

[Next >](#)

ial  
Post  
s  
Questio  
ns  
es

Filed Under: [Pandas](#)

# Module Tutorial

1 Comments

[Home](#) » [Python](#) » Python Pandas Module Tutorial

## Python Pandas Module

Pandas is an open source library in Python. It provides ready to use high-performance data structures and data analysis tools.

Pandas module runs on top of [NumPy](#) and it is popularly used for data science and data analytics.

NumPy is a low-level data structure that supports multi-dimensional arrays and a wide range of mathematical array operations. Pandas has a higher-level interface. It also provides streamlined alignment of tabular data and powerful time series functionality.

a  
ows  
te

[Next >](#)

ame.  
ent,

grouping, merging,  
concatenating data, etc.

## Installing and Getting Started with Pandas

You need to have Python 2.7 and above to install Pandas module.

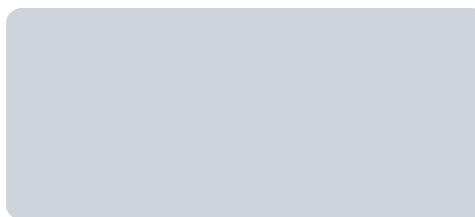
If you are using **conda**, then you can install it using below command.

IS



pandas

Next >

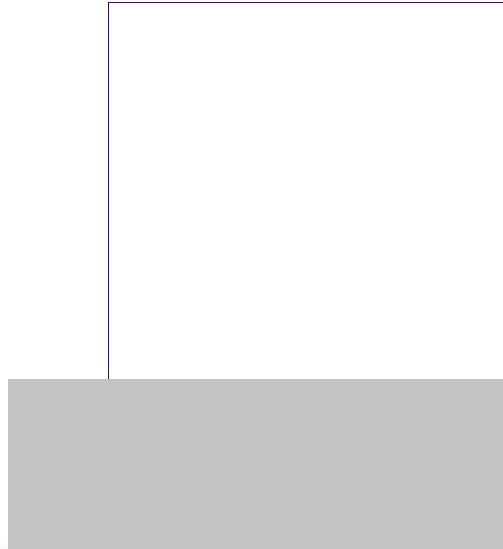


Python Install Pandas Module

To import Pandas and NumPy in your Python script, add the below piece of code:

```
import pandas as pd  
import numpy as np
```

As Pandas is dependent on the NumPy library, we need to import this dependency.



[Next >](#)

ule,

**Series:** It is a 1-D size-immutable array like structure having homogeneous data.

**DataFrames:** It is a 2-D size-mutable tabular structure with heterogeneously typed columns.

**Panel:** It is a 3-D, size-mutable array.

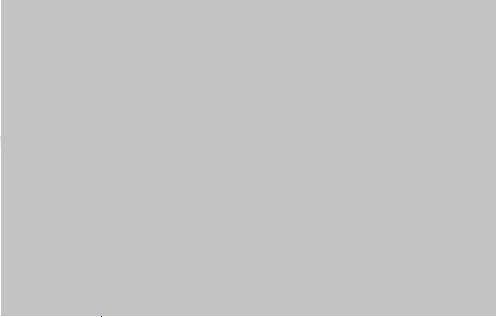
## Pandas DataFrame

DataFrame is the most important and widely used data structure and is a standard way to store data.

DataFrame has data aligned in rows and columns like the SQL table or a spreadsheet database.

We can either hard code data into a DataFrame or import a CSV file, tsv file, Excel file, SQL table, etc.

We can use the below

[Next >](#)

```
pandas.DataFrame(dat  
a, index, columns,  
dtype, copy)
```

Below is a short description of the parameters:

**data** – create a DataFrame object from the input data. It can be list, dict, series, Numpy ndarrays or even, any other DataFrame.

**index** – has the row labels

**columns** – used to create column labels

**dtype** – used to specify the data type of each column, optional parameter

**copy** – used for copying data, if any



ate a

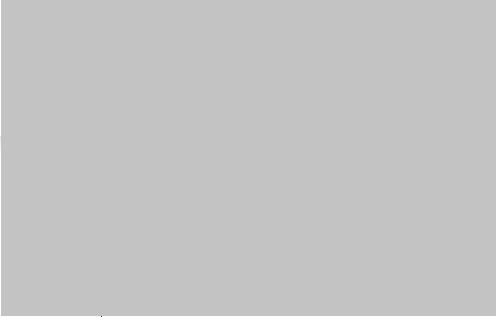
[Next >](#)

ate  
a database from the list of  
dictionaries.

```
import pandas as pd
import numpy as np
df = pd.DataFrame({
    "State": [
        'Andhra Pradesh',
        'Maharashtra',
        'Karnataka',
        'Kerala', 'Tamil
Nadu'],
    "Capital": [
        'Hyderabad',
        'Mumbai',
        'Bengaluru',
        'Trivandrum',
        'Chennai'],
    "Literacy %": [
        89, 77, 82, 97, 85],
    "Avg High
Temp(c)": [33, 30,
29, 31, 32]
})
print(df)
```

## Output:

	State	Capital	Literacy %	Avg High Temp(c)
0	Andhra Pradesh	Hyderabad	89	33
1	Maharashtra	Mumbai	77	30
2	Karnataka	Bengaluru	82	29
3	Kerala	Trivandrum	97	31
4	Tamil Nadu	Chennai	85	32

[Next >](#)

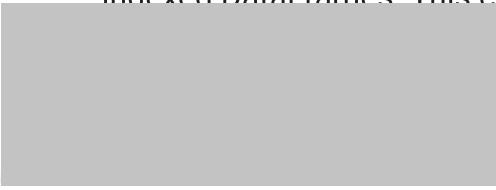
The first step is to create a dictionary. The second step is to pass the dictionary as an argument in the DataFrame() method. The final step is to print the DataFrame.

As you see, the DataFrame can be compared to a table having heterogeneous value. Also, the size of the DataFrame can be modified.

We have supplied the data in the form of the `map` and the keys of the map are considered by Pandas as the row labels.

The index is displayed in the leftmost column and has the row labels. The column header and data are displayed in a tabular fashion.

It is also possible to create indexed DataFrames. This can be



x

[Next >](#)

## Importing data from CSV to DataFrame

We can also create a DataFrame by importing a CSV file. A CSV file is a text file with one record of data per line. The values within the record are separated using the “comma” character.

Pandas provides a useful method, named `read_csv()` to read the contents of the CSV file into a DataFrame.

For example, we can create a file named ‘cities.csv’ containing details of Indian cities. The CSV file is stored in the same directory that contains Python scripts. This file can be imported using:

.  
.

A large gray rectangular placeholder box occupies the top-left portion of the slide.

Next >



So, we can use any convenient method to load the data. In this tutorial, we are hard-coding the data of the DataFrame.

---

## Inspecting data in DataFrame

Running the DataFrame using its name displays the entire table. In real-time, the datasets to analyze will have thousands of rows. For analyzing data, we need to inspect data from huge volumes of datasets. Pandas provide many useful functions to inspect only the data we need. We can use `df.head(n)` to get the first n rows or `df.tail(n)` to print the last n rows. For example, the below code prints the first 2 rows and last 1 row from the DataFrame.

A large gray rectangular placeholder box occupies the bottom-left portion of the slide.

[Next >](#)

```
print(df.head(2))
```

**Output:**

	State	Capital	Literacy %	Avg High Temp(c)
0	Andhra Pradesh	Hyderabad	89	33
1	Maharashtra	Mumbai	77	30

```
print(df.tail(1))
```

**Output:**

	State	Capital	Literacy %	Avg High Temp(c)
4	Tamil Nadu	Chennai	85	32

Similarly, `print(df.dtypes)` prints the data types.

[Next >](#)

### Output:

```
State          object
Capital        object
Literacy %     int64
Avg High Temp(c)    int64
dtype: object
```

`print(df.index)` prints  
index.

### Output:

```
| RangeIndex(start=0, stop=5, step=1)
```

`print(df.columns)` prints  
the.

[Next >](#)

days

```
[['Andhra Pradesh' 'Hyderabad' 89 33]
 ['Maharashtra' 'Mumbai' 77 30]
 ['Karnataka' 'Bengaluru' 82 29]
 ['Kerala' 'Trivandrum' 97 31]
 ['Tamil Nadu' 'Chennai' 85 32]]
```

## 1. Getting Statistical summary of records

We can get statistical summary (count, mean, standard deviation, min, max etc.) of the data using `df.describe()` function.

Now, let's use this function, to display the statistical summary of "Literacy %" column. To do this,

of

[Next >](#)

```
count      5.000000
mean     86.000000
std      7.549834
min     77.000000
25%    82.000000
50%    85.000000
75%    89.000000
max     97.000000
Name: Literacy %, dtype: float64
```

The `df.describe()` function displays the statistical summary, along with the data type.

## 2. Sorting records

We can sort records by any column using `df.sort_values()` function. For example, let's sort the "Literacy %" column in descending order.

[Next >](#)

### Output.

	State	Capital	Literacy %	Avg	High Temp(c)
3	Kerala	Trivandrum	97	31	
0	Andhra Pradesh	Hyderabad	89	33	
4	Tamil Nadu	Chennai	85	32	
2	Karnataka	Bengaluru	82	29	
1	Maharashtra	Mumbai	77	30	

## 3. Slicing records

It is possible to extract data of a particular column, by using the column name. For example, to extract the 'Capital' column, we use:

```
df['Capital']
```

or

[Next >](#)

It is also possible to slice multiple columns. This is done by enclosing multiple column names enclosed in 2 square brackets, with the column names separated using commas. The following code slices the 'State' and 'Capital' columns of the DataFrame.

```
print(df[['State',  
          'Capital']])
```

### Output:

	State	Capital
0	Andhra Pradesh	Hyderabad
1	Maharashtra	Mumbai
2	Karnataka	Bengaluru
3	Kerala	Trivandrum
4	Tamil Nadu	Chennai

[Next >](#)

## Output:

	State	Capital	Literacy %	Avg	High Temp(c)
0	Andhra Pradesh	Hyderabad	89	33	
1	Maharashtra	Mumbai	77	30	
2	Karnataka	Bengaluru	82	29	

An interesting feature of Pandas library is to select data based on its row and column labels using `iloc[0]` function.

Many times, we may need only few columns to analyze. We can also can select by index using `loc['index_one'])`.

For example, to select the second row. we can use `df.iloc[1,:]`

[Next >](#)

displays numerical output.

## 4. Filtering data

It is also possible to filter on column values. For example, below code filters the columns having Literacy% above 90%.

```
print(df[df['Literacy %']>90])
```

Any comparison operator can be used to filter, based on a condition.

**Output:**

[Next >](#)

```
[  
    .isin(['Karnataka',  
           'Tamil Nadu']))]
```

### Output:

	State	Capital	Literacy %	Avg High Temp(c)
2	Karnataka	Bengaluru	82	29
4	Tamil Nadu	Chennai	85	32

## 5. Rename column

It is possible to use the `df.rename()` function to rename a column. The function takes the old column name and new column name as arguments. For example, let's rename the column 'Literacy %' to 'Literacy percentage'.



The argument `inplace=True` makes the changes to the DataFrame.

### Output:

	State	Capital	Literacy percentage	Avg High Temp(c)
0	Andhra Pradesh	Hyderabad	89	33
1	Maharashtra	Mumbai	77	30
2	Karnataka	Bengaluru	82	29
3	Kerala	Trivandrum	97	31
4	Tamil Nadu	Chennai	85	32

## 6. Data Wrangling

Data Science involves the processing of data so that the data can work well with the data algorithms. Data Wrangling is the process of processing data, like merging, grouping and concatenating.

The Pandas library provides useful functions like `merge()` , `groupby()` and `concat()` to support Data Wrangling tasks.

Let's create 2 DataFrames and show the Data Wrangling

etter.

[Next >](#)

```
,
```

```
        'Employee_name':
```

```
        ['Akshar', 'Jones',
```

```
        'Kate', 'Mike',
```

```
        'Tina']
```

```
}
```

```
df1 =
```

```
pd.DataFrame(d,
```

```
columns=
```

```
['Employee_id',
```

```
'Employee_name'])
```

```
print(df1)
```

## Output:

	Employee_id	Employee_name
0	1	Akshar
1	2	Jones
2	3	Kate
3	4	Mike
4	5	Tina

Let's create the second  
DataFrame using the below code:

```
import pandas as pd
```

```
data = {
```

```
        'Employee_id':
```

```
        ['4', '5', '6', '7',
```

```
        '8'],
```

```
        'Employee_name':
```

```
        ['Meera', 'Tia',
```

```
        'Varsha',
```

```
        'Williams', 'Ziva']
```

```
}
```

[Next >](#)

```
Employee_id Employee_name
0          4      Meera
1          5       Tia
2          6     Varsha
3          7  Williams
4          8      Ziva
```

## a. Merging

Now, let's merge the 2 DataFrames we created, along the values of 'Employee\_id' using the `merge()` function:

```
print(pd.merge(df1,
               df2,
               on='Employee_id'))
```

### Output:

Next >

## b. Grouping

Grouping is a process of collecting data into different categories. For example, in the below example, the “Employee\_Name” field has the name “Meera” two times. So, let’s group it by “Employee\_name” column.

```
import pandas as pd
import numpy as np

data = {
    'Employee_id': [
        '4', '5', '6', '7',
        '8'],
    'Employee_name': [
        'Meera', 'Meera',
        'Varsha',
        'Williams', 'Ziva']
}
df2 =
pd.DataFrame(data)

group =
df2.groupby('Employee_name')
print(group.get_group(
    'Meera'))
```



aving  
he

[Next >](#)

1      5      Meera

### c. Concatenating

Concatenating data involves to add one set of data to other. Pandas provides a function named `concat()` to concatenate DataFrames. For example, let's concatenate the DataFrames `df1` and `df2`, using :

```
print(pd.concat([df1  
, df2]))
```

#### Output:

	Employee_id	Employee_name
0	1	Akshar
1	2	Jones
2	3	Kate
3	4	Mike
4	5	Tina
0	4	Meera
1	5	Tia
2	6	Varsha
3	7	Williams
4	8	Ziva

[Next >](#)

To create a Series, we can use the `pd.Series()` method and pass an array to it. Let's create a simple Series as follows:

```
series_sample =  
pd.Series([100, 200,  
300, 400])  
print(series_sample)
```

### Output:

```
0    100  
1    200  
2    300  
3    400  
dtype: int64
```

We have created a Series. You can see that 2 columns are displayed. The first column contains the index values starting from 0. The second column contains the elements passed as series.

It is possible to create a DataFrame by passing a dictionary of `Series`. Let's create a DataFrame that is formed by

yes

[Next >](#)

```
        ),
'Position' :
pd.Series([1, 2, 3,
4], index=['Sachin',
'Kohli', 'Raina',
'Dravid'])}
df = pd.DataFrame(d)
print(df)
```

### Sample output

	Matches played	Position
Dravid	NaN	4
Kohli	300.0	2
Raina	200.0	3
Sachin	400.0	1

For series one, as we have not specified label 'd', NaN is returned.

## Column Selection, Addition, Deletion

It is possible to select a specific column from the DataFrame. For example, to display only the first column, we can re-write the above code as:

```
) ,
```

[Next >](#)

```
,
```

```
)
```

```
print(atl['Matches
```

```
played'])
```

The above code prints only the “Matches played” column of the DataFrame.

## Output

```
Dravid      NaN
Kohli     300.0
Raina     200.0
Sachin    400.0
Name: Matches played, dtype: float64
```

It is also possible to add columns to an existing DataFrame. For example, the below code adds a new column named “Runrate” to the above DataFrame.

```
d = {'Matches
played' :
pd.Series([400, 300,
200], index=
['Sachin', 'Kohli',
'Raina']),
'Position' :
pd.Series([1, 2, 3,
4], index=['Sachin',
'Kohli', 'Raina',
'Dravid'])}
```

```
)
```

```
er
```

[Next >](#)

	Matches played	Position	Runrate
Dravid	NaN	4	50
Kohli	300.0	2	70
Raina	200.0	3	60
Sachin	400.0	1	80

We can delete columns using the `delete` and `pop` functions. For example to delete the 'Matches played' column in the above example, we can do it by either of the below 2 ways:

```
del df['Matches  
played']
```

or

```
df.pop('Matches  
played')
```

## Output:

```
Position  
Dravid      4  
Kohli       2  
Raina       3  
Sachin      1
```

## Conclusion

[Next >](#)

nce.  
he  
Python library.

Reference: [Pandas Official Website](#)

Pandas `read_excel()` - Reading Excel File in Python

NEXT



## Comments

Ferdy says:

January 10, 2022 at 9:43 pm

Thanks for your Tutorials about simple basic data analysis it is very help to understand better of AI

Comments are closed.

Most Popular

Favorite Sites

AskPython

[Next >](#)

updated, error-free, and complete. Every month millions of developers like you visit JournalDev to read our tutorials.

JournalDev was founded by Pankaj Kumar in 2010 to share his experience and learnings with the whole world. He loves Open source technologies and writing on JournalDev has become his passion.

Core Java  
Tutorial

LinuxForDevice  
s

Python  
Tutorials

Mkyong

Java Interview  
Questions

VM-Help

Core Java  
Interview  
Questions

WP-Design

Java Design  
Patterns

Spring Tutorial

© 2022 · [Privacy Policy](#) · [Terms and Conditions](#) · [Contact Us](#) · [About](#) · Part of JournalDev IT Services

Private Limited



Next >

