

# DataFrame

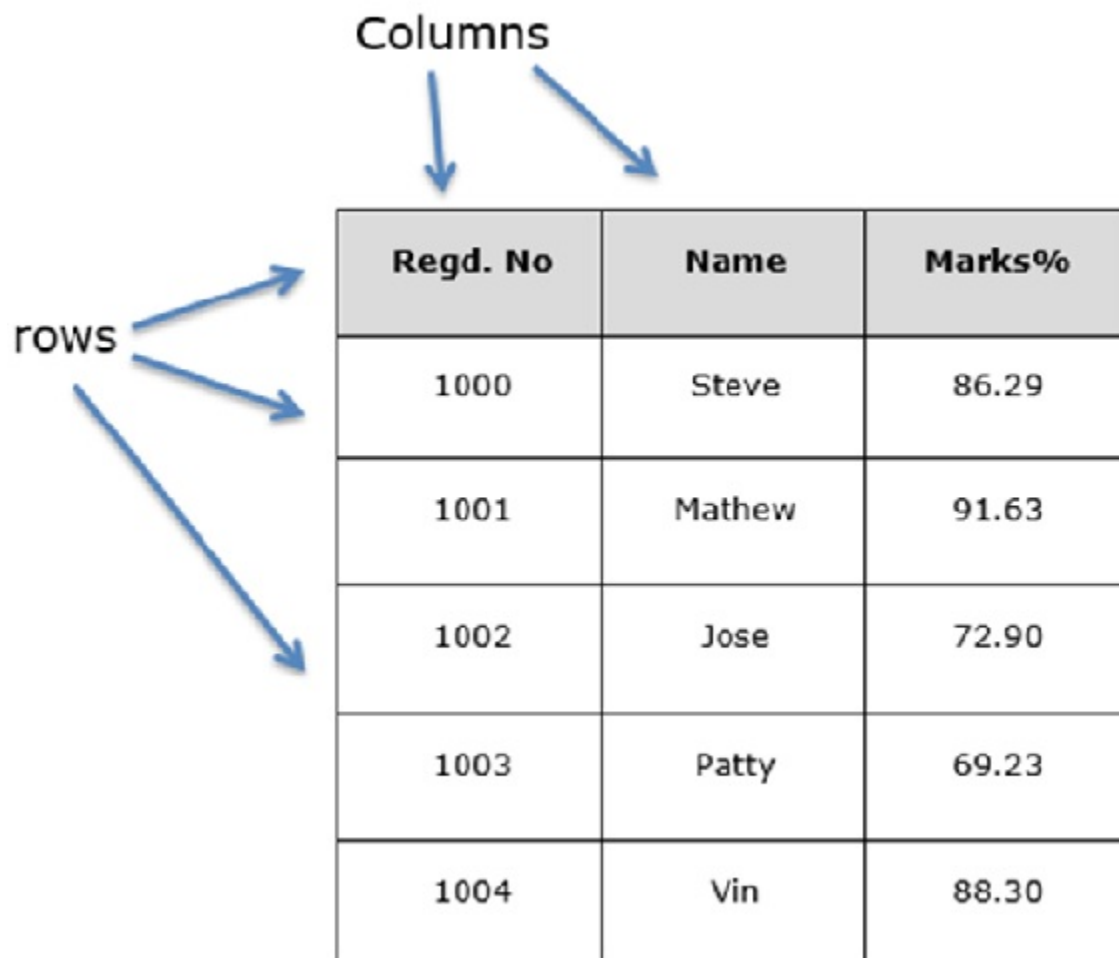
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

## 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.



The diagram illustrates the structure of a DataFrame using a table of student data. The word "Columns" is positioned above the table with two blue arrows pointing to the "Regd. No" and "Name" headers. The word "rows" is positioned to the left of the table with three blue arrows pointing to the first three rows of data.

| Regd. No | Name   | Marks% |
|----------|--------|--------|
| 1000     | Steve  | 86.29  |
| 1001     | Mathew | 91.63  |
| 1002     | Jose   | 72.90  |
| 1003     | Patty  | 69.23  |
| 1004     | Vin    | 88.30  |

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 –

| Sr.No | Parameter & Description  |
|-------|--|
| 1     | <b>data</b><br>data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.                    |
| 2     | <b>index</b><br>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     | <b>columns</b><br>For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.             |
| 4     | <b>dtype</b><br>Data type of each column.  |
| 5     | <b>copy</b><br>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

[Live Demo](#)

```
#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

[Live Demo](#)

```
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

[Live Demo](#)

```
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

[Live Demo](#)

```
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 |

**Note** – Observe, the **dtype** parameter changes the type of Age column to floating point.

## 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

[Live Demo](#)

```
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.

[Live Demo](#)

```
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.

[Live Demo](#)

```
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.

[Live Demo](#)

```
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.

[Live Demo](#)

```
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.

## Create a DataFrame from Dict 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

[Live Demo](#)

```
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
```

Its output is as follows –

|   | one | two |
|---|-----|-----|
| a | 1.0 | 1   |
| b | 2.0 | 2   |
| c | 3.0 | 3   |
| d | NaN | 4   |

**Note** – Observe, for the series one, there is no label ‘d’ passed, but in the result, for the **d** label, NaN is appended with NaN.

Let us now understand **column selection**, **addition**, and **deletion** through examples.

## Column Selection

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

## Example

[Live Demo](#)

```
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

[Live Demo](#)

```
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  |

## Column Deletion

Columns can be deleted or popped; let us take an example to understand how.

## Example

[Live Demo](#)



```

# 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   |

## Row Selection, Addition, and Deletion

We will now understand row selection, addition and deletion through examples. Let us begin with the concept of selection.

## Selection by Label

Rows can be selected by passing row label to a **loc** function.

[Live Demo](#)

```
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.loc['b']
```

Its output is as follows –

```
one 2.0
two 2.0
Name: b, dtype: float64
```

The result is a series with labels as column names of the DataFrame. And, the Name of the series is the label with which it is retrieved.

## Selection by integer location

Rows can be selected by passing integer location to an **iloc** function.

[Live Demo](#)

```
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.iloc[2]
```

Its output is as follows –

```
one 3.0
two 3.0
Name: c, dtype: float64
```

## Slice Rows

Multiple rows can be selected using ‘:’ operator.

[Live Demo](#)

```
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[2:4]
```

Its output is as follows –

```
   one  two
c  3.0    3
d  NaN    4
```

## Addition of Rows

Add new rows to a DataFrame using the **append** function. This function will append the rows at the end.

[Live Demo](#)

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a', 'b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a', 'b'])

df = df.append(df2)
print df
```

Its output is as follows –

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

## Deletion of Rows

Use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped.

If you observe, in the above example, the labels are duplicate. Let us drop a label and will see how many rows will get dropped.

[Live Demo](#)

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a', 'b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a', 'b'])

df = df.append(df2)

# Drop rows with label 0
df = df.drop(0)

print df
```

Its **output** is as follows –

```
   a b
1  3 4
1  7 8
```

In the above example, two rows were dropped because those two contain the same label 0

## Panel

A **panel** is a 3D container of data. The term **Panel data** is derived from econometrics and is partially responsible for the name pandas – **pan(el)-da(ta)-s**.

The names for the 3 axes are intended to give some semantic meaning to describing operations involving panel data. They are –

- **items** – axis 0, each item corresponds to a DataFrame contained inside.
- **major\_axis** – axis 1, it is the index (rows) of each of the DataFrames.
- **minor\_axis** – axis 2, it is the columns of each of the DataFrames.

## pandas.Panel()

A Panel can be created using the following constructor –

```
pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)
```

The parameters of the constructor are as follows –

| Parameter  | Description   |
|------------|---|
| data       | Data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame |
| items      | axis=0  |
| major_axis | axis=1  |
| minor_axis | axis=2  |

|       |                                  |
|-------|----------------------------------|
| dtype | Data type of each column         |
| copy  | Copy data. Default, <b>false</b> |

## Create Panel

A Panel can be created using multiple ways like –

- From ndarrays
- From dict of DataFrames

### From 3D ndarray

[Live Demo](#)

```
# creating an empty panel
import pandas as pd
import numpy as np

data = np.random.rand(2,4,5)
p = pd.Panel(data)
print p
```

Its output is as follows –

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 5 (minor_axis)
Items axis: 0 to 1
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 4
```

**Note** – Observe the dimensions of the empty panel and the above panel, all the objects are different.

### From dict of DataFrame Objects

[Live Demo](#)

```
#creating an empty panel
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p
```

Its output is as follows –

Dimensions: 2 (items) x 4 (major\_axis) x 3 (minor\_axis)  
Items axis: Item1 to Item2  
Major\_axis axis: 0 to 3  
Minor\_axis axis: 0 to 2

## Create an Empty Panel

An empty panel can be created using the Panel constructor as follows –

[Live Demo](#)

```
#creating an empty panel
import pandas as pd
p = pd.Panel()
print p
```

Its output is as follows –

```
<class 'pandas.core.panel.Panel'>
Dimensions: 0 (items) x 0 (major_axis) x 0 (minor_axis)
Items axis: None
Major_axis axis: None
Minor_axis axis: None
```

## Selecting the Data from Panel

Select the data from the panel using –

- Items
- Major\_axis
- Minor\_axis

## Using Items

[Live Demo](#)

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p['Item1']
```

Its output is as follows –

|   | 0         | 1         | 2        |
|---|-----------|-----------|----------|
| 0 | 0.488224  | -0.128637 | 0.930817 |
| 1 | 0.417497  | 0.896681  | 0.576657 |
| 2 | -2.775266 | 0.571668  | 0.290082 |
| 3 | -0.400538 | -0.144234 | 1.110535 |

We have two items, and we retrieved item1. The result is a DataFrame with 4 rows and 3 columns, which are the **Major\_axis** and **Minor\_axis** dimensions.

## Using major\_axis

Data can be accessed using the method **panel.major\_axis(index)**.

[Live Demo](#)

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p.major_xs(1)
```

Its output is as follows –

|   | Item1    | Item2     |
|---|----------|-----------|
| 0 | 0.417497 | 0.748412  |
| 1 | 0.896681 | -0.557322 |
| 2 | 0.576657 | NaN       |

## Using minor\_axis

Data can be accessed using the method **panel.minor\_axis(index)**.

[Live Demo](#)

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p.minor_xs(1)
```

Its output is as follows –

|   | Item1     | Item2     |
|---|-----------|-----------|
| 0 | -0.128637 | -1.047032 |
| 1 | 0.896681  | -0.557322 |
| 2 | 0.571668  | 0.431953  |
| 3 | -0.144234 | 1.302466  |

**Note** – Observe the changes in the dimensions.