Advantages pandas

Library Highlights

* A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
* Tools for **reading and writing data** between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
* Intelligent **data alignment** and integrated handling of **missing data**: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
* Flexible **reshaping** and pivoting of data sets;
* Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets;
* Columns can be inserted and deleted from data structures for **size mutability**;
* Aggregating or transforming data with a powerful **group by** engine allowing split-apply-combine operations on data sets;
* High performance **merging and joining** of data sets;
* **Hierarchical axis indexing** provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
* **Time series**-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
* Highly **optimized for performance**, with critical code paths written in [Cython](http://www.cython.org/)or C.
* Python with *pandas* is in use in a wide variety of **academic and commercial**domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

Pandas deals with the following three data structures −

* Series
* DataFrame
* Panel

These data structures are built on top of Numpy array, which means they are fast.

## Dimension & Description

The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure.

For example, DataFrame is a container of Series, Panel is a container of DataFrame.

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Dimensions** | **Description** |
| Series | 1 | 1D labeled homogeneous array, sizeimmutable. |
| Data Frames | 2 | General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns. |
| Panel | 3 | General 3D labeled, size-mutable array. |

Building and handling two or more dimensional arrays is a tedious task, burden is placed on the user to consider the orientation of the data set when writing functions. But using Pandas data structures, the mental effort of the user is reduced.

For example, with tabular data DataFrame it is more semantically helpful to think of the **index** therows the rows and the **columns** rather than axis 0 and axis 1.

### Mutability

All Pandas data structures are value mutable can be changed and except

Series all are size mutable. Series is size immutable.

**Note** − DataFrame is widely used and one of the most important data structures. Panel is used much less.

## Series

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, …

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 23 | 56 | 17 | 52 | 61 | 73 | 90 | 26 | 72 |

### Key Points

* Homogeneous data
* Size Immutable
* Values of Data Mutable

## DataFrame

DataFrame is a two-dimensional array with heterogeneous data. For example,

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Age** | **Gender** | **Rating** |
| Steve | 32 | Male | 3.45 |
| Lia | 28 | Female | 4.6 |
| Vin | 45 | Male | 3.9 |
| Katie | 38 | Female | 2.78 |

The table represents the data of a sales team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

## Data Type of Columns

The data types of the four columns are as follows −

|  |  |
| --- | --- |
| **Column** | **Type** |
| Name | String |
| Age | Integer |
| Gender | String |
| Rating | Float |

### Key Points

* Heterogeneous data
* Size Mutable
* Data Mutable

## Panel

Panel is a three-dimensional data structure with heterogeneous data. It is hard to represent the panel in graphical representation. But a panel can be illustrated as a container of DataFrame.

### Key Points

* Heterogeneous data
* Size Mutable
* Data Mutable

**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)

The parameters of the constructor are as follows −

|  |  |
| --- | --- |
| **S.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

## Create an Empty Series

A basic series, which can be created is an Empty Series.

### 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)

## Create a Series from ndarray

If data is an ndarray, then index passed must be of the same length. If no index is passed, then by default index will be **range(n)** where **n** is array length, i.e., [0,1,2,3…. **range(len(array))-1].**

### 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 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

dtype: object

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

## Create a Series from dict

A **dict** can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If **index** is passed, the values in data corresponding to the labels in the index will be pulled out.

### Example 1

#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)

print (s)

Its **output** is as follows −

a 0.0

b 1.0

c 2.0

dtype: float64

**Observe** − Dictionary keys are used to construct index.

### 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).

## Create a Series from Scalar

If data is a scalar value, an index must be provided. The value will be repeated to match the length of **index**

#import the pandas library and aliasing as pd

import pandas as pd

import numpy as np

s = pd.Series(5, index=[0, 1, 2, 3])

print (s)

Its **output** is as follows −

0 5

1 5

2 5

3 5

dtype: int64

## Accessing Data from Series with Position

Data in the series can be accessed similar to that in an **ndarray.**

### Example 1

Retrieve the first element. As we already know, the counting starts from zero for the array, which means the first element is stored at zeroth position and so on.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first element

print (s[0])

Its **output** is as follows −

1

### Example 2

Retrieve the first three elements in the Series. If a : is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with : between them) is used, items between the two indexes (not including the stop index)

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first three element

print (s[:3])

Its **output** is as follows −

a 1

b 2

c 3

dtype: int64

### Example 3

Retrieve the last three elements.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the last three element

print (s[-3:])

Its **output** is as follows −

c 3

d 4

e 5

dtype: int64

## Retrieve Data Using Label (Index)

A Series is like a fixed-size **dict** in that you can get and set values by index label.

### Example 1

Retrieve a single element using index label value.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve a single element

print s['a']

Its **output** is as follows −

1

### Example 2

Retrieve multiple elements using a list of index label values.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements

print (['a','c','d']])

Its **output** is as follows −

a 1

c 3

d 4

dtype: int64

### Example 3

If a label is not contained, an exception is raised.

import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements

print (s['f'])

Its **output** is as follows −

…

KeyError: 'f'

**Data frame**

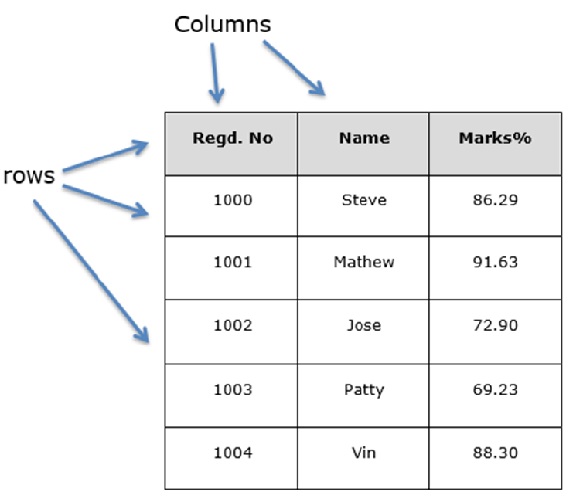
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

### 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.



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 −

|  |  |
| --- | --- |
| **S.No** | **Parameter & Description** |
| 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.arrange(n) if no index is passed. |
| 3 | **columns**  For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed. |
| 4 | **dtype**  Data type of each column. |
| 4 | **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

**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

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.

## 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

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

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

## Column Deletion

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

### 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

## 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.

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.

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.

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.

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.

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.

Ex1:

dict = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],

"capital": ["Brasilia", "Moscow", "New Dehli", "Beijing", "Pretoria"],

"area": [8.516, 17.10, 3.286, 9.597, 1.221],

"population": [200.4, 143.5, 1252, 1357, 52.98] }

import pandas as pd

brics = pd.DataFrame(dict)

print(brics)

o/p

area capital country population

0 8.516 Brasilia Brazil 200.40

1 17.100 Moscow Russia 143.50

2 3.286 New Dehli India 1252.00

3 9.597 Beijing China 1357.00

4 1.221 Pretoria South Africa 52.98

Panels:

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, **false** |

## Create Panel

A Panel can be created using multiple ways like −

* From ndarrays
* From dict of DataFrames

### From 3D ndarray

# 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

#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 −

<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

### Create an Empty Panel

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

#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

# 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)**.

# 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).**

# 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.

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

|  |  |  |
| --- | --- | --- |
| **S.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

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.

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.

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

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.

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.

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.

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.

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.

|  |  |  |
| --- | --- | --- |
| **S.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

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

# Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.**

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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.

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),

'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