DATA FRAMES IN PYTHON

PANDAS

In computer programming, **pandas** is a software library written for the **Python** programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

Why do we need pandas in Python?

pandas is a **Python** package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in **Python**.

Objective

This lesson has been prepared to learn the basics and various functions of Pandas. It will be specifically useful for people working with data cleansing and analysis. After completing this chapter, you will find yourself at a moderate level of expertise from where you can take yourself to higher levels of expertise.

CHARACTERISTICS OF DATA FRAME:

1. has two axes – row index or column index
2. like a spreadsheet where each value is identifiable with the combination of row index and column index
3. indices can be of number or letter or strings
4. can easily change its value

For working in pandas generally we import both numpy and pandas. We import numpy because sometimes numpy function are also needed by giving import statements.

Import numpy as np
Import pandas as pd

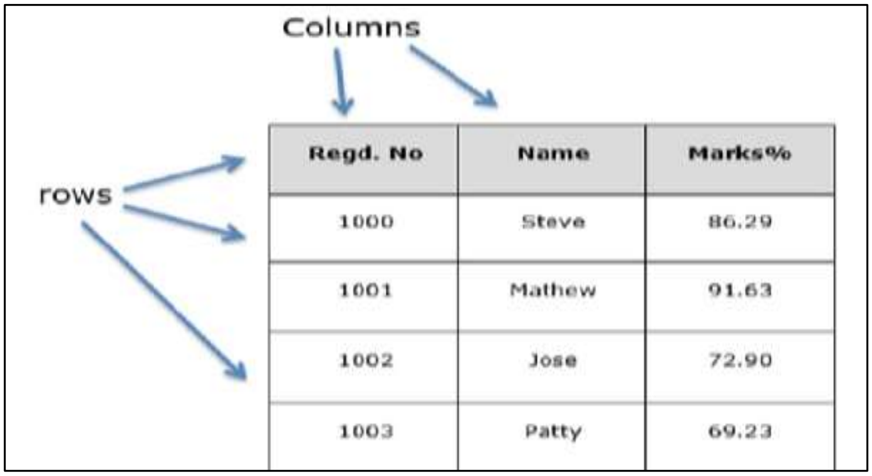
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Features of DataFrame

- Potentially columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

Structure

Let us assume that we are creating a data frame with student's data.



Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23

You can think of it as an SQL table or a spreadsheet data representation.

pandas.DataFrame

A pandas DataFrame can be created using the following constructor −

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

The parameters of the constructor are as follows −

1	**data** data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**index** For the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed.
3	**columns** For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.
4	**dtype** Data type of each column.
5	**copy** This command (or whatever it is) is used for copying of data, if the default is False.

Create DataFrame

A pandas DataFrame can be created using various inputs like −

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

In the subsequent sections of this chapter, we will see how to create a DataFrame using these inputs.

Create an Empty DataFrame

A basic DataFrame, which can be created is an Empty Dataframe.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame()
print df
```

Its **output** is as follows −

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

create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

Example 1

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows −

```
     0
0    1
1    2
2    3
3    4
```

```
4     5
```

Example 2

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Its **output** is as follows −

```
      Name      Age
0     Alex      10
1     Bob       12
2     Clarke    13
```

Example 3

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)
print df
```

Its **output** is as follows −

```
     Name     Age
0     Alex     10.0
1     Bob      12.0
2     Clarke   13.0
```

Create a DataFrame from Dict of ndarrays / Lists

All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

If no index is passed, then by default, index will be range(n), where **n** is the array length.

Example 1

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve',
'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

Its **output** is as follows −

```
      Age      Name
0     28        Tom
1     34       Jack
2     29      Steve
```

```
3      42       Ricky
```

Note **− Observe the values 0,1,2,3. They are the default index assigned to each using the function range(n).**

Example 2

Let us now create an indexed DataFrame using arrays.

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve',
'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print df
```

Its **output** is as follows −

```
         Age     Name
rank1    28        Tom
rank2    34       Jack
rank3    29      Steve
rank4    42      Ricky
```

Note − Observe, the **index** parameter assigns an index to each row.

Create a DataFrame from List of Dicts

List of Dictionaries can be passed as input data to create a DataFrame. The dictionary keys are by default taken as column names.

Example 1

The following example shows how to create a DataFrame by passing a list of dictionaries.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows −

```
    a    b      c
0   1    2     NaN
1   5    10   20.0
```

Note − Observe, NaN (Not a Number) is appended in missing areas.

Example 2

The following example shows how to create a DataFrame by passing a list of dictionaries and the row indices.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print df
```

Its **output** is as follows −

```
        a   b       c
first   1   2     NaN
second  5   10   20.0
```

Example 3

The following example shows how to create a DataFrame with a list of dictionaries, row indices, and column indices.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

#With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a',
'b'])

#With two column indices with one index with other name
df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a',
'b1'])
print df1
print df2
```

Its **output** is as follows –

```
#df1 output
         a  b
first    1  2
second   5  10

#df2 output
         a  b1
first    1  NaN
second   5  NaN
```

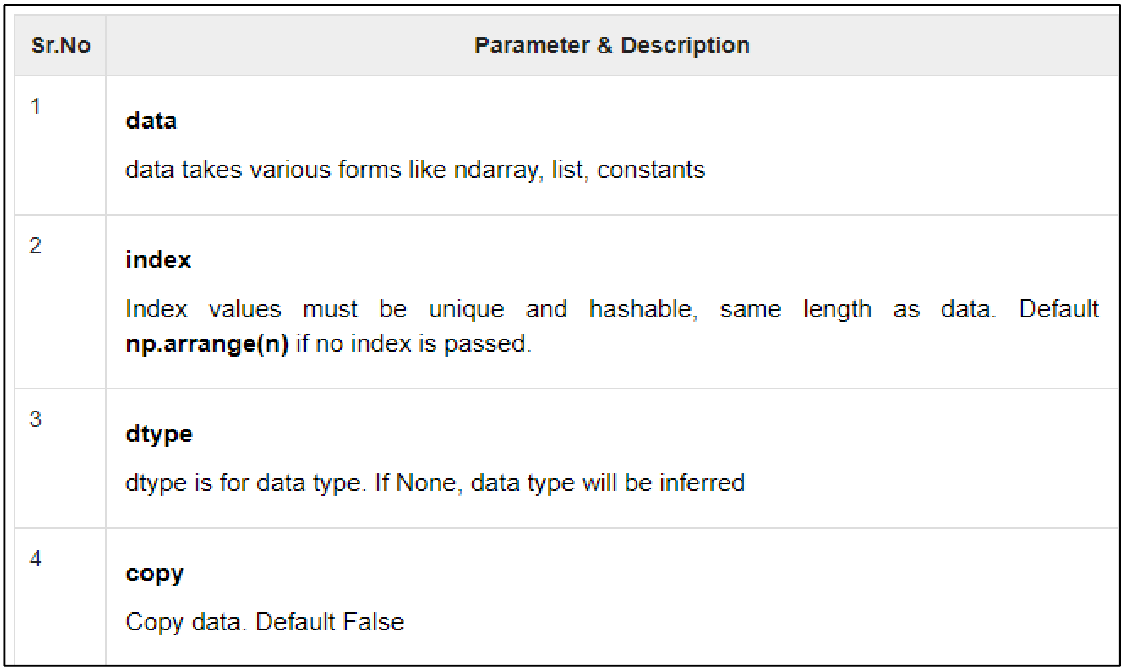
Note − Observe, df2 DataFrame is created with a column index other than the dictionary key; thus, appended the NaN's in place. Whereas, df1 is created with column indices same as dictionary keys, so NaN's appended.

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

pandas.Series

A pandas Series can be created using the following constructor −

```
pandas.Series( data, index, dtype, copy)
```



Sr.No	Parameter & Description
1	**data** data takes various forms like ndarray, list, constants
2	**index** Index values must be unique and hashable, same length as data. Default **np.arrange(n)** if no index is passed.
3	**dtype** dtype is for data type. If None, data type will be inferred
4	**copy** Copy data. Default False

A series can be created using various inputs like −

- Array
- Dict
- Scalar value or constant

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print s
```

Its **output** is as follows −

```
Series([], dtype: float64)
```

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np

data = np.array(['a','b','c','d'])
s = pd.Series(data)
print s
```

Its **output** is as follows −

```
0    a
1    b
2    c
3    d
dtype: object
```

We did not pass any index, so by default, it assigned the indexes ranging from 0 to **len(data)-1**, i.e., 0 to 3.

Example

```
import pandas as pd
import numpy as np
info = np.array(['P','a','n','d','a','s'])
a = pd.Series(info)
print(a)
```

Output

```
0     P
1     a
2     n
3     d
4     a
5     s
dtype: object
```

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print s
```

Its **output** is as follows −

```
100   a
101   b
102   c
103   d
```

We passed the index values here. Now we can see the customized indexed values in the output.

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print s
```

Its **output** is as follows −

```
b 1.0
c 2.0
d NaN
a 0.0
dtype: float64
```

Observe − Index order is persisted and the missing element is filled with NaN (Not a Number).

Column Selection

We will understand this by selecting a column from the DataFrame.

Example

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df ['one']
```

Its **output** is as follows −

```
a      1.0
b      2.0
c      3.0
d      NaN
Name: one, dtype: float64
```

Column Addition

We will understand this by adding a new column to an existing data frame.

Example

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

# Adding a new column to an existing DataFrame object with column
label by passing new series

print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print df

print ("Adding a new column using the existing columns in
DataFrame:")
df['four']=df['one']+df['three']

print(df)
```

Its **output** is as follows −

```
Adding a new column by passing as Series:
     one   two   three
a    1.0    1    10.0
b    2.0    2    20.0
c    3.0    3    30.0
d    NaN    4    NaN

Adding a new column using the existing columns in DataFrame:
      one   two   three    four
a     1.0    1    10.0     11.0
b     2.0    2    20.0     22.0
c     3.0    3    30.0     33.0
d     NaN    4     NaN     NaN
```

Example

```
# Using the previous DataFrame, we will delete a column
# using del function
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
   'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print ("Our dataframe is:")
print df

# using del function
print ("Deleting the first column using DEL function:")
del df['one']
print df

# using pop function
print ("Deleting another column using POP function:")
df.pop('two')

print df
```

Its **output** is as follows −

```
Our dataframe is:
      one    three  two
a     1.0    10.0   1
b     2.0    20.0   2
c     3.0    30.0   3
d     NaN     NaN   4

Deleting the first column using DEL function:
      three    two
a     10.0     1
b     20.0     2
c     30.0     3
d     NaN      4

Deleting another column using POP function:
   three
a  10.0
b  20.0
c  30.0
d  NaN
```