Visualization

Basic Plotting: plot

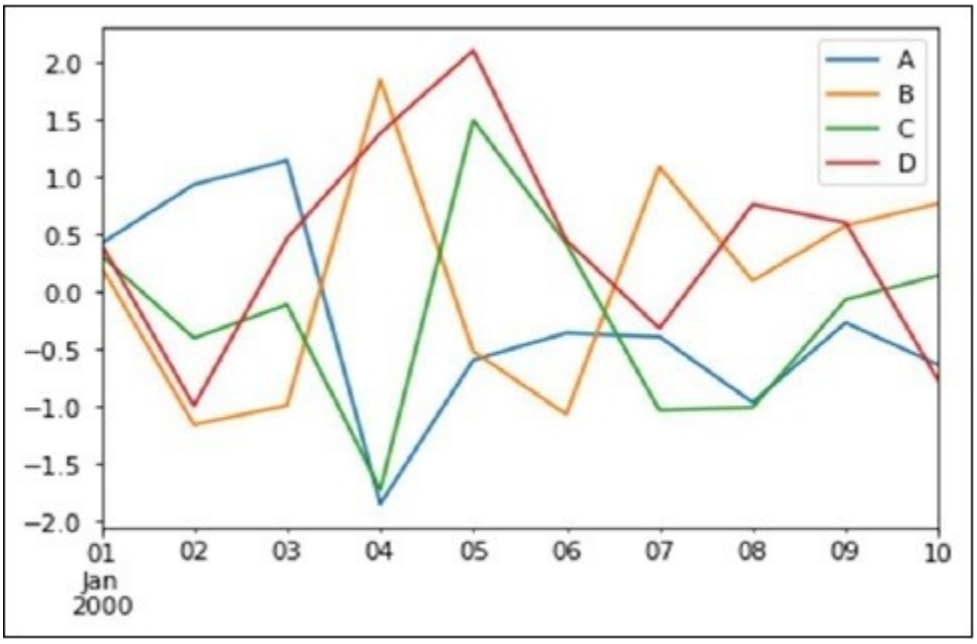
This functionality on Series and DataFrame is just a simple wrapper around the **matplotlib libraries plot()** method.

```
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=list('ABCD'))

df.plot()
```

Its **output** is as follows −



If the index consists of dates, it calls **gct().autofmt_xdate()** to format the x-axis as shown in the above illustration.

We can plot one column versus another using the **x** and **y** keywords.

Plotting methods allow a handful of plot styles other than the default line plot. These methods can be provided as the kind keyword argument to **plot()**. These include −

- bar or barh for bar plots
- hist for histogram
- box for boxplot
- 'area' for area plots

- 'scatter' for scatter plots

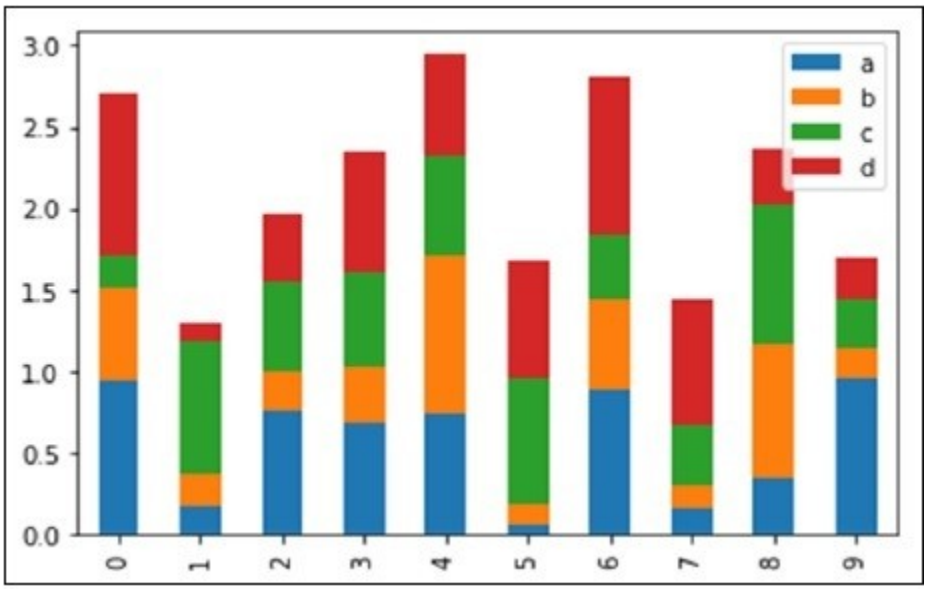
Bar Plot

Let us now see what a Bar Plot is by creating one. A bar plot can be created in the following way −

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d')
df.plot.bar()
```

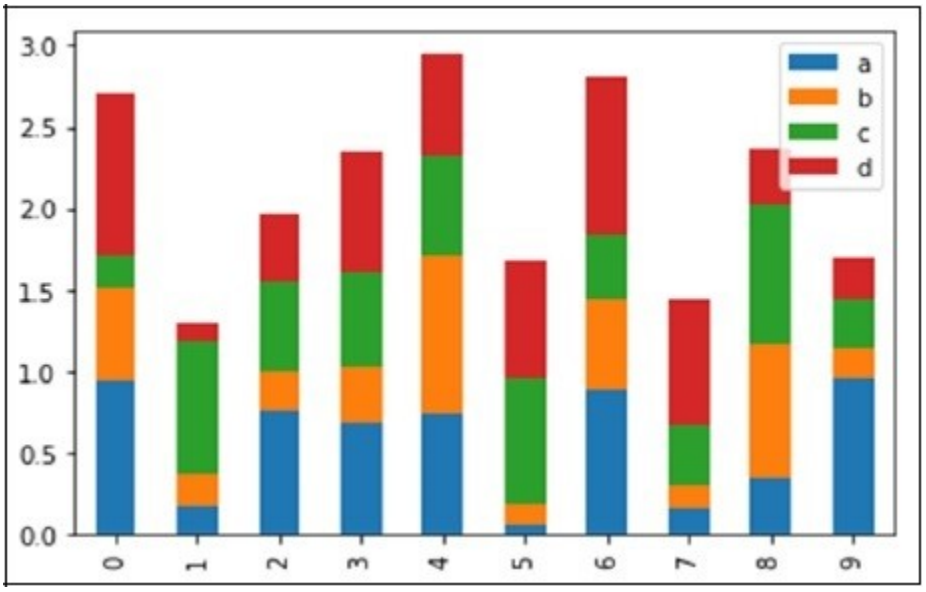
Its **output** is as follows −



To produce a stacked bar plot, **pass stacked=True** −

```
import pandas as pd
df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d')
df.plot.bar(stacked=True)
```

Its **output** is as follows −



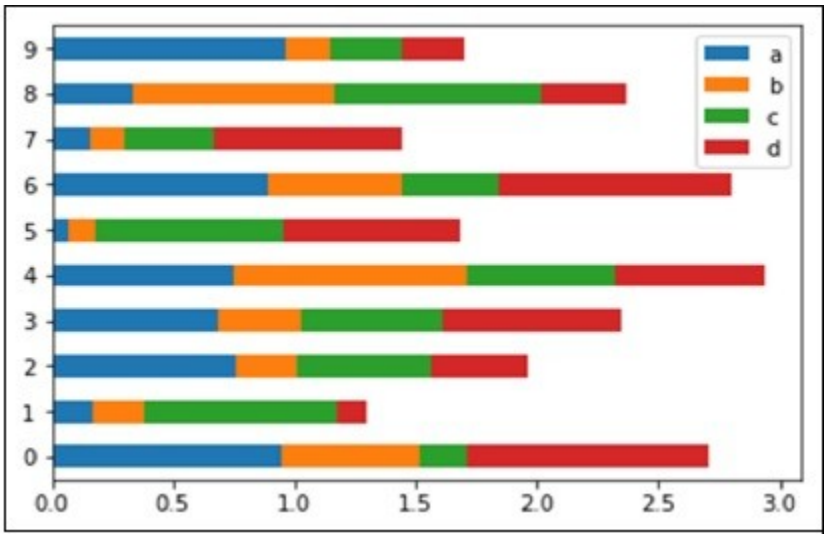
To get horizontal bar plots, use the **barh** method −

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d')

df.plot.barh(stacked=True)
```

Its **output** is as follows −



Histograms

Histograms can be plotted using the **plot.hist()** method. We can specify number of bins.

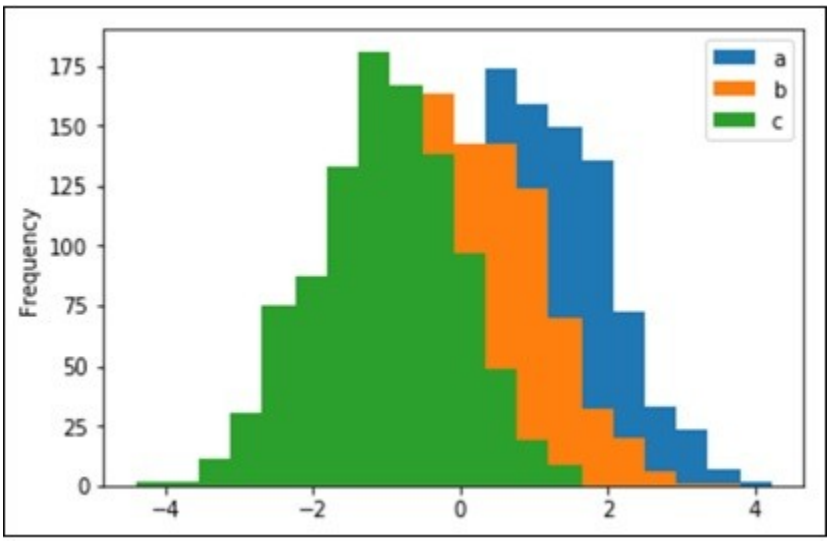
```
import pandas as pd
```

```
import numpy as np

df =
pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000)
,'c':
np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

df.plot.hist(bins=20)
```

Its **output** is as follows −



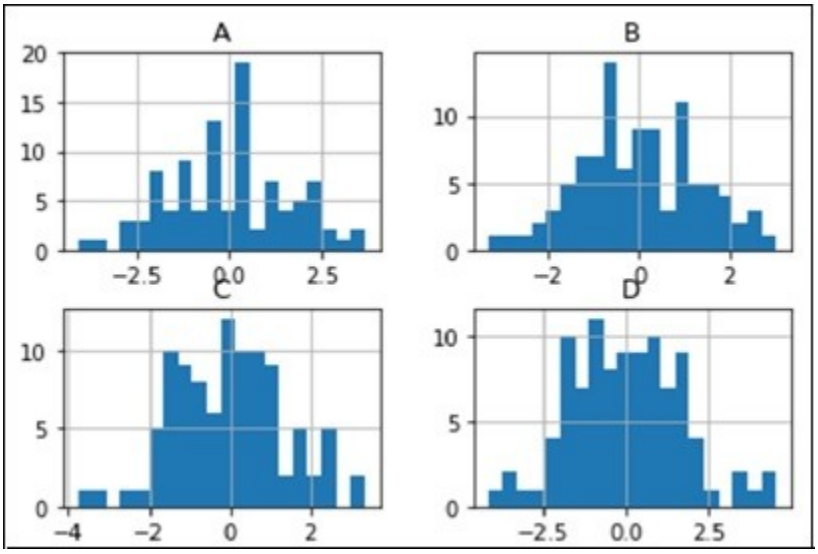
To plot different histograms for each column, use the following code −

```
import pandas as pd
import numpy as np

df=pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(10
00),'c':
np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

df.diff.hist(bins=20)
```

Its **output** is as follows −



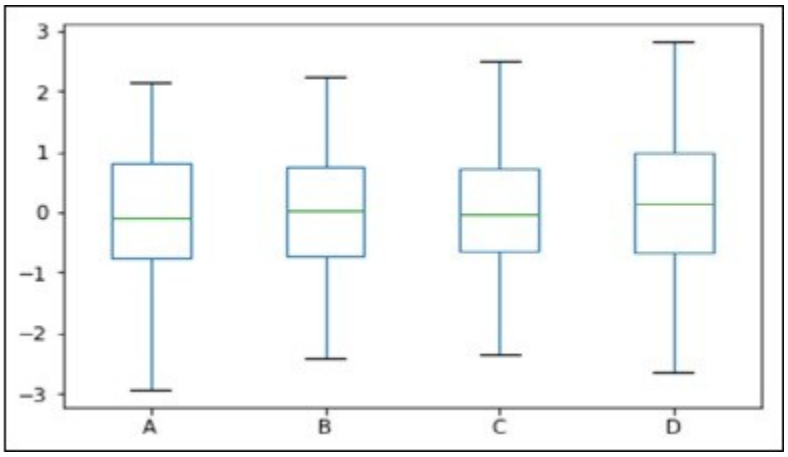
Box Plots

Boxplot can be drawn calling **Series.box.plot()** and **DataFrame.box.plot()**, or **DataFrame.boxplot()** to visualize the distribution of values within each column.

For instance, here is a boxplot representing five trials of 10 observations of a uniform random variable on [0,1).

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C',
'D', 'E'])
df.plot.box()
```

Its **output** is as follows −



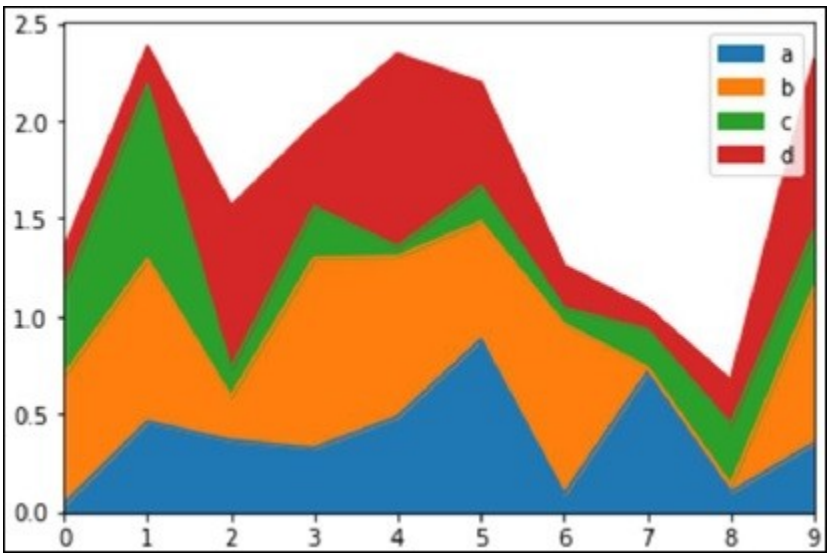
Area Plot

Area plot can be created using the **Series.plot.area()** or the **DataFrame.plot.area()** methods.

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c',
'd'])
df.plot.area()
```

Its **output** is as follows −

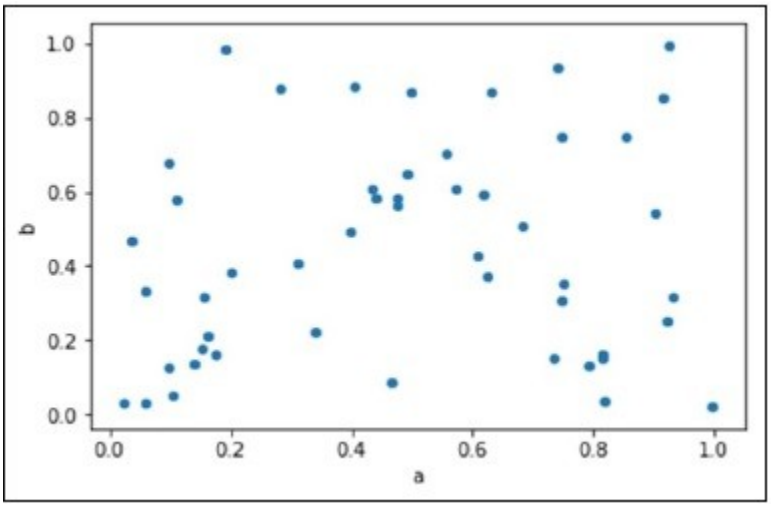


Scatter Plot

Scatter plot can be created using the **DataFrame.plot.scatter()** methods.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c',
'd'])
df.plot.scatter(x='a', y='b')
```

Its **output** is as follows −



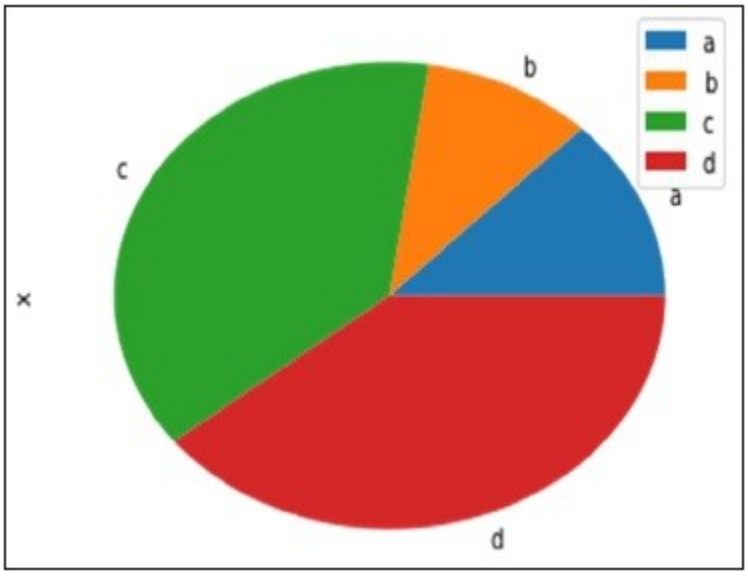
Pie Chart

Pie chart can be created using the **DataFrame.plot.pie()** method.

```
import pandas as pd
import numpy as np

df = pd.DataFrame(3 * np.random.rand(4), index=['a', 'b', 'c',
'd'], columns=['x'])
df.plot.pie(subplots=True)
```

Its **output** is as follows −



IO tools

The **Pandas I/O API** is a set of top level reader functions accessed like **pd.read_csv()** that generally return a Pandas object.

The two workhorse functions for reading text files (or the flat files) are **read_csv()** and **read_table()**. They both use the same parsing code to intelligently convert tabular data into a **DataFrame** object −

```
pandas.read_csv(filepath_or_buffer, sep=',', delimiter=None,
header='infer',
names=None, index_col=None, usecols=None
pandas.read_csv(filepath_or_buffer, sep='\t', delimiter=None,
header='infer',
names=None, index_col=None, usecols=None
```

Here is how the **csv** file data looks like −

```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Save this data as **temp.csv** and conduct operations on it.


```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Save this data as **temp.csv** and conduct operations on it.

read.csv

read.csv reads data from the csv files and creates a DataFrame object.

```
import pandas as pd

df=pd.read_csv("temp.csv")
print df
```

Its **output** is as follows −

```
   S.No     Name   Age       City   Salary
0     1      Tom    28    Toronto    20000
1     2      Lee    32   HongKong     3000
2     3   Steven    43   Bay Area     8300
3     4      Ram    38  Hyderabad     3900
```

custom index

This specifies a column in the csv file to customize the index using **index_col.**

```
import pandas as pd

df=pd.read_csv("temp.csv",index_col=['S.No'])
print df
```

Its **output** is as follows −

```
S.No    Name    Age       City    Salary
1        Tom     28    Toronto     20000
2        Lee     32   HongKong      3000
3     Steven     43   Bay Area      8300
4        Ram     38  Hyderabad      3900
```

Converters

dtype of the columns can be passed as a dict.

```
import pandas as pd

df = pd.read_csv("temp.csv", dtype={'Salary': np.float64})
print df.dtypes
```

Its **output** is as follows −

```
S.No       int64
Name      object
Age        int64
City      object
Salary   float64
dtype: object
```

By default, the **dtype** of the Salary column is **int**, but the result shows it as **float** because we have explicitly casted the type.

Thus, the data looks like float −

```
  S.No    Name    Age       City     Salary
0    1     Tom    28    Toronto    20000.0
1    2     Lee    32   HongKong     3000.0
2    3  Steven    43   Bay Area     8300.0
3    4     Ram    38  Hyderabad     3900.0
```

header_names

Specify the names of the header using the names argument.

```
import pandas as pd
```

```
df=pd.read_csv("temp.csv", names=['a', 'b', 'c','d','e'])
print df
```

Its **output** is as follows −

```
       a        b    c           d        e
0   S.No     Name   Age       City   Salary
1      1      Tom   28     Toronto    20000
2      2      Lee   32    HongKong     3000
3      3   Steven   43    Bay Area     8300
4      4      Ram   38   Hyderabad     3900
```

Observe, the header names are appended with the custom names, but the header in the file has not been eliminated. Now, we use the header argument to remove that.

If the header is in a row other than the first, pass the row number to header. This will skip the preceding rows.

```
import pandas as pd

df=pd.read_csv("temp.csv",names=['a','b','c','d','e'],header=0)
print df
```

Its **output** is as follows −

```
      a        b    c           d        e
0  S.No     Name   Age       City   Salary
1     1      Tom   28     Toronto    20000
2     2      Lee   32    HongKong     3000
3     3   Steven   43    Bay Area     8300
4     4      Ram   38   Hyderabad     3900
```

skiprows

skiprows skips the number of rows specified.

```
import pandas as pd

df=pd.read_csv("temp.csv", skiprows=2)
print df
```

Its **output** is as follows −

```
    2      Lee   32    HongKong   3000
0   3   Steven   43    Bay Area   8300
1   4      Ram   38   Hyderabad   3900
```

Sparse Data

The **Pandas I/O API** is a set of top level reader functions accessed like **pd.read_csv()** that generally return a Pandas object.

The two workhorse functions for reading text files (or the flat files) are **read_csv()** and **read_table()**. They both use the same parsing code to intelligently convert tabular data into a **DataFrame** object −

```
pandas.read_csv(filepath_or_buffer, sep=',', delimiter=None,
header='infer',
names=None, index_col=None, usecols=None
pandas.read_csv(filepath_or_buffer, sep='\t', delimiter=None,
header='infer',
names=None, index_col=None, usecols=None
```

Here is how the **csv** file data looks like −

```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Save this data as **temp.csv** and conduct operations on it.


```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Save this data as **temp.csv** and conduct operations on it.

read.csv

read.csv reads data from the csv files and creates a DataFrame object.

```
import pandas as pd

df=pd.read_csv("temp.csv")
print df
```

Its **output** is as follows −

```
   S.No     Name   Age       City   Salary
0     1      Tom    28    Toronto    20000
1     2      Lee    32   HongKong     3000
2     3   Steven    43   Bay Area     8300
3     4      Ram    38  Hyderabad     3900
```

custom index

This specifies a column in the csv file to customize the index using **index_col.**

```
import pandas as pd

df=pd.read_csv("temp.csv",index_col=['S.No'])
print df
```

Its **output** is as follows −

```
S.No    Name    Age        City    Salary
1        Tom     28     Toronto     20000
2        Lee     32    HongKong      3000
3     Steven     43    Bay Area      8300
4        Ram     38   Hyderabad      3900
```

Converters

dtype of the columns can be passed as a dict.

```
import pandas as pd

df = pd.read_csv("temp.csv", dtype={'Salary': np.float64})
print df.dtypes
```

Its **output** is as follows −

```
S.No       int64
Name      object
Age        int64
City      object
Salary   float64
dtype: object
```

By default, the **dtype** of the Salary column is **int**, but the result shows it as **float** because we have explicitly casted the type.

Thus, the data looks like float −

```
  S.No    Name    Age       City     Salary
0    1     Tom     28     Toronto    20000.0
1    2     Lee     32    HongKong     3000.0
2    3  Steven     43    Bay Area     8300.0
3    4     Ram     38   Hyderabad     3900.0
```

header_names

Specify the names of the header using the names argument.

```
import pandas as pd

df=pd.read_csv("temp.csv", names=['a', 'b', 'c','d','e'])
print df
```

Its **output** is as follows −

```
       a         b    c           d        e
0   S.No      Name   Age       City   Salary
1      1       Tom   28     Toronto    20000
2      2       Lee   32    HongKong     3000
3      3    Steven   43    Bay Area     8300
4      4       Ram   38   Hyderabad     3900
```

Observe, the header names are appended with the custom names, but the header in the file has not been eliminated. Now, we use the header argument to remove that.

If the header is in a row other than the first, pass the row number to header. This will skip the preceding rows.

```
import pandas as pd

df=pd.read_csv("temp.csv",names=['a','b','c','d','e'],header=0)
print df
```

Its **output** is as follows −

```
      a         b    c           d        e
0  S.No      Name   Age       City   Salary
1     1       Tom   28     Toronto    20000
2     2       Lee   32    HongKong     3000
3     3    Steven   43    Bay Area     8300
4     4       Ram   38   Hyderabad     3900
```

skiprows

skiprows skips the number of rows specified.

```
import pandas as pd

df=pd.read_csv("temp.csv", skiprows=2)
print df
```

Its **output** is as follows −

```
    2      Lee   32    HongKong   3000
0   3   Steven   43    Bay Area   8300
1   4      Ram   38   Hyderabad   3900
```