Python Pandas - Environment Setup

Standard Python distribution doesn't come bundled with Pandas module. A lightweight alternative is to install NumPy using popular Python package installer, **pip.**

pip install pandas

If you install Anaconda Python package, Pandas will be installed by default with the following −

## Windows

* **Anaconda** (from [https://www.continuum.io](https://www.continuum.io/)) is a free Python distribution for SciPy stack. It is also available for Linux and Mac.
* **Canopy** ([https://www.enthought.com/products/canopy/](https://www.enthought.com/products/canopy)) is available as free as well as commercial distribution with full SciPy stack for Windows, Linux and Mac.
* **Python** (x,y) is a free Python distribution with SciPy stack and Spyder IDE for Windows OS. (Downloadable from <http://python-xy.github.io/>)

## Linux

Package managers of respective Linux distributions are used to install one or more packages in SciPy stack.

**For Ubuntu Users**

sudo apt-get install python-numpy python-scipy python-matplotlibipythonipythonnotebook

python-pandas python-sympy python-nose

**For Fedora Users**

sudo yum install numpyscipy python-matplotlibipython python-pandas sympy

python-nose atlas-devel

# Data Structures

andas deals with the following three data structures −

* Series : one dimension
* DataFrame: table
* 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

# Pandas - Series

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 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']

Its **output** is as follows −

…

KeyError: 'f'

# Pandas - DataFrame

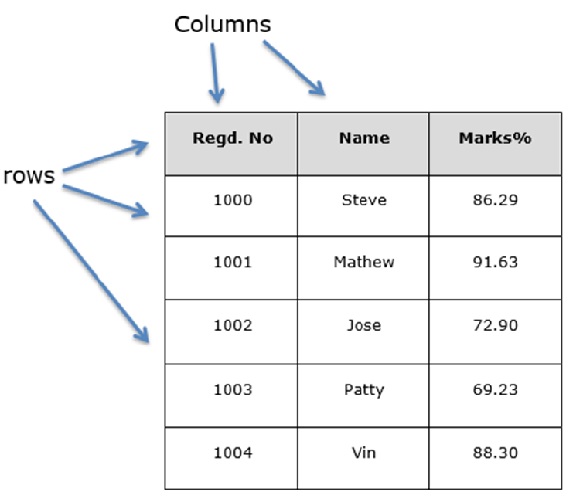
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

### 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.



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.

# Pandas - Aggregations

## Applying Aggregations on DataFrame

Let us create a DataFrame and apply aggregations on it.

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(10, 4),

index = pd.date\_range('1/1/2000', periods=10),

columns = ['A', 'B', 'C', 'D'])

print df

r = df.rolling(window=3,min\_periods=1)

print r

Its **output** is as follows −

A B C D

2000-01-01 1.088512 -0.650942 -2.547450 -0.566858

2000-01-02 0.790670 -0.387854 -0.668132 0.267283

2000-01-03 -0.575523 -0.965025 0.060427 -2.179780

2000-01-04 1.669653 1.211759 -0.254695 1.429166

2000-01-05 0.100568 -0.236184 0.491646 -0.466081

2000-01-06 0.155172 0.992975 -1.205134 0.320958

2000-01-07 0.309468 -0.724053 -1.412446 0.627919

2000-01-08 0.099489 -1.028040 0.163206 -1.274331

2000-01-09 1.639500 -0.068443 0.714008 -0.565969

2000-01-10 0.326761 1.479841 0.664282 -1.361169

Rolling [window=3,min\_periods=1,center=False,axis=0]

We can aggregate by passing a function to the entire DataFrame, or select a column via the standard **get item** method.

### Apply Aggregation on a Whole Dataframe

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(10, 4),

index = pd.date\_range('1/1/2000', periods=10),

columns = ['A', 'B', 'C', 'D'])

print df

r = df.rolling(window=3,min\_periods=1)

print r.aggregate(np.sum)

Its **output** is as follows −

A B C D

2000-01-01 1.088512 -0.650942 -2.547450 -0.566858

2000-01-02 1.879182 -1.038796 -3.215581 -0.299575

2000-01-03 1.303660 -2.003821 -3.155154 -2.479355

2000-01-04 1.884801 -0.141119 -0.862400 -0.483331

2000-01-05 1.194699 0.010551 0.297378 -1.216695

2000-01-06 1.925393 1.968551 -0.968183 1.284044

2000-01-07 0.565208 0.032738 -2.125934 0.482797

2000-01-08 0.564129 -0.759118 -2.454374 -0.325454

2000-01-09 2.048458 -1.820537 -0.535232 -1.212381

2000-01-10 2.065750 0.383357 1.541496 -3.201469

A B C D

2000-01-01 1.088512 -0.650942 -2.547450 -0.566858

2000-01-02 1.879182 -1.038796 -3.215581 -0.299575

2000-01-03 1.303660 -2.003821 -3.155154 -2.479355

2000-01-04 1.884801 -0.141119 -0.862400 -0.483331

2000-01-05 1.194699 0.010551 0.297378 -1.216695

2000-01-06 1.925393 1.968551 -0.968183 1.284044

2000-01-07 0.565208 0.032738 -2.125934 0.482797

2000-01-08 0.564129 -0.759118 -2.454374 -0.325454

2000-01-09 2.048458 -1.820537 -0.535232 -1.212381

2000-01-10 2.065750 0.383357 1.541496 -3.201469

### Apply Aggregation on a Single Column of a Dataframe

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(10, 4),

index = pd.date\_range('1/1/2000', periods=10),

columns = ['A', 'B', 'C', 'D'])

print df

r = df.rolling(window=3,min\_periods=1)

print r['A'].aggregate(np.sum)

Its **output** is as follows −

A B C D

2000-01-01 1.088512 -0.650942 -2.547450 -0.566858

2000-01-02 1.879182 -1.038796 -3.215581 -0.299575

2000-01-03 1.303660 -2.003821 -3.155154 -2.479355

2000-01-04 1.884801 -0.141119 -0.862400 -0.483331

2000-01-05 1.194699 0.010551 0.297378 -1.216695

2000-01-06 1.925393 1.968551 -0.968183 1.284044

2000-01-07 0.565208 0.032738 -2.125934 0.482797

2000-01-08 0.564129 -0.759118 -2.454374 -0.325454

2000-01-09 2.048458 -1.820537 -0.535232 -1.212381

2000-01-10 2.065750 0.383357 1.541496 -3.201469

2000-01-01 1.088512

2000-01-02 1.879182

2000-01-03 1.303660

2000-01-04 1.884801

2000-01-05 1.194699

2000-01-06 1.925393

2000-01-07 0.565208

2000-01-08 0.564129

2000-01-09 2.048458

2000-01-10 2.065750

Freq: D, Name: A, dtype: float64

### Apply Aggregation on Multiple Columns of a DataFrame

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(10, 4),

index = pd.date\_range('1/1/2000', periods=10),

columns = ['A', 'B', 'C', 'D'])

print df

r = df.rolling(window=3,min\_periods=1)

print r[['A','B']].aggregate(np.sum)

Its **output** is as follows −

A B C D

2000-01-01 1.088512 -0.650942 -2.547450 -0.566858

2000-01-02 1.879182 -1.038796 -3.215581 -0.299575

2000-01-03 1.303660 -2.003821 -3.155154 -2.479355

2000-01-04 1.884801 -0.141119 -0.862400 -0.483331

2000-01-05 1.194699 0.010551 0.297378 -1.216695

2000-01-06 1.925393 1.968551 -0.968183 1.284044

2000-01-07 0.565208 0.032738 -2.125934 0.482797

2000-01-08 0.564129 -0.759118 -2.454374 -0.325454

2000-01-09 2.048458 -1.820537 -0.535232 -1.212381

2000-01-10 2.065750 0.383357 1.541496 -3.201469

A B

2000-01-01 1.088512 -0.650942

2000-01-02 1.879182 -1.038796

2000-01-03 1.303660 -2.003821

2000-01-04 1.884801 -0.141119

2000-01-05 1.194699 0.010551

2000-01-06 1.925393 1.968551

2000-01-07 0.565208 0.032738

2000-01-08 0.564129 -0.759118

2000-01-09 2.048458 -1.820537

2000-01-10 2.065750 0.383357

### Apply Multiple Functions on a Single Column of a DataFrame

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(10, 4),

index = pd.date\_range('1/1/2000', periods=10),

columns = ['A', 'B', 'C', 'D'])

print df

r = df.rolling(window=3,min\_periods=1)

print r['A'].aggregate([np.sum,np.mean])

Its **output** is as follows −

A B C D

2000-01-01 1.088512 -0.650942 -2.547450 -0.566858

2000-01-02 1.879182 -1.038796 -3.215581 -0.299575

2000-01-03 1.303660 -2.003821 -3.155154 -2.479355

2000-01-04 1.884801 -0.141119 -0.862400 -0.483331

2000-01-05 1.194699 0.010551 0.297378 -1.216695

2000-01-06 1.925393 1.968551 -0.968183 1.284044

2000-01-07 0.565208 0.032738 -2.125934 0.482797

2000-01-08 0.564129 -0.759118 -2.454374 -0.325454

2000-01-09 2.048458 -1.820537 -0.535232 -1.212381

2000-01-10 2.065750 0.383357 1.541496 -3.201469

sum mean

2000-01-01 1.088512 1.088512

2000-01-02 1.879182 0.939591

2000-01-03 1.303660 0.434553

2000-01-04 1.884801 0.628267

2000-01-05 1.194699 0.398233

2000-01-06 1.925393 0.641798

2000-01-07 0.565208 0.188403

2000-01-08 0.564129 0.188043

2000-01-09 2.048458 0.682819

2000-01-10 2.065750 0.688583

### Apply Multiple Functions on Multiple Columns of a DataFrame

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(10, 4),

index = pd.date\_range('1/1/2000', periods=10),

columns = ['A', 'B', 'C', 'D'])

print df

r = df.rolling(window=3,min\_periods=1)

print r[['A','B']].aggregate([np.sum,np.mean])

Its **output** is as follows −

A B C D

2000-01-01 1.088512 -0.650942 -2.547450 -0.566858

2000-01-02 1.879182 -1.038796 -3.215581 -0.299575

2000-01-03 1.303660 -2.003821 -3.155154 -2.479355

2000-01-04 1.884801 -0.141119 -0.862400 -0.483331

2000-01-05 1.194699 0.010551 0.297378 -1.216695

2000-01-06 1.925393 1.968551 -0.968183 1.284044

2000-01-07 0.565208 0.032738 -2.125934 0.482797

2000-01-08 0.564129 -0.759118 -2.454374 -0.325454

2000-01-09 2.048458 -1.820537 -0.535232 -1.212381

2000-01-10 2.065750 0.383357 1.541496 -3.201469

A B

sum mean sum mean

2000-01-01 1.088512 1.088512 -0.650942 -0.650942

2000-01-02 1.879182 0.939591 -1.038796 -0.519398

2000-01-03 1.303660 0.434553 -2.003821 -0.667940

2000-01-04 1.884801 0.628267 -0.141119 -0.047040

2000-01-05 1.194699 0.398233 0.010551 0.003517

2000-01-06 1.925393 0.641798 1.968551 0.656184

2000-01-07 0.565208 0.188403 0.032738 0.010913

2000-01-08 0.564129 0.188043 -0.759118 -0.253039

2000-01-09 2.048458 0.682819 -1.820537 -0.606846

2000-01-10 2.065750 0.688583 0.383357 0.127786

### Apply Different Functions to Different Columns of a Dataframe

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(3, 4),

index = pd.date\_range('1/1/2000', periods=3),

columns = ['A', 'B', 'C', 'D'])

print df

r = df.rolling(window=3,min\_periods=1)

print r.aggregate({'A' : np.sum,'B' : np.mean})

Its **output** is as follows −

A B C D

2000-01-01 -1.575749 -1.018105 0.317797 0.545081

2000-01-02 -0.164917 -1.361068 0.258240 1.113091

2000-01-03 1.258111 1.037941 -0.047487 0.867371

A B

2000-01-01 -1.575749 -1.018105

2000-01-02 -1.740666 -1.189587

2000-01-03 -0.482555 -0.447078

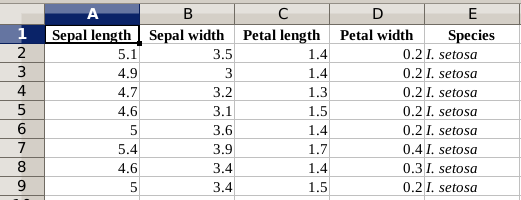
**Read Excel column names**  
We import the pandas module, including ExcelFile. The method read\_excel() reads the data into a Pandas Data Frame, where the first parameter is the filename and the second parameter is the sheet.  
  
The list of columns will be called df.columns.

|  |
| --- |
| **import** pandas **as** pd  **from** pandas **import** ExcelWriter  **from** pandas **import** ExcelFile    df = pd.read\_excel('File.xlsx', sheetname='Sheet1')    **print**("Column headings:")  **print**(df.columns) |

Using the data frame, we can get all the rows below an entire column as a list. To get such a list, simply use the column header

|  |
| --- |
| **print**(df['Sepal width']) |

**Read Excel data**  
We start with a simple Excel file, a subset of the Iris dataset.

[](https://pythonspot-9329.kxcdn.com/wp-content/uploads/2017/03/excel-pandas.png)

To iterate over the list we can use a loop:

|  |
| --- |
| **for** i **in** df.index:  **print**(df['Sepal width'][i]) |

We can save an entire column into a list:

|  |
| --- |
| listSepalWidth = df['Sepal width']  **print**(listSepalWidth[0]) |

We can simply take entire columns from an excel sheet:

|  |
| --- |
| **import** pandas **as** pd  **from** pandas **import** ExcelWriter  **from** pandas **import** ExcelFile    df = pd.read\_excel('File.xlsx', sheetname='Sheet1')    sepalWidth = df['Sepal width']  sepalLength = df['Sepal length']  petalLength = df['Petal length'] |