames.
Syntax : DataFrame.append(dataframename, sort = 'True/False)

	Name	id	subject_id
1	Ajay	1	Eng
2	Amit	2	Maths
3	Allen	3	Physics
4	Arick	4	Chemistry
5	Ayush	5	IP
1	Vinay	1	Eng
2	Virat	2	Maths
3	Vivan	3	Economics
4	Viraj	4	CS
5	Ayush	5	IP

To get the column labels appear in sorted order we can set the parameter sort = True

Method read_csv()

- Reads data from a comma separated values (csv) file.
- By default takes first row as column headings.
- It is possible to provide column names explicitly using names parameter.
- Row labels can be taken from a column using index_col parameter.

```
import pandas as pd
filename = r'C:\XII2021\sales.csv'
sales = pd.read_csv(filename, header=None)
print(sales)
```

```
      0     1   2
0   JAN   B100  30
1   JAN   B200  40
…
```

```
sales = pd.read_csv(filename, header=None, names=['Month','Book','Units'])
```

```
  Month  Book  Units
0   JAN  B100     30
1   JAN  B200     40
```

Important parameters of read_csv()

sep	Delimiter to use. If sep is None, it automatically detects the separator.
header	Row number(s) to use as the column names. By default column names are inferred from the first line of the file. Set it to None to provide column names using names parameter.
names	List of column names to use.
index_col	Column to use as row labels of the DataFrame.
nrows	Number of rows of the file to read.
skip_blank_lines	Skips over blank lines rather than interpreting as NaN values.
usecols	Specifies which columns to use.

Methods related to reading data into DataFrame

- read_excel('path_to_file.xls', sheet_name='Sheet1')
- read_html(url)
- read_sql_table(table_name, con)
- read_sql_query(query,con)

Writing DataFrame

- to_csv(file)
- to_json(file)
- to_excel()
- to_clipboard()
- to_sql()
- to_html()
- to_pickle()

```
sales.to_json("sales.json")                          # Writes column-wise

sales.to_json("sales.json", orient="records")        # Writes row-wise

sales.to_excel('sales.xlsx', sheet_name='Sales')

sales.to_sql('sales',con)

sales.to_pickle('sales.pickle')
```