Python Pandas- II
Dataframes and Other Operations

Based on CBSE Curriculum

Class -11

Chapter-11

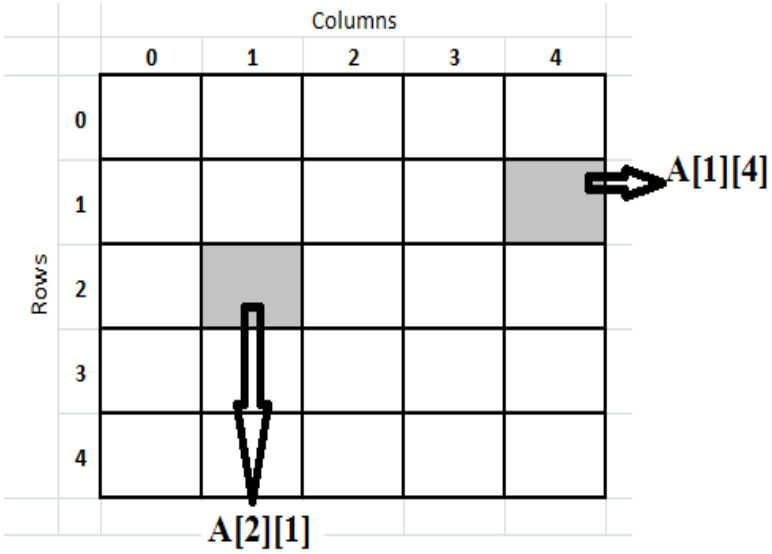
By-
Neha Tyagi
PGT CS
KV 5 Jaipur II Shift
Jaipur Region

Introduction

- In last chapter, we got some information about python pandas ,data structure and series. It is not able to handle the data in the form of 2D or multidimensional related to real time.
- For such tasks, python pandas provides some other data structure like dataframes and panels etc.
- Dataframe objects of Pandas can store 2 D hetrogenous data.
- On the other hand, panels objects of Pandas can store 3 D hetrogenous data.
- In this chapter, we will discuss them.

DataFrame Data Structure

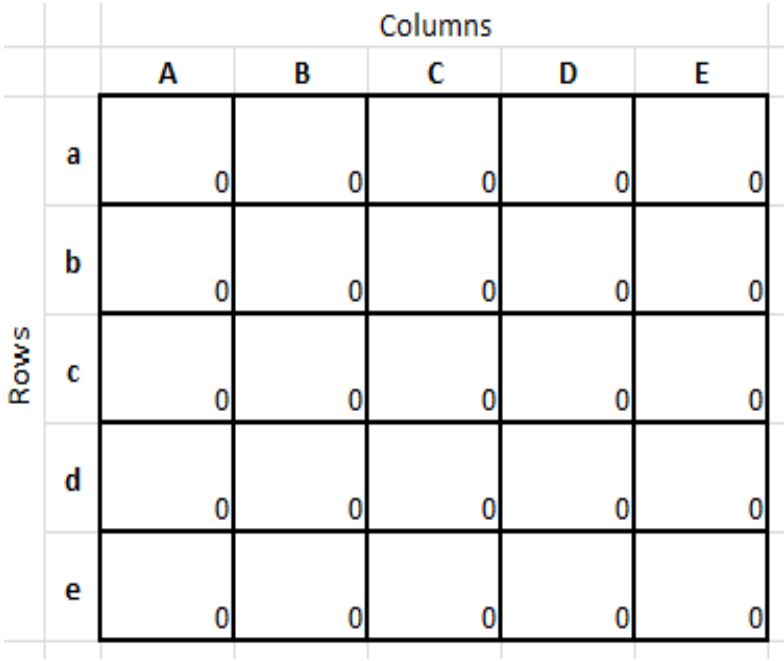
- A DataFrame is a kind of panda structure which stores data in 2D form.
- Actually, it is 2 dimensional labeled array which is an ordered collection of columns where columns can store different kinds of data.
- A 2D array is a collection of row and column where each row and column shows a definite index starts from 0.
- In the given diagram, there are 5 rows and 5 columns. Row and column index are from 0 to 4 respectively.
  Each cell has the address like-
  A[2][1], A[1][4] etc like shown in the diagram.



Characteristics of DataFrame

Characteristics of a DataFrame are as follows-

- It has 2 index or 2 axes.
- It is somewhat like a spreadsheet where row index is called index and column index is called column name.
- Indexes can be prepared by numbers, strings or letters.
- It is possible to have any kind of data in columns.
- its values are mutable and can be changed anytime.
- Size of DataFrame is also mutable i.e. The number of row and column can be increaded or decreased anytime.



Creation and presentation of DataFrame

- DataFrame object can be created by passing a data in 2D format.

  import pandas as pd

  <dataFrameObject> = pd.DataFrame(<a 2D Data Structure>,\ [columns=<column sequence>],[index=<index sequence>])

- You can create a DataFrame by various methods by passing data values. Like-
- 2D dictionaries
  - 2D ndarrays
  - Series type object
  - Another DataFrame object

Creation of DataFrame from 2D Dictionary

A. Creation of DataFrame from dictionary of List or ndarrays.

```
>>> import pandas as pd
>>> dict={'Students':['Pratibha','Ritika','Saumya','Aryan','Keshwam','Priyanka']
,'Marks':[69,65,64,59,59,40], 'Sports':['TQ','TT','KB','VB','CR','KO']}
>>> dtf=pd.DataFrame(dict)
>>> dtf
   Students  Marks Sports
0  Pratibha     69     TQ
1    Ritika     65     TT
2    Saumya     64     KB
3     Aryan     59     VB
4   Keshwam     59     CR
5  Priyanka     40     KO
```

In the above example, index are automatically generated from 0 to 5 and column name are same as keys in dictionary.

Indexes are automatically generated by using np.range(n)

column name are generated from keys of 2D Dictionary

```
>>> import pandas as pd
>>> dict={'Students':['Pratibha','Ritika','Saumya','Aryan','Keshwam','Priyanka'
,'Marks':[69,65,64,59,59,40], 'Sports':['TQ','TT','KB','VB','CR','KO']}
>>> dtf=pd.DataFrame(dict,index=['I','II','III','IV','V','VI'])
>>> dtf
     Students  Marks Sports
I    Pratibha     69     TQ
II     Ritika     65     TT
III    Saumya     64     KB
IV      Aryan     59     VB
V     Keshwam     59     CR
VI   Priyanka     40     KO
```

Here, indexes are specified by you.

Meaning, if you specify the sequence of index then index wlil be the set specified by you only otherwise it will be automatically generated from 0 to n-1.

Creation of DataFrame from 2D Dictionary

B. Creation of DataFrame from dictionary of Dictionaries-

```
>>> yr2015={'Qtr1':40000, 'Qtr2':35000,'Qtr3': 47000,'Qtr4':45000}
>>> yr2016={'Qtr1':42000, 'Qtr2':37000,'Qtr3': 49000,'Qtr4':47000}
>>> yr2017={'Qtr1':43000, 'Qtr2':38000,'Qtr3': 50000,'Qtr4':48000}
>>> kvfee={2015:yr2015,2016:yr2016,2017:yr2017}
>>> dtFee=pd.DataFrame(kvfee)
>>> dtFee
       2015   2016   2017
Qtr1  40000  42000  43000
Qtr2  35000  37000  38000
Qtr3  47000  49000  50000
Qtr4  45000  47000  48000
```

It is a 2D Dictionary made up of above given dictionaries.

DataFrame object created.

Here, you can get an idea of how index and column name have assigned.

If keys of yr2015, yr2016 and yr2017 were different here then rows and columns of dataframe would have increased and non-matching rows and column would store NaN.

Creation of Dataframe from 2D ndarray

```
>>> import pandas as pd
>>> import numpy as np
>>> narr=np.array([[1,2,3],[4,5,6]],np.int32)
>>> narr.shape
(2, 3)
>>> dtf=pd.DataFrame(narr)
>>> dtf
   0  1  2
0  1  2  3
1  4  5  6
```

column name and index have automatically been generated here.

```
>>> import pandas as pd
>>> import numpy as np
>>> narr=np.array([[1,2,3],[4,5,6]],np.int32)
>>> dtf=pd.DataFrame(narr, columns=['One','Two','Three'])
>>> dtf
   One  Two  Three
0    1    2      3
1    4    5      6
```

Here, user has given column name .

```
>>> dtf=pd.DataFrame(narr, columns=['One','Two','Three'],index=['A','B'])
>>> dtf
   One  Two  Three
A    1    2      3
B    4    5      6
```

Here, column name and index both have given by user.

Creation of DataFarme from 2D Dictionary of same Series Object

```
>>> import pandas as pd
>>> population=pd.Series([35,39,34,64],index=['Class12','Class11','Class10','Cla
ss9'])
>>> AvgMarks=pd.Series([350,390,340,400],index=['Class12','Class11','Class10','C
lass9'])
>>> dict={0:population,1:AvgMarks}
>>> dtf=pd.DataFrame(dict)
>>> dtf
          0    1
Class12  35  350
Class11  39  390
Class10  34  340
Class9   64  400
```

It is a 2D Dictionary made up of series given above.

DataFrame object created.

```
>>> dict={'Population':population,'AverageMarks':AvgMarks}
>>> dtf=pd.DataFrame(dict)
>>> dtf
         Population  AverageMarks
Class12          35           350
Class11          39           390
Class10          34           340
Class9           64           400
```

DataFrame object can also be created like this.

Creation of DataFrame from object of other DataFrame

```
>>> import pandas as pd
>>> import numpy as np
>>> narr=np.array([[1,2,3],[4,5,6]])
>>> dtf=pd.DataFrame(narr,columns=['first','Second','Third'],index=['A','B'])
>>> dtf
   first  Second  Third
A      1       2      3
B      4       5      6
>>> dtf2=pd.DataFrame(dtf)
>>> dtf2
   first  Second  Third
A      1       2      3
B      4       5      6
```

DataFrame object is created from object of other DataFrame.

Displaying DataFrame Object

```
>>> dtf
   first  Second  Third
A      1       2      3
B      4       5      6
>>> dtf2=pd.DataFrame(dtf)
>>> dtf2
   first  Second  Third
A      1       2      3
B      4       5      6
```

Syntax for displaying DataFrame object.

DataFrame Attributes

- When we create an object of a DataFrame then all information related to it like size, datatype etc can be accessed by attributes.

  <DataFrame Object>.<attribute name>

- Some attributes are -

Attribute	Description
index	It shows index of dataframe.
columns	It shows column labels of DataFrame.
axes	It return both the axes i.e. index and column.
dtypes	It returns data type of data contained by dataframe.
size	It returns number of elements in an object.
shape	It returns tuple of dimension of dataframe.
values	It return numpy form of dataframe.
empty	It is an indicator to check whether dataframe is empty or not.
ndim	Return an int representing the number of axes / array dimensions.
T	It Transpose index and columns.

DataFrame Attributes

```
>>> dtf.index
Index(['A', 'B'], dtype='object')
>>> dtf.columns
Index(['first', 'Second', 'Third'], dtype='object')
>>> dtf.axes
[Index(['A', 'B'], dtype='object'), Index(['first', 'Second', 'Third'], dtype='object')
]
>>> dtf.dtypes
first     int32
Second    int32
Third     int32
dtype: object
>>> dtf.size
6
>>> dtf.shape
(2, 3)
>>> dtf.ndim
2
```

```
>>> dtf.empty
False
>>> dtf.count()
first     2
Second    2
Third     2
dtype: int64
>>> dtf.T
          A  B
first     1  4
Second    2  5
Third     3  6
```

```
>>> dtf.values
array([[1, 2, 3],
       [4, 5, 6]])
```

Selecting and Accessing from DataFrame

- Selecting a Column-

<DataFrame Object>[<column name>]

To select a column

or <DataFrame Object>.<column name>

Selection of multiple column

<DataFrame Object>[List of column name]

```
>>> dtf.first
<bound method NDFrame.first of    first  Second  Third
A       1       2       3
B       4       5       6>
>>> dtf['Second']
A    2
B    5
Name: Second, dtype: int32
```

```
>>> dtf[['Second','first']]
   Second  first
A       2      1
B       5      4
```

We can change the order in column.

Selection of subset from DataFrame

<DataFrameObject>.loc [<StartRow> : <EndRow>, <StartCol> : <EndCol>]

```
>>> dtf
         Population  Avg Income  Per Capita Income
Delhi          1001       45000           44.955045
Mumbai         2005       56000           27.930175
Chennai       30236       57000            1.885170
Kolkata        4662       46000            9.867010
```

```
>>> dtf.loc['Delhi',:]
Population              1001.000000
Avg Income             45000.000000
Per Capita Income         44.955045
Name: Delhi, dtype: float64
```

```
>>> dtf.loc[:,'Population':'Per Capita Income']
         Population  Avg Income  Per Capita Income
Delhi          1001       45000           44.955045
Mumbai         2005       56000           27.930175
Chennai       30236       57000            1.885170
Kolkata        4662       46000            9.867010
```

```
>>> dtf.loc['Mumbai':'Kolkata',:]
         Population  Avg Income  Per Capita Income
Mumbai         2005       56000           27.930175
Chennai       30236       57000            1.885170
Kolkata        4662       46000            9.867010
```

```
>>> dtf.loc['Delhi':'Mumbai','Population':'Avg Income']
         Population  Avg Income
Delhi          1001       45000
Mumbai         2005       56000
```

Selection of subset from DataFrame

<DataFrameObject> .iloc [<Row Index> : <RowIndex>, <ColIndex> : <ColIndex>]

```
>>> dtf.iloc[0:2,1:3]
        Avg Income  Per Capita Income
Delhi        45000           44.955045
Mumbai       56000           27.930175
```

```
>>> dtf.iloc[0:2,1:2]
        Avg Income
Delhi        45000
Mumbai       56000
```

Selection of an Individual Value from DataFrame

<DFObject>. <col name.[row name or row index]

or

<DFObject> . at [<row name>,<col name>]

or

<DFObject> iat[<row index>, <col index>]

```
>>> dtf
   One  Two  Three
A    1    2      3
B    4    5      6
>>> dtf.Two['B']
5
```

```
>>> dtf.Two[0]
2
```

```
>>> dtf.at['A','Three']
3
>>> dtf.iat[1,2]
6
```

Accessing and modifying values in DataFrame

a) Syntax to add or change a column-

<DFObject>.<Col Name>[<row label>]=<new value>

```
>>> dtf
   One  Two  Three
A    1    2      3
B    4    5      6
C    7    8      9
>>> dtf['Four']=44
>>> dtf
   One  Two  Three  Four
A    1    2      3    44
B    4    5      6    44
C    7    8      9    44
```

A new column will be created because there is no column with the name 'Four'.

The values of column will get change because there is a column with the name 'Four'.

```
>>> dtf['Four']=66
>>> dtf
   One  Two  Three  Four
A    1    2      3    66
B    4    5      6    66
C    7    8      9    66
```

Accessing and modifying values in DataFrame

b) Syntax to add or change a row-

<DFObject> at[<RowName>, :] =<new value>

या

<DFObject> loc[<RowName>, :] =<new value>

```
>>> dtf.at['D',:]=88
>>> dtf
     One   Two  Three  Four
A    1.0   2.0    3.0  66.0
B    4.0   5.0    6.0  66.0
C    7.0   8.0    9.0  66.0
D   88.0  88.0   88.0  88.0
```

A new row will be created because there is no row with the name 'D'.

```
>>> dtf.at['D',:]=99
>>> dtf
     One   Two  Three  Four
A    1.0   2.0    3.0  66.0
B    4.0   5.0    6.0  66.0
C    7.0   8.0    9.0  66.0
D   99.0  99.0   99.0  99.0
```

The values of row will get change because there is a row with the name 'D'.

Accessing and modifying values in DataFrame

c) Syntax to change single value-

<DFObject>.<ColName>[<RowName/Lebel>]

```
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0  66.0
B   4.0   5.0    6.0  66.0
C   7.0   8.0    9.0  66.0
D  99.0  99.0   99.0  99.0
>>> dtf.Three['D']=100
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0  66.0
B   4.0   5.0    6.0  66.0
C   7.0   8.0    9.0  66.0
D  99.0  99.0  100.0  99.0
```

Here, value of column 'Three' of row 'D' got changed.

```
>>> dtf['Four']=[10,11,12,13]
>>> dtf.at['D']=[13,14,15,16]
>>> dtf
    One   Two  Three  Four
A   1.0   2.0    3.0    10
B   4.0   5.0    6.0    11
C   7.0   8.0    9.0    12
D  13.0  14.0   15.0    16
```

Values can be changed like this also. Values of row and column can be given separately.

Accessing and modifying values in DataFrame

d) Syntax for Column deletion-

del <DFObject>[<ColName>] or

df.drop([<Col1Name>,<Col2Name>, . .], axis=1)

```
>>> del dtf['Four']
>>> dtf
    One   Two  Three
A   1.0   2.0    3.0
B   4.0   5.0    6.0
C   7.0   8.0    9.0
D  13.0  14.0   15.0
>>> dtf.drop(['Two','Three'],axis=1)
    One
A   1.0
B   4.0
C   7.0
D  13.0
```

axis =1 specifies deletion of column.

del command does not return value after deletion whereas drop method returns the value to dataframe after deletion.

Iteration in DataFrame

- Sometimes we need to perform iteration on complete DataFrame. In such cases, it is difficult to write code to access values separately. Therefore, it is necessary to perform iteration on dataframe which is to be done as-

- <DFObject>.iterrows( ) it represents dataframe in row-wise subsets .

- <DFObject>.iteritems( ) it represents dataframe in column-wise subsets.

Use of pandas.iterrows () function

iter.py - C:/Users/KVBBKServer/AppData/Local/Programs/Python/Python36/iter.py (3.6.5)

File Edit Format Run Options Window Help

```
import pandas as pd
disales={2015:{'Qtr1':34500,'Qtr2':56000,'Qtr3':47000,'Qtr4':49000},\
         2016:{'Qtr1':44500,'Qtr2':46100,'Qtr3':57000,'Qtr4':59000},\
         2017:{'Qtr1':54500,'Qtr2':51000,'Qtr3':47000,'Qtr4':58500}}
df1=pd.DataFrame(disales)
for (row,rowSeries) in df1.iterrows():
    print("RowIndex : ",row)
    print("Containing : ")
    print(rowSeries)
```

```
>>> df1
      2015   2016   2017
Qtr1  34500  44500  54500
Qtr2  56000  46100  51000
Qtr3  47000  57000  47000
Qtr4  49000  59000  58500
```

These are the values of df1 which are processed one by one.

```
RowIndex :  Qtr1
Containing :
2015    34500
2016    44500
2017    54500
Name: Qtr1, dtype: int64
RowIndex :  Qtr2
Containing :
2015    56000
2016    46100
2017    51000
Name: Qtr2, dtype: int64
RowIndex :  Qtr3
Containing :
2015    47000
2016    57000
2017    47000
Name: Qtr3, dtype: int64
RowIndex :  Qtr4
Containing :
2015    49000
2016    59000
2017    58500
Name: Qtr4, dtype: int64
>>>
```

Try the code given below after creation of DataFrame.

```
for (row,rowSeries) in df1.iterrows():
    print("RowIndex : ",row)
    print("Containing : ")
    i=0
    for val in rowSeries:
        print("At",i,"Position:",val)
        i=i+1
```

Use of pandas.iteritems() function

```
import pandas as pd
disales={2015:{'Qtr1':34500,'Qtr2':56000,'Qtr3':47000,'Qtr4':49000},\
         2016:{'Qtr1':44500,'Qtr2':46100,'Qtr3':57000,'Qtr4':59000},\
         2017:{'Qtr1':54500,'Qtr2':51000,'Qtr3':47000,'Qtr4':58500}}
df1=pd.DataFrame(disales)
for (col,colSeries) in df1.iteritems():
    print("Column Index : ", col)
    print("Containing : ")
    print(colSeries)
```

```
>>> df1
       2015   2016   2017
Qtr1  34500  44500  54500
Qtr2  56000  46100  51000
Qtr3  47000  57000  47000
Qtr4  49000  59000  58500
```

These are the values of df1 which are processed one by one.

```
Column Index :  2015
Containing :
Qtr1    34500
Qtr2    56000
Qtr3    47000
Qtr4    49000
Name: 2015, dtype: int64
Column Index :  2016
Containing :
Qtr1    44500
Qtr2    46100
Qtr3    57000
Qtr4    59000
Name: 2016, dtype: int64
Column Index :  2017
Containing :
Qtr1    54500
Qtr2    51000
Qtr3    47000
Qtr4    58500
Name: 2017, dtype: int64
>>>
```

Try the code given below after creation of DataFrame.

```
df1=pd.DataFrame(disales)
for (col,colSeries) in df1.iteritems():
    print("Column Index : ", col)
    print("Containing : ")
    i=0
    for val in colSeries:
        print("At row",i," : ", val)
        i=i+1
```

Program for iteration

- Write a program to iterate over a dataframe containing names and marks, then calculates grades as per marks (as per guideline below) and adds them to the grade column.

Marks > =90	Grade A+
Marks 70 – 90	Grade A
Marks 60 – 70	Grade B
Marks 50 – 60	Grade C
Marks 40 – 50	Grade D
Marks < 40	Grade F

Program for iteration

```
import pandas as pd
import numpy as np
names=pd.Series(['Sanjeev','Rajeev','Sanjay','Abhay'])
marks=pd.Series([76,86,55,54])
stud={'Name':names,'Marks':marks}
df=pd.DataFrame(stud,columns=['Name','Marks'])
df['Grade']=np.NaN        #this will add NaN to all records of dataframe
print("Initial values in DataFrame")
print(df)
for (col,colSeries) in df.iteritems():
    length=len(colSeries)
    if col=='Marks':
        lstMrks=[]
        for row in range(length):
            mrks=colSeries[row]
            if mrks>=90:
                lstMrks.append('A+')
            elif mrks>=70:
                lstMrks.append('A')
            elif mrks>=60:
                lstMrks.append('B')
            elif mrks>=50:
                lstMrks.append('C')
            elif mrks>=40:
                lstMrks.append('D')
            else:
                lstMrks.append('F')
df['Grade']=lstMrks
print("\n\nDataFrame after calculation of Grades")
print(df)
```

```
Initial values in DataFrame
      Name  Marks  Grade
0  Sanjeev     76    NaN
1   Rajeev     86    NaN
2   Sanjay     55    NaN
3    Abhay     54    NaN


DataFrame after calculation of Grades
      Name  Marks Grade
0  Sanjeev     76     A
1   Rajeev     86     A
2   Sanjay     55     C
3    Abhay     54     C
```

Binary Operations in a DataFrame

It is possible to perform add, subtract, multiply and devision operations on DataFrame.

To Add - (+, add or radd)

To Subtract - (-, sub or rsub)

To Multiply– (* or mul)

To Divide - (/ or div)

We will perform operations on following dataframes-

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df4
      A     B
0  1000  2000
1  3000  4000
2  5000  6000
```

Addition

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df4
      A     B
0  1000  2000
1  3000  4000
2  5000  6000
```

DataFrame follows index matching to perform arithmetic operations. If matches, operation takes place otherwise it shows NaN (Not a Number). It is called *Data Alignment* in panda object.
This behavior of 'data alignment' on the basis of "matching indexes" is called MATCHING.

```
>>> df1+df3
       A      B      C
0  101.0  202.0  303.0
1  404.0  505.0  606.0
2    NaN    NaN    NaN
>>> df1+df2
    A   B   C
0  11  22  33
1  44  55  66
2  77  88  99
>>> df1+df4
      A     B   C
0  1001  2002 NaN
1  3004  4005 NaN
2  5007  6008 NaN
```

```
>>> df1.add(df2)
    A   B   C
0  11  22  33
1  44  55  66
2  77  88  99
>>> df1.add(df3)
       A      B      C
0  101.0  202.0  303.0
1  404.0  505.0  606.0
2    NaN    NaN    NaN
>>> df1.add(df4)
      A     B   C
0  1001  2002 NaN
1  3004  4005 NaN
2  5007  6008 NaN
```

```
>>> df1.radd(df2)
    A   B   C
0  11  22  33
1  44  55  66
2  77  88  99
>>> df1.radd(df3)
       A      B      C
0  101.0  202.0  303.0
1  404.0  505.0  606.0
2    NaN    NaN    NaN
>>> df1.radd(df4)
      A     B   C
0  1001  2002 NaN
1  3004  4005 NaN
2  5007  6008 NaN
```

Subtraction

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df4
      A     B
0  1000  2000
1  3000  4000
2  5000  6000
```

```
>>> df1-df2
    A   B   C
0  -9 -18 -27
1 -36 -45 -54
2 -63 -72 -81
>>> df1-df3
       A      B      C
0  -99.0 -198.0 -297.0
1 -396.0 -495.0 -594.0
2    NaN    NaN    NaN
>>> df1-df4
      A     B   C
0  -999 -1998 NaN
1 -2996 -3995 NaN
2 -4993 -5992 NaN
```

```
>>> df1.sub(df2)
    A   B   C
0  -9 -18 -27
1 -36 -45 -54
2 -63 -72 -81
>>> df1.sub(df3)
       A      B      C
0  -99.0 -198.0 -297.0
1 -396.0 -495.0 -594.0
2    NaN    NaN    NaN
>>> df1.sub(df4)
      A     B   C
0  -999 -1998 NaN
1 -2996 -3995 NaN
2 -4993 -5992 NaN
```

```
>>> df1.rsub(df2)
    A   B   C
0   9  18  27
1  36  45  54
2  63  72  81
>>> df1.rsub(df3)
       A      B      C
0   99.0  198.0  297.0
1  396.0  495.0  594.0
2    NaN    NaN    NaN
>>> df1.rsub(df4)
      A     B   C
0   999  1998 NaN
1  2996  3995 NaN
2  4993  5992 NaN
```

Multiplication

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df4
      A     B
0  1000  2000
1  3000  4000
2  5000  6000
```

```
>>> df1*df2
     A    B    C
0   10   40   90
1  160  250  360
2  490  640  810
>>> df1*df3
       A      B      C
0  100.0  400.0  900.0
1 1600.0 2500.0 3600.0
2    NaN    NaN    NaN
>>> df1*df4
       A      B   C
0   1000   4000 NaN
1  12000  20000 NaN
2  35000  48000 NaN
```

```
>>> df1.mul(df2)
     A    B    C
0   10   40   90
1  160  250  360
2  490  640  810
>>> df1.mul(df3)
       A      B      C
0  100.0  400.0  900.0
1 1600.0 2500.0 3600.0
2    NaN    NaN    NaN
>>> df1.mul(df4)
       A      B   C
0   1000   4000 NaN
1  12000  20000 NaN
2  35000  48000 NaN
```

Division

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df4
      A     B
0  1000  2000
1  3000  4000
2  5000  6000
```

```
>>> df1/df2
     A    B    C
0  0.1  0.1  0.1
1  0.1  0.1  0.1
2  0.1  0.1  0.1
>>> df2/df1
      A     B     C
0  10.0  10.0  10.0
1  10.0  10.0  10.0
2  10.0  10.0  10.0
>>> df1/df3
      A     B     C
0  0.01  0.01  0.01
1  0.01  0.01  0.01
2   NaN   NaN   NaN
>>> df3/df1
       A      B      C
0  100.0  100.0  100.0
1  100.0  100.0  100.0
2    NaN    NaN    NaN
```

See the operation of the rdiv carefully

```
>>> df1.div(df2)
     A    B    C
0  0.1  0.1  0.1
1  0.1  0.1  0.1
2  0.1  0.1  0.1
>>> df1.rdiv(df2)
      A     B     C
0  10.0  10.0  10.0
1  10.0  10.0  10.0
2  10.0  10.0  10.0
>>> df1.div(df3)
      A     B     C
0  0.01  0.01  0.01
1  0.01  0.01  0.01
2   NaN   NaN   NaN
>>> df1.rdiv(df3)
       A      B      C
0  100.0  100.0  100.0
1  100.0  100.0  100.0
2    NaN    NaN    NaN
```

Other important functions

Other important functions of DataFrame are as under-

<DF>.info ( )

<DF>.describe ( )

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df4
      A     B
0  1000  2000
1  3000  4000
2  5000  6000
```

```
>>> df1.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
A    3 non-null int32
B    3 non-null int32
C    3 non-null int32
dtypes: int32(3)
memory usage: 116.0 bytes
```

```
>>> df1.describe()
              A    B    C
count   3.0  3.0  3.0
mean    4.0  5.0  6.0
std     3.0  3.0  3.0
min     1.0  2.0  3.0
25%     2.5  3.5  4.5
50%     4.0  5.0  6.0
75%     5.5  6.5  7.5
max     7.0  8.0  9.0
```

Other important functions

Other important functions of DataFrame are as under-

<DF>.head ([n=<n>]) here, default value of n is 5.

<DF>.tail ([n=<n>])

```
>>> df1
     A    B    C
0    1    2    3
1    4    5    6
2    7    8    9
3   10   20   30
4   40   50   60
5   70   80   90
6  100  200  300
7  400  500  600
8  700  800  900
```

```
>>> df1.head()
    A   B   C
0   1   2   3
1   4   5   6
2   7   8   9
3  10  20  30
4  40  50  60
>>> df1.tail()
     A    B    C
4   40   50   60
5   70   80   90
6  100  200  300
7  400  500  600
8  700  800  900
```

```
>>> df1.head(n=3)
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
>>> df1.tail(n=4)
     A    B    C
5   70   80   90
6  100  200  300
7  400  500  600
8  700  800  900
```

Cumulative Calculations Functions

In DataFrame, for cumulative sum, function is as under-

<DF>.cumsum([axis = None]) here, axis argument is optional.

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df1.cumsum()
    A   B   C
0   1   2   3
1   5   7   9
2  12  15  18
```

```
>>> df1.cumsum(axis='rows')
    A   B   C
0   1   2   3
1   5   7   9
2  12  15  18
>>> df1.cumsum(axis='columns')
   A   B   C
0  1   3   6
1  4   9  15
2  7  15  24
```

Index of Maximum and Minimum Values

```
>>> df5
    A    B    C
0   1    2    3
1   4    5    6
2   7    8    9
3  10   20   30
4  40   50   60
5  70   80   90
6 100  200  300
7 400  500  600
8 700  800  900
```

<DF>.idxmax ( )

<DF>.idxmin ( )

```
>>> df5.idxmax()
A    8
B    8
C    8
dtype: int64
>>> df5.idxmin()
A    0
B    0
C    0
dtype: int64
>>>
```

Handling of Missing Data

- The values with no computational significance are called missing values.
- Handling methods for missing values-
  - Dropping missing data
  - Filling missing data (Imputation)

```
>>> df10
      A     B   C
0  1001  2002 NaN
1  3004  4005 NaN
2  5007  6008 NaN
```

```
>>> df11.fillna(0)
      A     B    C
0  1001  2002  0.0
1  3004  4005  0.0
2  5007  6008  0.0
```

```
>>> df11=df10.dropna()
>>> df11
Empty DataFrame
Columns: [A, B, C]
Index: []
>>> df11=df10.dropna(how='all')
>>> df11
      A     B   C
0  1001  2002 NaN
1  3004  4005 NaN
2  5007  6008 NaN
```

Comparison of Pandas Objects

```
>>> df1
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

```
>>> df2
    A   B   C
0  10  20  30
1  40  50  60
2  70  80  90
```

```
>>> df3
     A    B    C
0  100  200  300
1  400  500  600
```

```
>>> df12
       A      B      C
0  101.0  202.0  303.0
1  404.0  505.0  606.0
2    NaN    NaN    NaN
```

```
>>> df1+df2==df1.add(df2)
      A     B     C
0  True  True  True
1  True  True  True
2  True  True  True
>>> df1+df3==df1.add(df3)
       A      B      C
0   True   True   True
1   True   True   True
2  False  False  False
```

```
>>> df1>5
       A      B      C
0  False  False  False
1  False  False   True
2   True   True   True
```

```
>>> (df1+df3).equals(df1.add(df3))
True
```

equals () checks both the objects for equality.

Thank you

Please follow us on our blog

www.pythontrends.wordpress.com