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** (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 −

|  |  |
| --- | --- |
| **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

Create an Empty Series

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

Example

[Live Demo](http://tpcg.io/gtANpp)

#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

[Live Demo](http://tpcg.io/FLrJSx)

#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

[Live Demo](http://tpcg.io/gAGm9R)

#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

[Live Demo](http://tpcg.io/AES3sj)

#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

[Live Demo](http://tpcg.io/UYPfMz)

#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**

[Live Demo](http://tpcg.io/DAa5Bv)

#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.

[Live Demo](http://tpcg.io/0XykSB)

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)

[Live Demo](http://tpcg.io/RuNATH)

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.

[Live Demo](http://tpcg.io/wKa6WD)

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.

[Live Demo](http://tpcg.io/N7jQ50)

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.

[Live Demo](http://tpcg.io/iVbMDT)

import pandas as pd

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

#retrieve multiple elements

print s[['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']

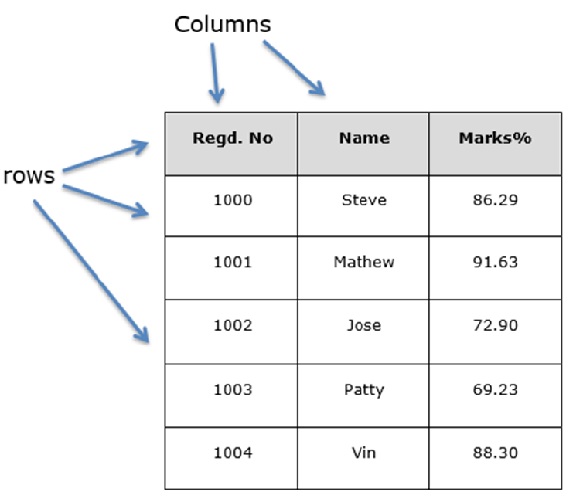
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

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 −

|  |  |
| --- | --- |
| **Sr.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.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

[Live Demo](http://tpcg.io/pYtdhG)

#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](http://tpcg.io/cqMm2H)

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](http://tpcg.io/4O3Ab7)

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](http://tpcg.io/9bLwXj)

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](http://tpcg.io/sWDCEY)

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](http://tpcg.io/juJRE9)

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](http://tpcg.io/PynkrD)

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](http://tpcg.io/qRx0ic)

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](http://tpcg.io/Biq25j)

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](http://tpcg.io/yn5FTV)

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](http://tpcg.io/Rq2wBY)

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](http://tpcg.io/dPCFDX)

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](http://tpcg.io/tHyPQW)

# 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](http://tpcg.io/FYCqj5)

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](http://tpcg.io/lA7zea)

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](http://tpcg.io/iNZZv6)

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](http://tpcg.io/8zJwv2)

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](http://tpcg.io/Eq7pwq)

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

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

[Live Demo](http://tpcg.io/joQRq9)

# 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](http://tpcg.io/2lC1h3)

#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](http://tpcg.io/6IoJuL)

#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](http://tpcg.io/c7pygU)

# 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](http://tpcg.io/E7C5s4)

# 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](http://tpcg.io/wj53EO)

# 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

Functions using data sets:

|  |
| --- |
| import pandas as pd |
|  |  |
|  | # data can be downloaded from the url: https://www.kaggle.com/vikrishnan/boston-house-prices |
|  | df = pd.read\_csv('./06\_input\_data.csv') |
|  |  |
|  | # Understanding data |
|  | print (df.shape) |
|  | print (df.columns) |
|  | print(df.head(5)) |
|  | print(df.info()) |
|  | print(df.describe()) |
|  | print(df.groupby('LotShape').size()) |
|  |  |
|  | # Dropping null value columns which cross the threshold |
|  | a = df.isnull().sum() |
|  | print (a) |
|  | b = a[a>(0.05\*len(a))] |
|  | print (b) |
|  | df = df.drop(b.index, axis=1) |
|  | print (df.shape) |
|  |  |
|  | # Replacing null value columns (text) with most used value |
|  | a1 = df.select\_dtypes(include=['object']).isnull().sum() |
|  | print (a1) |
|  | print (a1.index) |
|  | for i in a1.index: |
|  | b1 = df[i].value\_counts().index.tolist() |
|  | print (b1) |
|  | df[i] = df[i].fillna(b1[0]) |
|  |  |
|  | # Replacing null value columns (int, float) with most used value |
|  | a2 = df.select\_dtypes(include=['integer','float']).isnull().sum() |
|  | print (a2) |
|  | b2 = a2[a2!=0].index |
|  | print (b2) |
|  | df = df.fillna(df[b2].mode().to\_dict(orient='records')[0]) |
|  |  |
|  | # Creating new columns from existing columns |
|  | print (df.shape) |
|  | a3 = df['YrSold'] - df['YearBuilt'] |
|  | b3 = df['YrSold'] - df['YearRemodAdd'] |
|  | df['Years Before Sale'] = a3 |
|  | df['Years Since Remod'] = b3 |
|  | print (df.shape) |
|  |  |
|  | # Dropping unwanted columns |
|  | df = df.drop(["Id", "MoSold", "SaleCondition", "SaleType", "YearBuilt", "YearRemodAdd"], axis=1) |
|  | print (df.shape) |
|  |  |
|  | # Dropping columns which has correlation with target less than threshold |
|  | target='SalePrice' |
|  | x = df.select\_dtypes(include=['integer','float']).corr()[target].abs() |
|  | print (x) |
|  | df=df.drop(x[x<0.4].index, axis=1) |
|  | print (df.shape) |
|  |  |
|  | # Checking for the necessary features after dropping some columns |
|  | l1 = ["PID","MS SubClass","MS Zoning","Street","Alley","Land Contour","Lot Config","Neighborhood","Condition 1","Condition 2","Bldg Type","House Style","Roof Style","Roof Matl","Exterior 1st","Exterior 2nd","Mas Vnr Type","Foundation","Heating","Central Air","Garage Type","Misc Feature","Sale Type","Sale Condition"] |
|  | l2 = [] |
|  | for i in l1: |
|  | if i in df.columns: |
|  | l2.append(i) |
|  |  |
|  | # Getting rid of nominal columns with too many unique values |
|  | for i in l2: |
|  | len(df[i].unique())>10 |
|  | df=df.drop(i, axis=1) |
|  | print (df.columns) |
|  |  |
|  | df.to\_csv('06\_output\_data.csv',index=False) |