Basic Functionality

By now, we learnt about the three Pandas DataStructures and how to create them. We will majorly focus on the DataFrame objects because of its importance in the real time data processing and also discuss a few other DataStructures.

Series Basic Functionality

Sr.No.	Attribute or Method & Description
1	**axes** Returns a list of the row axis labels
2	**dtype** Returns the dtype of the object.
3	**empty** Returns True if series is empty.
4	**ndim** Returns the number of dimensions of the underlying data, by definition 1.
5	**size** Returns the number of elements in the underlying data.
6	**values** Returns the Series as ndarray.
7	**head()** Returns the first n rows.
8	**tail()**

	Returns the last n rows.

Let us now create a Series and see all the above tabulated attributes operation.

Example

Live Demo

```
import pandas as pd
import numpy as np

#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print s
```

Its **output** is as follows −

```
0   0.967853
1  -0.148368
2  -1.395906
3  -1.758394
dtype: float64
```

axes

Returns the list of the labels of the series.

Live Demo

```
import pandas as pd
import numpy as np

#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print ("The axes are:")
print s.axes
```

Its **output** is as follows −

```
The axes are:
[RangeIndex(start=0, stop=4, step=1)]
```

The above result is a compact format of a list of values from 0 to 5, i.e., [0,1,2,3,4].

empty

Returns the Boolean value saying whether the Object is empty or not. True indicates that the object is empty.

Live Demo

```
import pandas as pd
```

```
import numpy as np

#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print ("Is the Object empty?")
print s.empty
```

Its **output** is as follows −

```
Is the Object empty?
False
```

ndim

Returns the number of dimensions of the object. By definition, a Series is a 1D data structure, so it returns

Live Demo

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print s

print ("The dimensions of the object:")
print s.ndim
```

Its **output** is as follows −

```
0   0.175898
1   0.166197
2  -0.609712
3  -1.377000
dtype: float64

The dimensions of the object:
1
```

size

Returns the size(length) of the series.

Live Demo

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(2))
print s
```

```
print ("The size of the object:")
print s.size
```

Its **output** is as follows −

```
0   3.078058
1  -1.207803
dtype: float64

The size of the object:
2
```

values

Returns the actual data in the series as an array.

Live Demo

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print s

print ("The actual data series is:")
print s.values
```

Its **output** is as follows −

```
0   1.787373
1  -0.605159
2   0.180477
3  -0.140922
dtype: float64

The actual data series is:
[ 1.78737302 -0.60515881 0.18047664 -0.1409218 ]
```

Head & Tail

To view a small sample of a Series or the DataFrame object, use the head() and the tail() methods.

head() returns the first **n** rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

Live Demo

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
```

```
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print s

print ("The first two rows of the data series:")
print s.head(2)
```

Its **output** is as follows −

```
The original series is:
0   0.720876
1  -0.765898
2   0.479221
3  -0.139547
dtype: float64

The first two rows of the data series:
0   0.720876
1  -0.765898
dtype: float64
```

tail() returns the last **n** rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

Live Demo

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print s

print ("The last two rows of the data series:")
print s.tail(2)
```

Its **output** is as follows −

```
The original series is:
0 -0.655091
1 -0.881407
2 -0.608592
3 -2.341413
dtype: float64

The last two rows of the data series:
2 -0.608592
3 -2.341413
dtype: float64
```

DataFrame Basic Functionality

Let us now understand what DataFrame Basic Functionality is. The following tables lists down the important attributes or methods that help in DataFrame Basic Functionality.

Sr.No.	Attribute or Method & Description
1	**T** Transposes rows and columns.
2	**axes** Returns a list with the row axis labels and column axis labels as the only members.
3	**dtypes** Returns the dtypes in this object.
4	**empty** True if NDFrame is entirely empty [no items]; if any of the axes are of length 0.
5	**ndim** Number of axes / array dimensions.
6	**shape** Returns a tuple representing the dimensionality of the DataFrame.
7	**size** Number of elements in the NDFrame.
8	**values** Numpy representation of NDFrame.
9	**head()** Returns the first n rows.

10	**tail()** Returns last n rows.

Let us now create a DataFrame and see all how the above mentioned attributes operate.

Example

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our data series is:")
print df
```

Its **output** is as follows −

```
Our data series is:
    Age   Name    Rating
0   25    Tom     4.23
1   26    James   3.24
2   25    Ricky   3.98
3   23    Vin     2.56
4   30    Steve   3.20
5   29    Smith   4.60
6   23    Jack    3.80
```

T (Transpose)

Returns the transpose of the DataFrame. The rows and columns will interchange.

Live Demo

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
```

```
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("The transpose of the data series is:")
print df.T
```

Its **output** is as follows −

```
The transpose of the data series is:
          0     1       2      3      4      5       6
Age       25    26      25     23     30     29      23
Name      Tom   James   Ricky  Vin    Steve  Smith   Jack
Rating    4.23  3.24    3.98   2.56   3.2    4.6     3.8
```

axes

Returns the list of row axis labels and column axis labels.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Row axis labels and column axis labels are:")
print df.axes
```

Its **output** is as follows −

```
Row axis labels and column axis labels are:

[RangeIndex(start=0, stop=7, step=1), Index([u'Age', u'Name',
u'Rating'],
dtype='object')]
```

dtypes

Returns the data type of each column.

Live Demo

```
import pandas as pd
```

```
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("The data types of each column are:")
print df.dtypes
```

Its **output** is as follows −

```
The data types of each column are:
Age      int64
Name    object
Rating  float64
dtype: object
```

empty

Returns the Boolean value saying whether the Object is empty or not; True indicates that the object is empty.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Is the object empty?")
print df.empty
```

Its **output** is as follows −

```
Is the object empty?
False
```

ndim

Returns the number of dimensions of the object. By definition, DataFrame is a 2D object.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print df
print ("The dimension of the object is:")
print df.ndim
```

Its **output** is as follows −

```
Our object is:
      Age    Name     Rating
0     25     Tom      4.23
1     26     James    3.24
2     25     Ricky    3.98
3     23     Vin      2.56
4     30     Steve    3.20
5     29     Smith    4.60
6     23     Jack     3.80

The dimension of the object is:
2
```

shape

Returns a tuple representing the dimensionality of the DataFrame. Tuple (a,b), where a represents the number of rows and **b** represents the number of columns.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}
```

```
#Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print df
print ("The shape of the object is:")
print df.shape
```

Its **output** is as follows −

```
Our object is:
   Age   Name    Rating
0  25    Tom     4.23
1  26    James   3.24
2  25    Ricky   3.98
3  23    Vin     2.56
4  30    Steve   3.20
5  29    Smith   4.60
6  23    Jack    3.80

The shape of the object is:
(7, 3)
```

size

Returns the number of elements in the DataFrame.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print df
print ("The total number of elements in our object is:")
print df.size
```

Its **output** is as follows −

```
Our object is:
    Age   Name    Rating
0   25    Tom     4.23
1   26    James   3.24
```

```
2    25    Ricky    3.98
3    23    Vin      2.56
4    30    Steve    3.20
5    29    Smith    4.60
6    23    Jack     3.80

The total number of elements in our object is:
21
```

values

Returns the actual data in the DataFrame as an **NDarray.**

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print df
print ("The actual data in our data frame is:")
print df.values
```

Its **output** is as follows −

```
Our object is:
    Age    Name     Rating
0    25    Tom      4.23
1    26    James    3.24
2    25    Ricky    3.98
3    23    Vin      2.56
4    30    Steve    3.20
5    29    Smith    4.60
6    23    Jack     3.80
The actual data in our data frame is:
[[25 'Tom' 4.23]
[26 'James' 3.24]
[25 'Ricky' 3.98]
[23 'Vin' 2.56]
[30 'Steve' 3.2]
[29 'Smith' 4.6]
[23 'Jack' 3.8]]
```

Head & Tail

To view a small sample of a DataFrame object, use the **head()** and tail() methods. **head()** returns the first **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our data frame is:")
print df
print ("The first two rows of the data frame is:")
print df.head(2)
```

Its **output** is as follows −

```
Our data frame is:
    Age   Name    Rating
0   25    Tom     4.23
1   26    James   3.24
2   25    Ricky   3.98
3   23    Vin     2.56
4   30    Steve   3.20
5   29    Smith   4.60
6   23    Jack    3.80

The first two rows of the data frame is:
   Age   Name   Rating
0  25    Tom    4.23
1  26    James  3.24
```

tail() returns the last **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
```

```
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our data frame is:")
print df
print ("The last two rows of the data frame is:")
print df.tail(2)
```

Its **output** is as follows −

```
Our data frame is:
    Age   Name    Rating
0   25    Tom     4.23
1   26    James   3.24
2   25    Ricky   3.98
3   23    Vin     2.56
4   30    Steve   3.20
5   29    Smith   4.60
6   23    Jack    3.80

The last two rows of the data frame is:
    Age   Name    Rating
5   29    Smith    4.6
6   23    Jack     3.8
```

Descriptive Statstics

A large number of methods collectively compute descriptive statistics and other related operations on DataFrame. Most of these are aggregations like **sum(), mean(),** but some of them, like **sumsum()**, produce an object of the same size. Generally speaking, these methods take an **axis** argument, just like *ndarray.{sum, std, ...},* but the axis can be specified by name or integer

- **DataFrame** − “index” (axis=0, default), “columns” (axis=1)

Let us create a DataFrame and use this object throughout this chapter for all the operations.

Example

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df
```

Its **output** is as follows −

```
    Age  Name   Rating
0   25   Tom     4.23
1   26   James   3.24
2   25   Ricky   3.98
3   23   Vin     2.56
4   30   Steve   3.20
5   29   Smith   4.60
6   23   Jack    3.80
7   34   Lee     3.78
8   40   David   2.98
9   30   Gasper  4.80
10  51   Betina  4.10
11  46   Andres  3.65
```

sum()

Returns the sum of the values for the requested axis. By default, axis is index (axis=0).

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
```

```
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df.sum()
```

Its **output** is as follows −

```
Age                                                382
Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Rating                                             44.92
dtype: object
```

Each individual column is added individually (Strings are appended).

axis=1

This syntax will give the output as shown below.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df.sum(1)
```

Its **output** is as follows −

```
0    29.23
1    29.24
2    28.98
3    25.56
4    33.20
5    33.60
6    26.80
```

```
7     37.78
8     42.98
9     34.80
10    55.10
11    49.65
dtype: float64
```

mean()

Returns the average value

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df.mean()
```

Its **output** is as follows −

```
Age       31.833333
Rating     3.743333
dtype: float64
```

std()

Returns the Bressel standard deviation of the numerical columns.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
```

```
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df.std()
```

Its **output** is as follows −

```
Age        9.232682
Rating     0.661628
dtype: float64
```

Functions & Description

Let us now understand the functions under Descriptive Statistics in Python Pandas. The following table list down the important functions −

Sr.No.	Function	Description
1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values
5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()	Minimum Value
8	max()	Maximum Value

9	abs()	Absolute Value
10	prod()	Product of Values
11	cumsum()	Cumulative Sum
12	cumprod()	Cumulative Product

Note − Since DataFrame is a Heterogeneous data structure. Generic operations don't work with all functions.

- Functions like **sum(), cumsum()** work with both numeric and character (or) string data elements without any error. Though **n** practice, character aggregations are never used generally, these functions do not throw any exception.
- Functions like **abs(), cumprod()** throw exception when the DataFrame contains character or string data because such operations cannot be performed.

Summarizing Data

The **describe()** function computes a summary of statistics pertaining to the DataFrame columns.

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df.describe()
```

Its **output** is as follows −

```
              Age          Rating
```

```
count    12.000000      12.000000
mean     31.833333       3.743333
std       9.232682       0.661628
min      23.000000       2.560000
25%      25.000000       3.230000
50%      29.500000       3.790000
75%      35.500000       4.132500
max      51.000000       4.800000
```

This function gives the **mean, std** and **IQR** values. And, function excludes the character columns and given summary about numeric columns. **'include'** is the argument which is used to pass necessary information regarding what columns need to be considered for summarizing. Takes the list of values; by default, 'number'.

- **object** − Summarizes String columns
- **number** − Summarizes Numeric columns
- **all** − Summarizes all columns together (Should not pass it as a list value)

Now, use the following statement in the program and check the output −

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df.describe(include=['object'])
```

Its **output** is as follows −

```
          Name
count       12
unique      12
top      Ricky
freq         1
```

Now, use the following statement and check the output −

Live Demo

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jac
k',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),

'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80
,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print df. describe(include='all')
```

Its **output** is as follows −

```
          Age          Name       Rating
count   12.000000        12    12.000000
unique        NaN        12          NaN
top           NaN     Ricky          NaN
freq          NaN         1          NaN
mean    31.833333       NaN     3.743333
std      9.232682       NaN     0.661628
min     23.000000       NaN     2.560000
25%     25.000000       NaN     3.230000
50%     29.500000       NaN     3.790000
75%     35.500000       NaN     4.132500
max     51.000000       NaN     4.800000
```