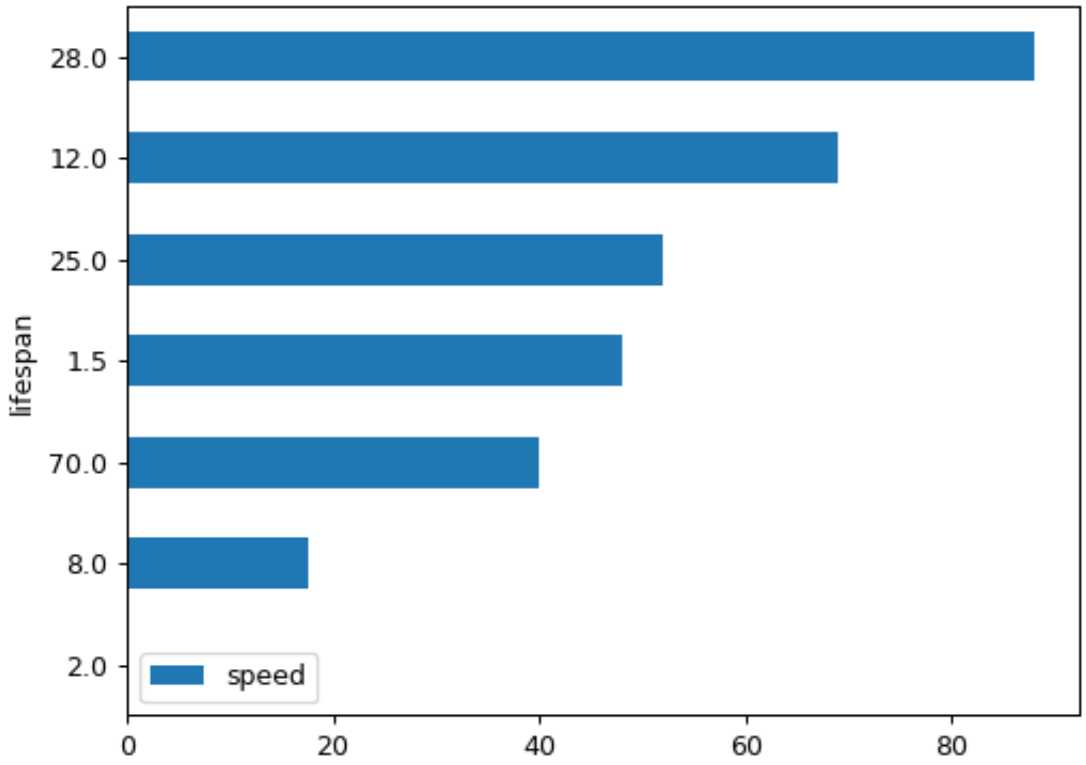
(continued from previous page)

```
...           'rabbit', 'giraffe', 'coyote', 'horse']
>>> df = pd.DataFrame({'speed': speed,
...                    'lifespan': lifespan}, index=index)
>>> ax = df.plot.barh(x='lifespan')
```



pandas.Series.plot.box

`Series.plot.`**box**(*self*, *by=None*, ***kwargs*)

Make a box plot of the DataFrame columns.

A box plot is a method for graphically depicting groups of numerical data through their quartiles. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2). The whiskers extend from the edges of box to show the range of the data. The position of the whiskers is set by default to 1.5*IQR (IQR = Q3 - Q1) from the edges of the box. Outlier points are those past the end of the whiskers.

For further details see Wikipedia's entry for boxplot.

A consideration when using this chart is that the box and the whiskers can overlap, which is very common when plotting small sets of data.

Parameters

by [str or sequence] Column in the DataFrame to group by.

****kwargs** Additional keywords are documented in *DataFrame.plot()*.

Returns

`matplotlib.axes.Axes` **or numpy.ndarray of them**

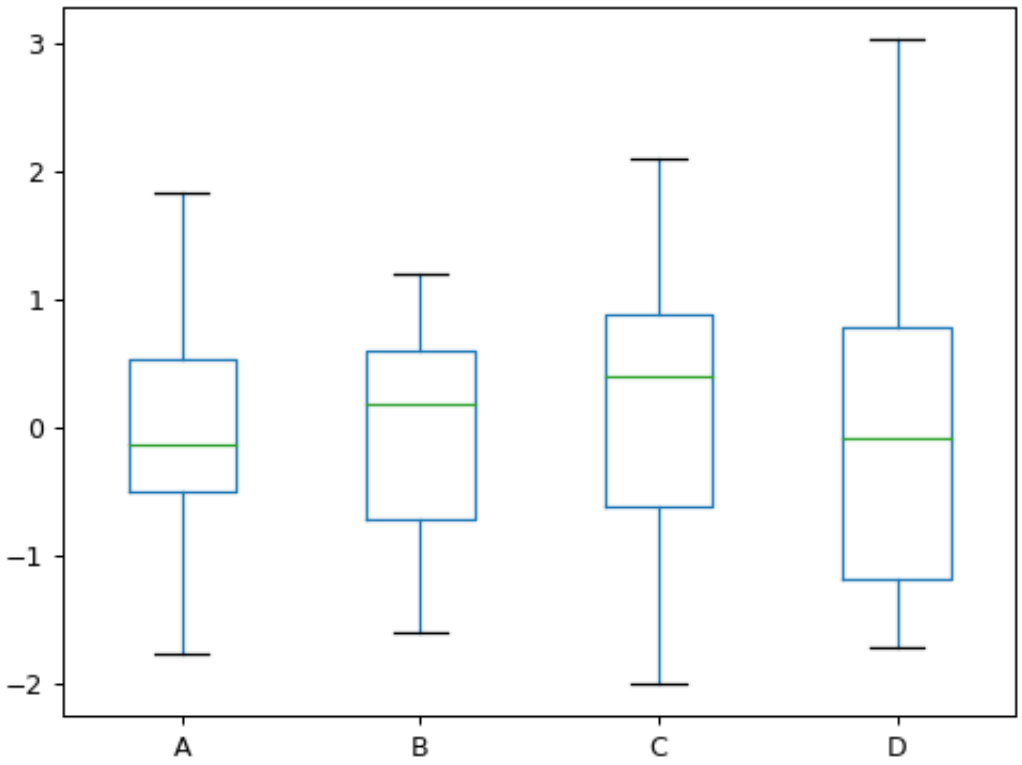
See also:

`DataFrame.boxplot`	Another method to draw a box plot.
`Series.plot.box`	Draw a box plot from a Series object.
`matplotlib.pyplot.boxplot`	Draw a box plot in matplotlib.

Examples

Draw a box plot from a DataFrame with four columns of randomly generated data.

```
>>> data = np.random.randn(25, 4)
>>> df = pd.DataFrame(data, columns=list('ABCD'))
>>> ax = df.plot.box()
```



pandas.Series.plot.density

Series.plot.**density**(*self*, *bw_method=None*, *ind=None*, ***kwargs*)

Generate Kernel Density Estimate plot using Gaussian kernels.

In statistics, kernel density estimation (KDE) is a non-parametric way to estimate the probability density function (PDF) of a random variable. This function uses Gaussian kernels and includes automatic bandwidth determination.

Parameters

bw_method [str, scalar or callable, optional] The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If None (default), 'scott' is used. See `scipy.stats.gaussian_kde` for more information.

ind [NumPy array or int, optional] Evaluation points for the estimated PDF. If None (default), 1000 equally spaced points are used. If *ind* is a NumPy array, the KDE is evaluated at the points passed. If *ind* is an integer, *ind* number of equally spaced points are used.

****kwargs** Additional keyword arguments are documented in `pandas.%(this-datatype)s.plot()`.

Returns

matplotlib.axes.Axes or numpy.ndarray of them

See also:

scipy.stats.gaussian_kde Representation of a kernel-density estimate using Gaussian kernels. This is the function used internally to estimate the PDF.

Examples

Given a Series of points randomly sampled from an unknown distribution, estimate its PDF using KDE with automatic bandwidth determination and plot the results, evaluating them at 1000 equally spaced points (default):

```
>>> s = pd.Series([1, 2, 2.5, 3, 3.5, 4, 5])
>>> ax = s.plot.kde()
```

A scalar bandwidth can be specified. Using a small bandwidth value can lead to over-fitting, while using a large bandwidth value may result in under-fitting:

```
>>> ax = s.plot.kde(bw_method=0.3)
```

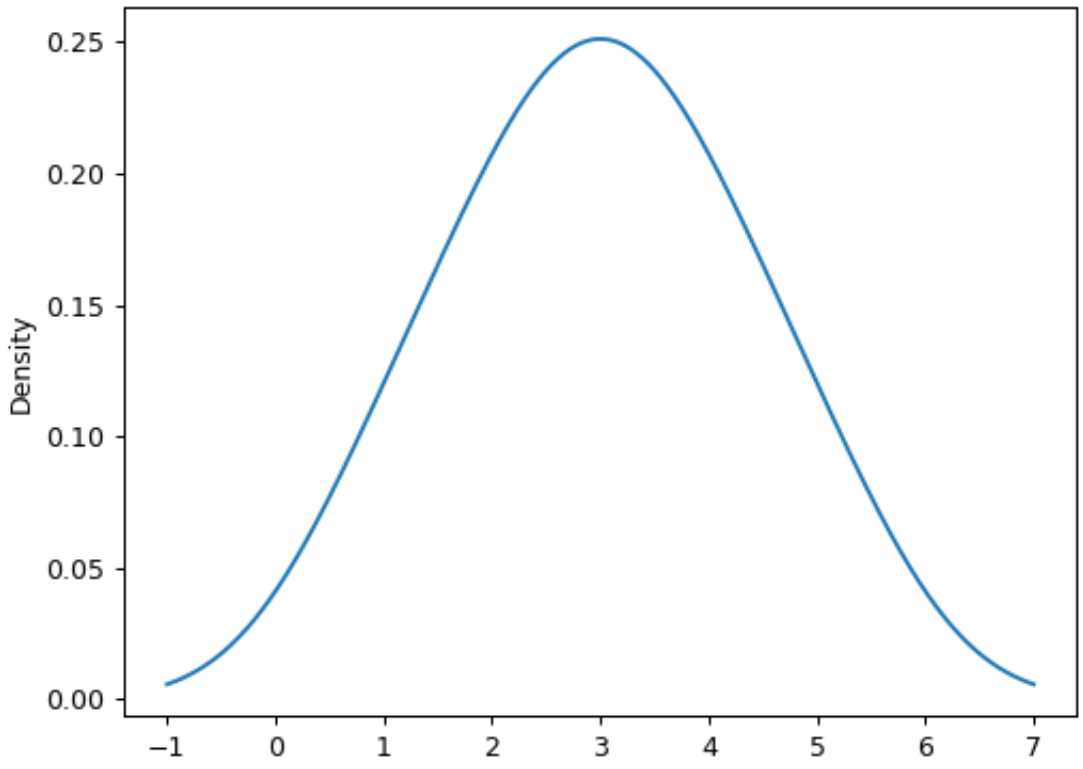
```
>>> ax = s.plot.kde(bw_method=3)
```

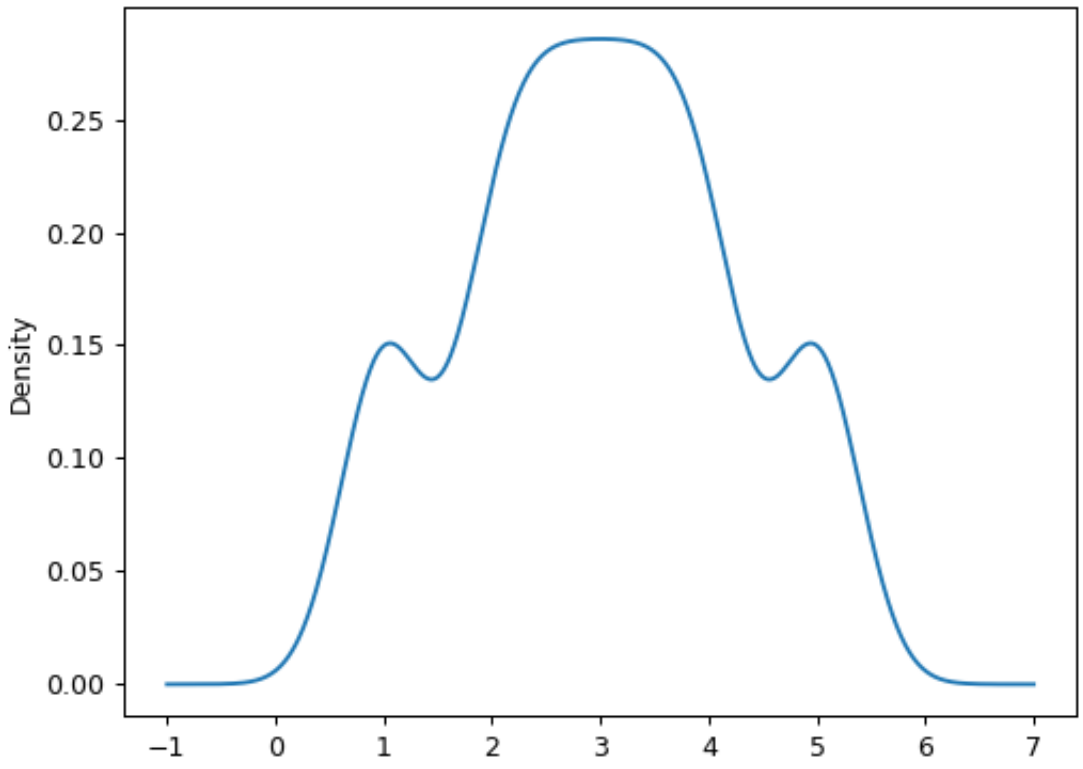
Finally, the *ind* parameter determines the evaluation points for the plot of the estimated PDF:

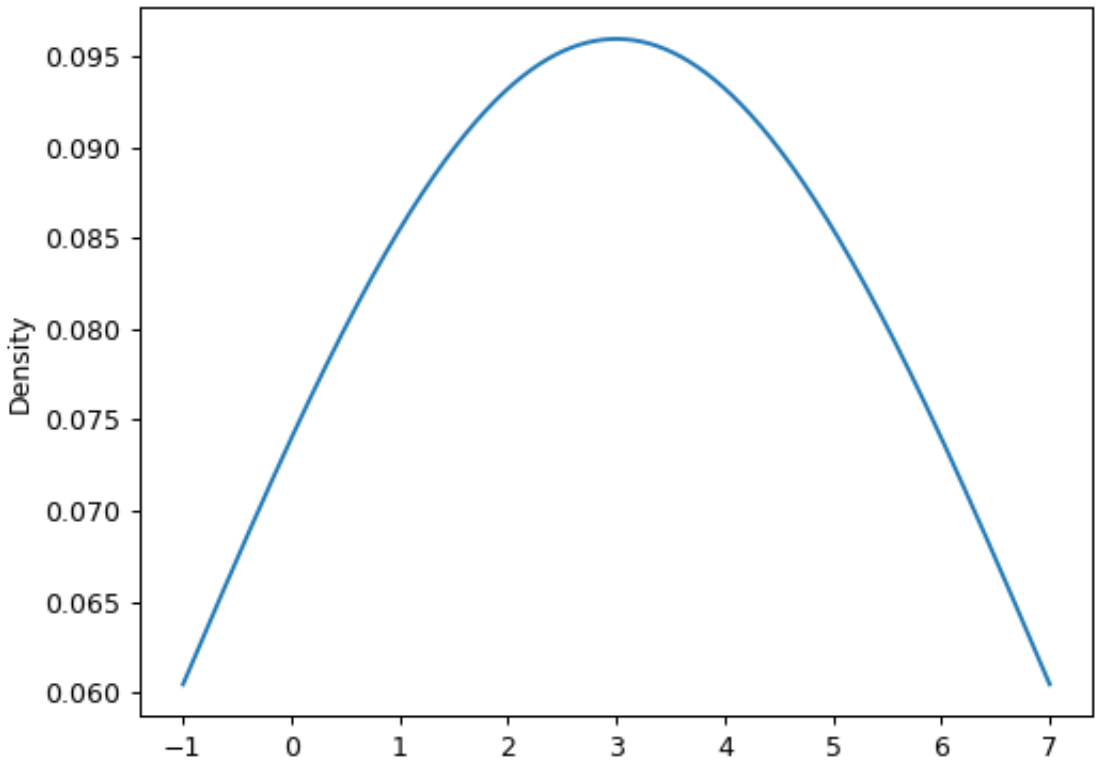
```
>>> ax = s.plot.kde(ind=[1, 2, 3, 4, 5])
```

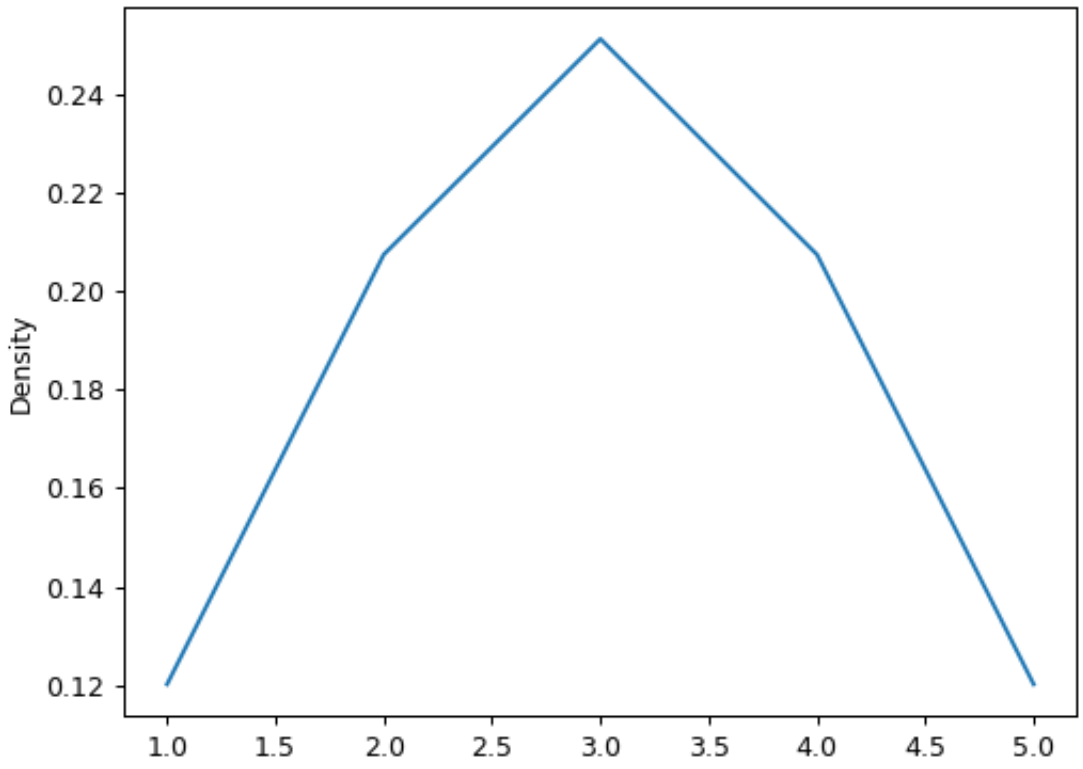
For DataFrame, it works in the same way:

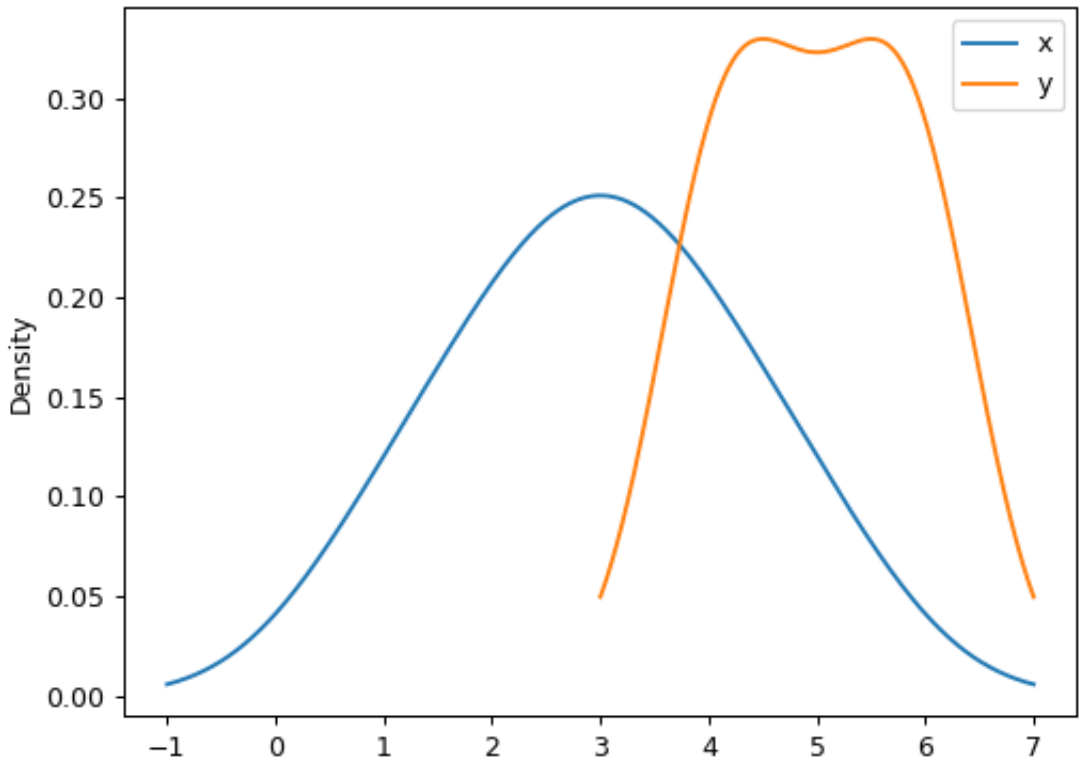
```
>>> df = pd.DataFrame({
...     'x': [1, 2, 2.5, 3, 3.5, 4, 5],
...     'y': [4, 4, 4.5, 5, 5.5, 6, 6],
... })
>>> ax = df.plot.kde()
```


0.25
0.20
0.15
0.10
0.05
0.00
Density
−1
0
1
2
3
4
5
6
7


Density
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4
5
6
7

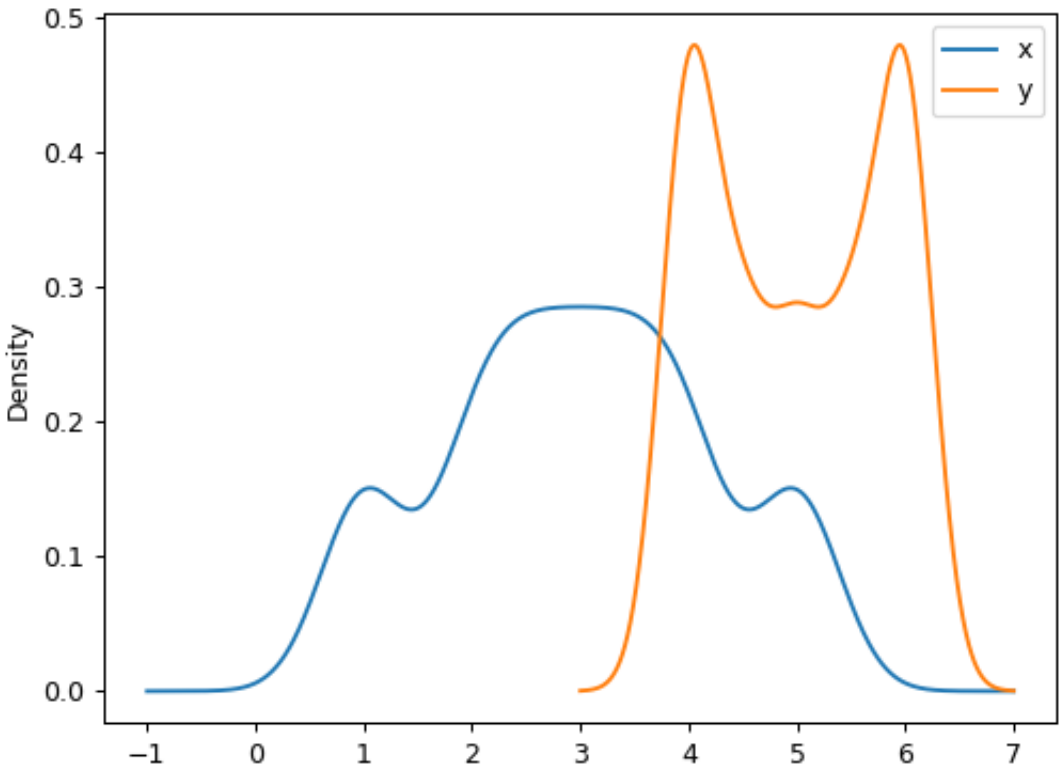

Density
0.095
0.090
0.085
0.080
0.075
0.070
0.065
0.060
−1
0
1
2
3
4
5
6
7


Density
0.24
0.22
0.20
0.18
0.16
0.14
0.12
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


x
y
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4
5
6
7

A scalar bandwidth can be specified. Using a small bandwidth value can lead to over-fitting, while using a large bandwidth value may result in under-fitting:

```
>>> ax = df.plot.kde(bw_method=0.3)
```




```
>>> ax = df.plot.kde(bw_method=3)
```

Finally, the *ind* parameter determines the evaluation points for the plot of the estimated PDF:

```
>>> ax = df.plot.kde(ind=[1, 2, 3, 4, 5, 6])
```

pandas.Series.plot.hist

`Series.plot.`**`hist`**(*self*, *by=None*, *bins=10*, ***kwargs*)

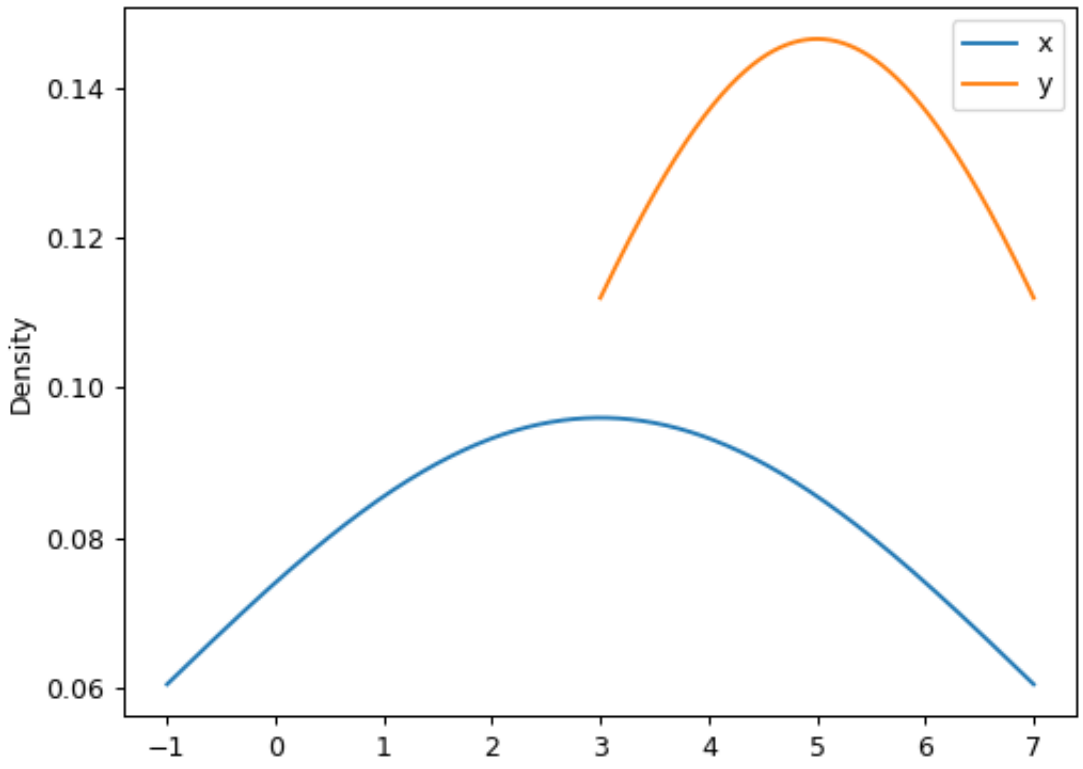
Draw one histogram of the DataFrame's columns.

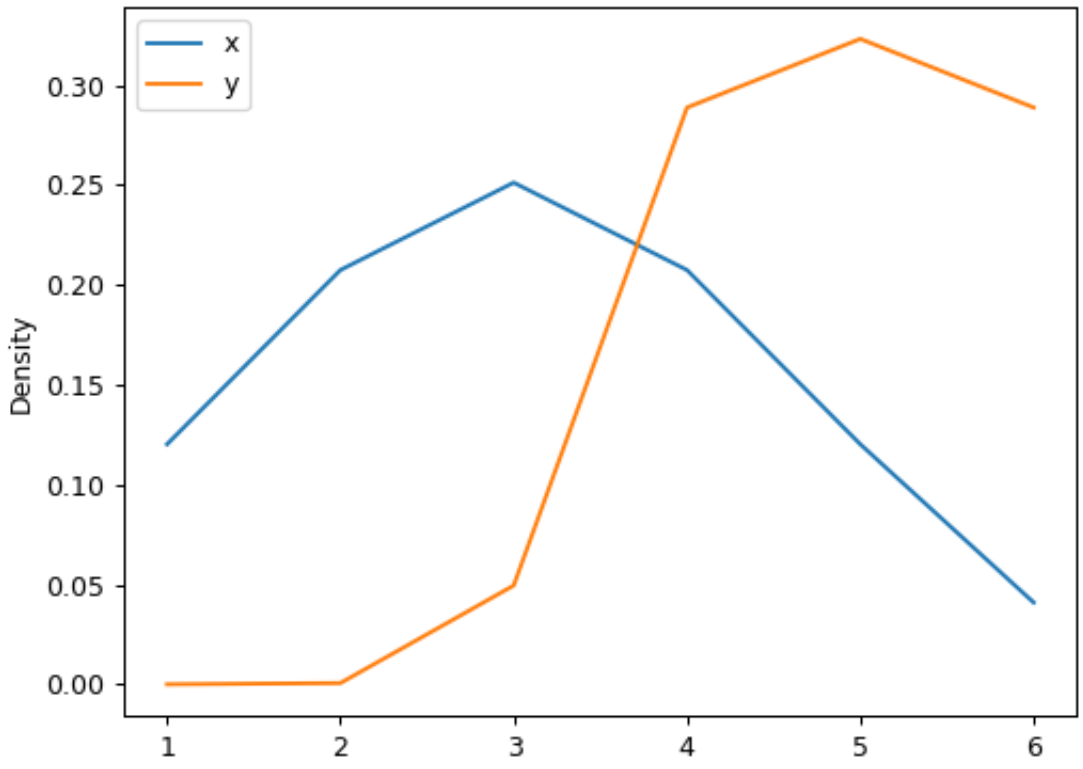
A histogram is a representation of the distribution of data. This function groups the values of all given Series in the DataFrame into bins and draws all bins in one `matplotlib.axes.Axes`. This is useful when the DataFrame's Series are in a similar scale.

Parameters

by [str or sequence, optional] Column in the DataFrame to group by.

bins [int, default 10] Number of histogram bins to be used.


x
y
0.14
0.12
0.10
0.08
0.06
Density
−1
0
1
2
3
4
5
6
7


x
y
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6

****kwargs** Additional keyword arguments are documented in *DataFrame.plot()*.

Returns

class:*matplotlib.AxesSubplot* Return a histogram plot.

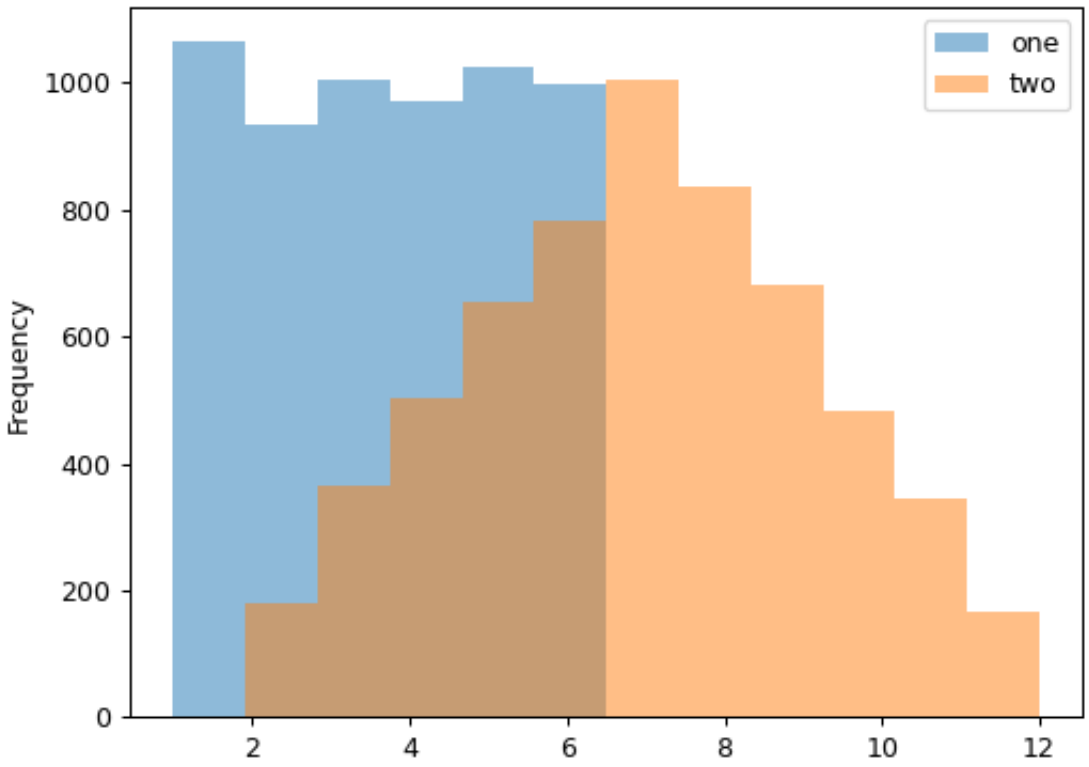
See also:

DataFrame.hist Draw histograms per DataFrame's Series.
Series.hist Draw a histogram with Series' data.

Examples

When we draw a dice 6000 times, we expect to get each value around 1000 times. But when we draw two dices and sum the result, the distribution is going to be quite different. A histogram illustrates those distributions.

```
>>> df = pd.DataFrame(
...     np.random.randint(1, 7, 6000),
...     columns = ['one'])
>>> df['two'] = df['one'] + np.random.randint(1, 7, 6000)
>>> ax = df.plot.hist(bins=12, alpha=0.5)
```



pandas.Series.plot.kde

Series.plot.**kde**(*self*, *bw_method=None*, *ind=None*, ***kwargs*)

Generate Kernel Density Estimate plot using Gaussian kernels.

In statistics, kernel density estimation (KDE) is a non-parametric way to estimate the probability density function (PDF) of a random variable. This function uses Gaussian kernels and includes automatic bandwidth determination.

Parameters

bw_method [str, scalar or callable, optional] The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If None (default), 'scott' is used. See `scipy.stats.gaussian_kde` for more information.

ind [NumPy array or int, optional] Evaluation points for the estimated PDF. If None (default), 1000 equally spaced points are used. If *ind* is a NumPy array, the KDE is evaluated at the points passed. If *ind* is an integer, *ind* number of equally spaced points are used.

****kwargs** Additional keyword arguments are documented in `pandas.%(this-datatype)s.plot()`.

Returns

matplotlib.axes.Axes or numpy.ndarray of them

See also:

scipy.stats.gaussian_kde Representation of a kernel-density estimate using Gaussian kernels. This is the function used internally to estimate the PDF.

Examples

Given a Series of points randomly sampled from an unknown distribution, estimate its PDF using KDE with automatic bandwidth determination and plot the results, evaluating them at 1000 equally spaced points (default):

```
>>> s = pd.Series([1, 2, 2.5, 3, 3.5, 4, 5])
>>> ax = s.plot.kde()
```

A scalar bandwidth can be specified. Using a small bandwidth value can lead to over-fitting, while using a large bandwidth value may result in under-fitting:

```
>>> ax = s.plot.kde(bw_method=0.3)
```

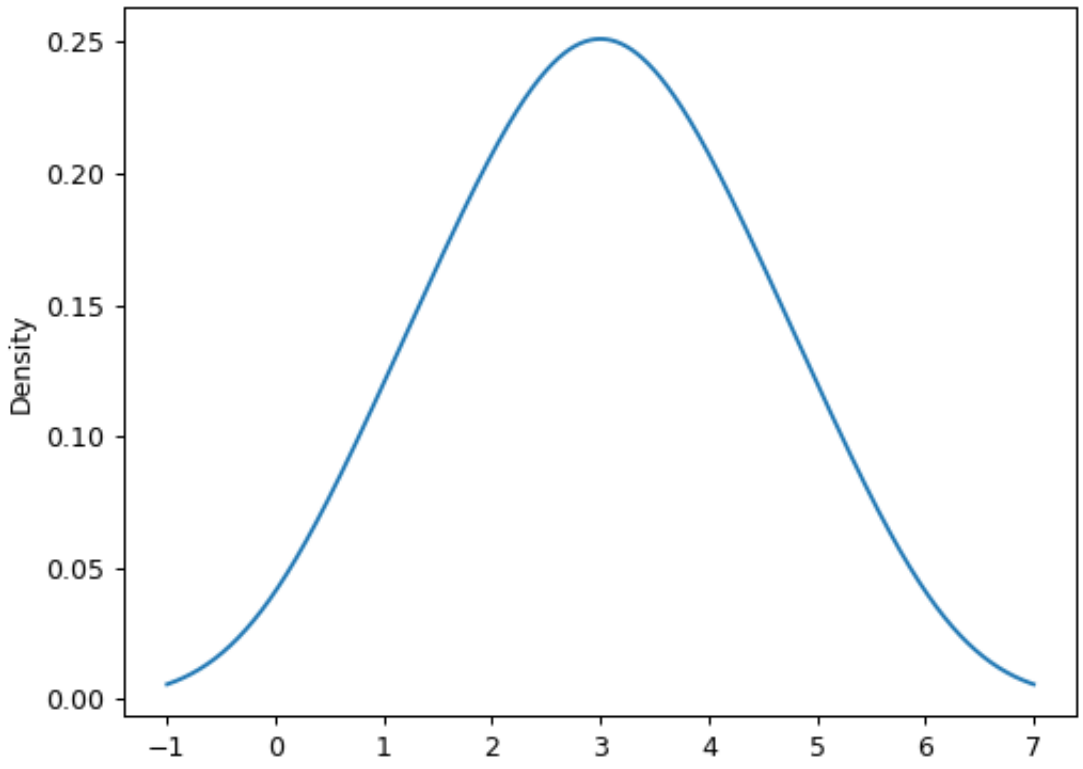
```
>>> ax = s.plot.kde(bw_method=3)
```

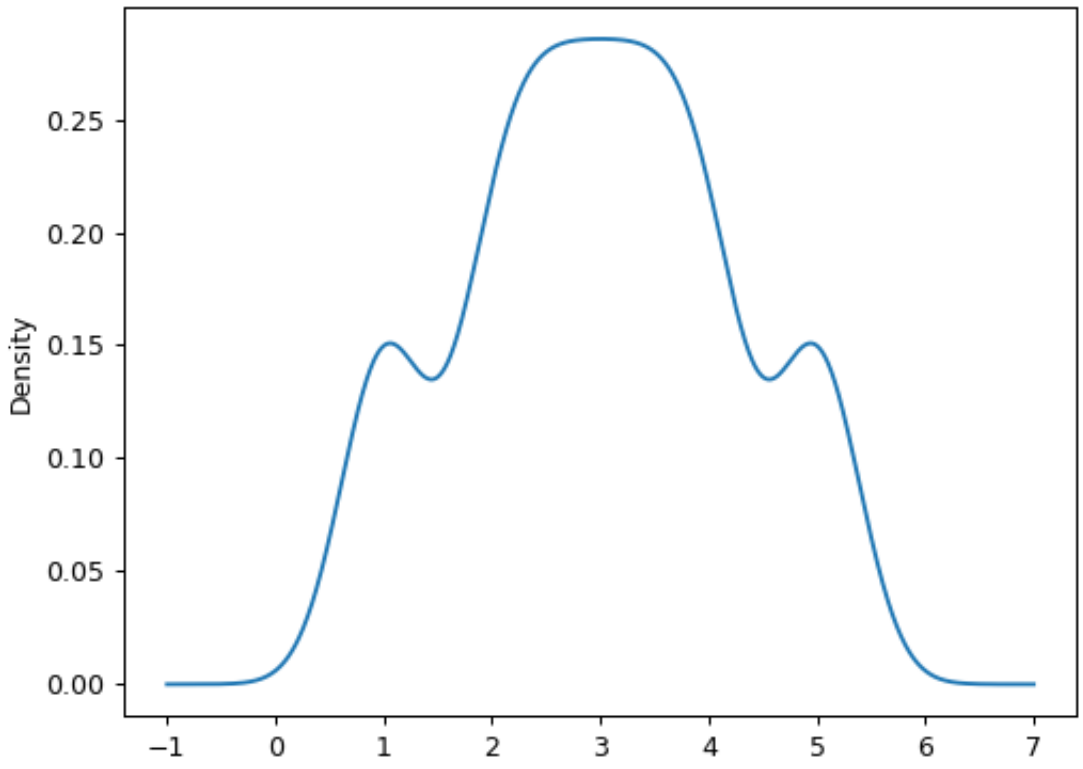
Finally, the *ind* parameter determines the evaluation points for the plot of the estimated PDF:

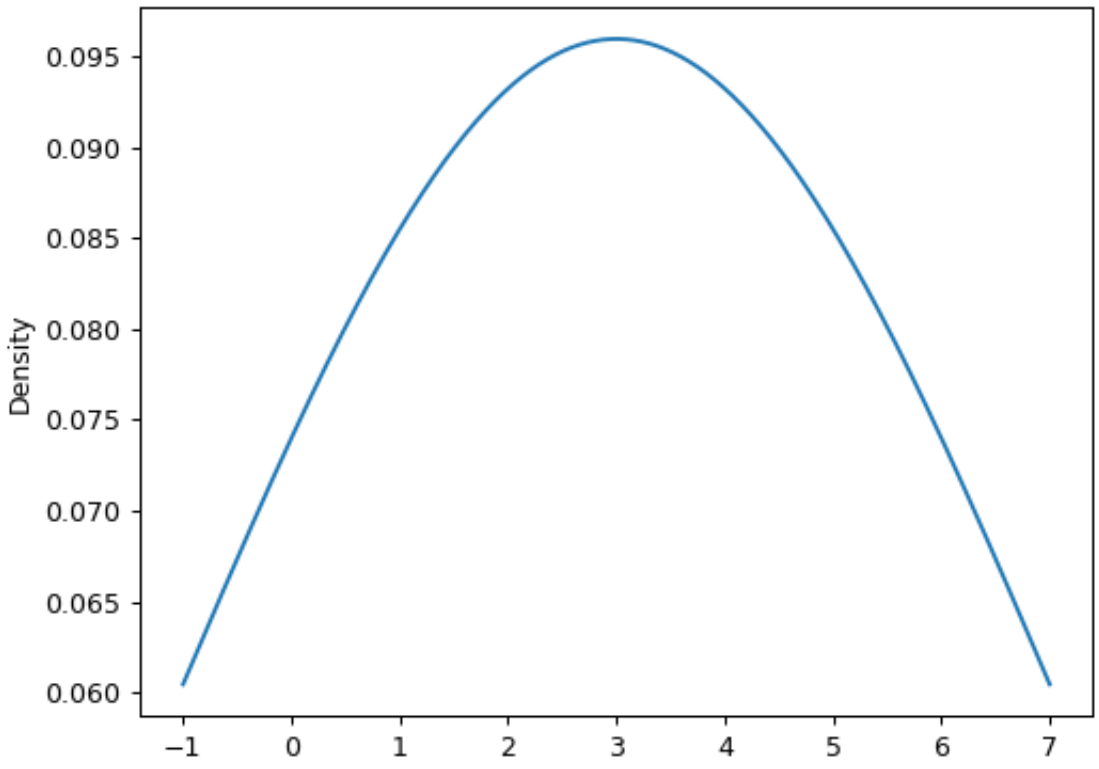
```
>>> ax = s.plot.kde(ind=[1, 2, 3, 4, 5])
```

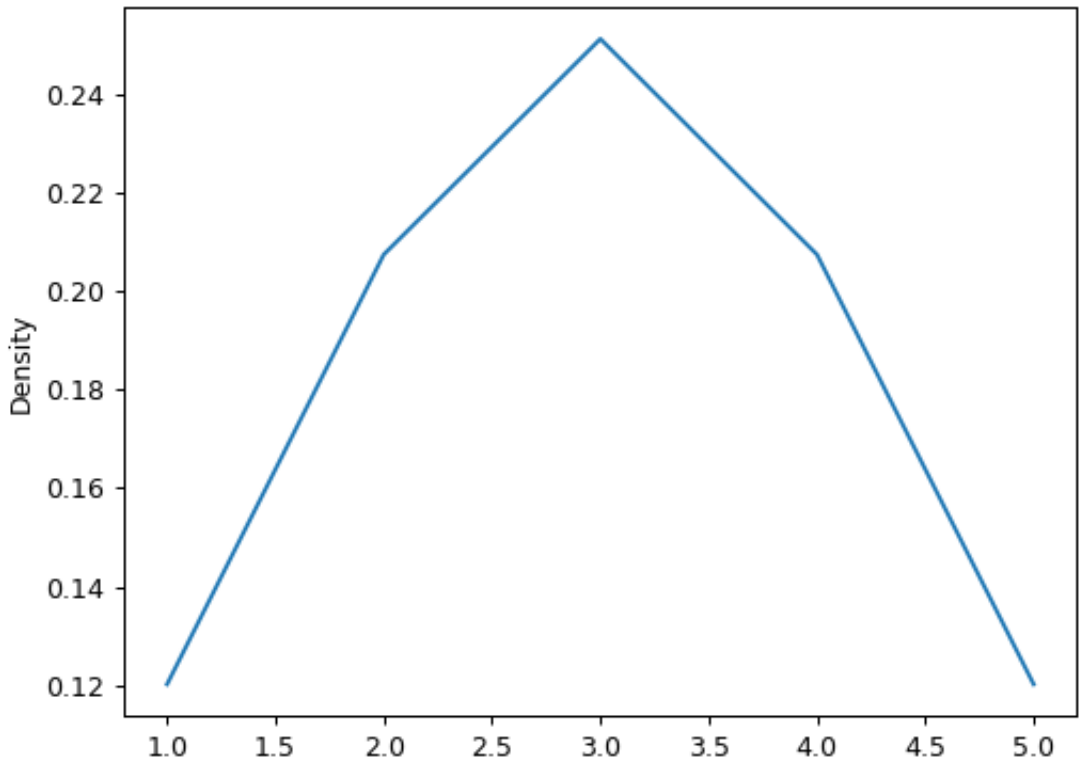
For DataFrame, it works in the same way:

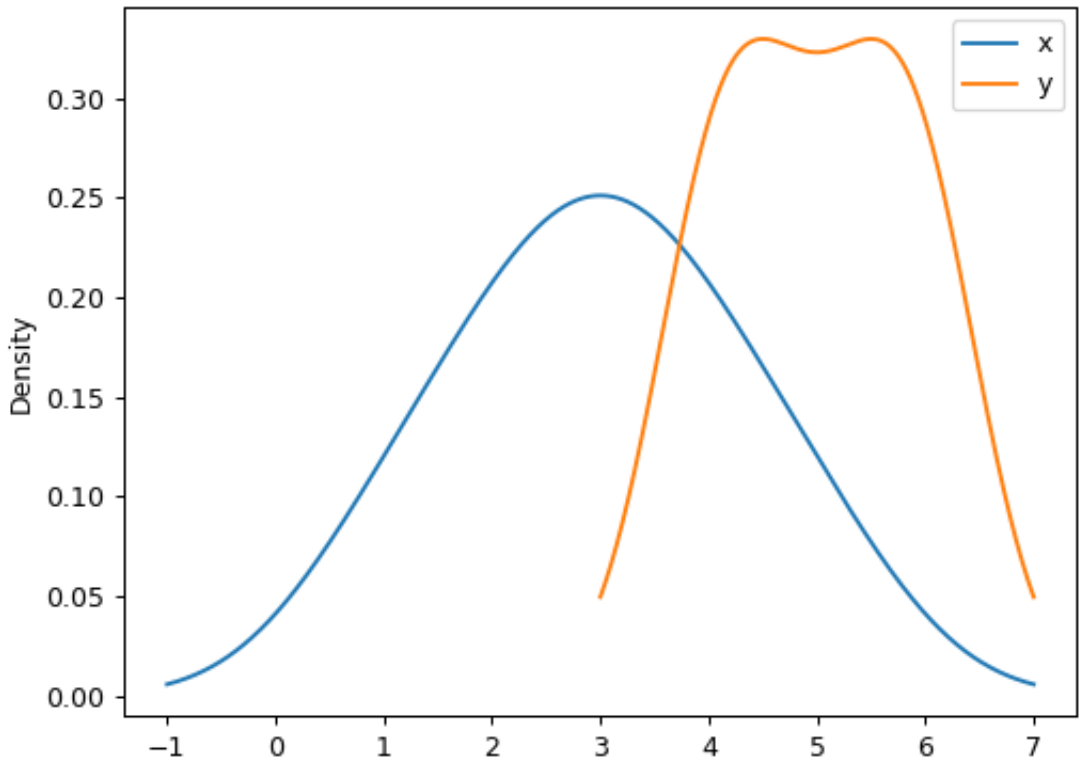
```
>>> df = pd.DataFrame({
...     'x': [1, 2, 2.5, 3, 3.5, 4, 5],
...     'y': [4, 4, 4.5, 5, 5.5, 6, 6],
... })
>>> ax = df.plot.kde()
```


0.25
0.20
0.15
0.10
0.05
0.00
Density
−1
0
1
2
3
4
5
6
7


Density
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4
5
6
7

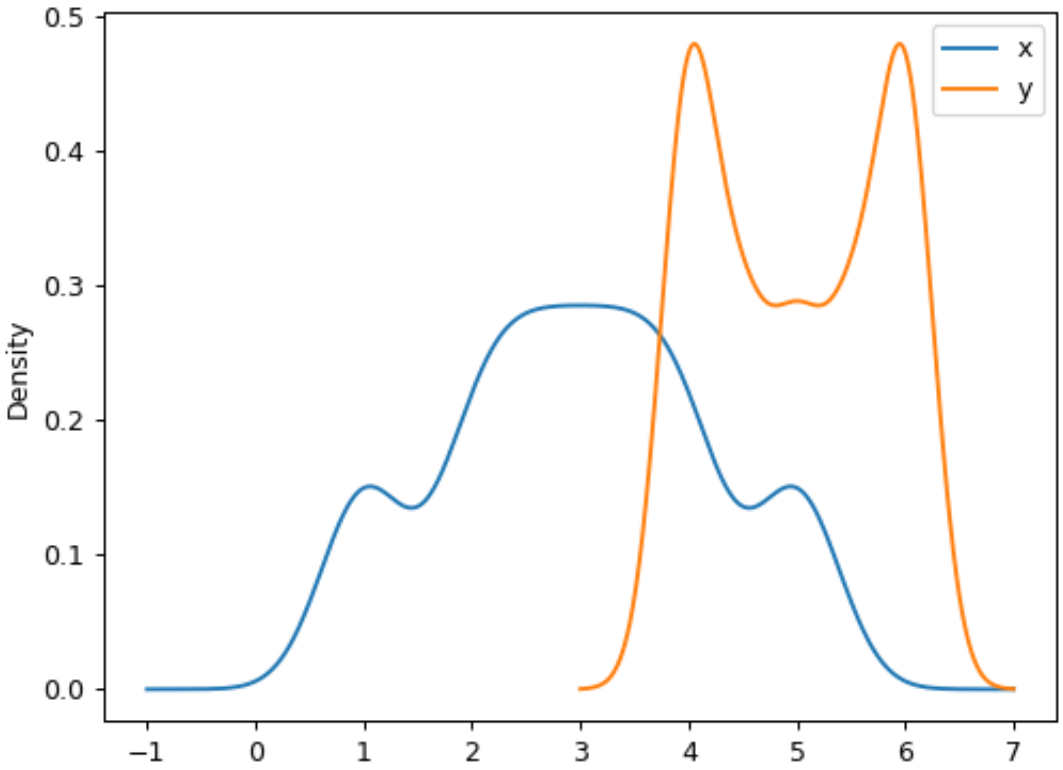

Density
0.095
0.090
0.085
0.080
0.075
0.070
0.065
0.060
−1
0
1
2
3
4
5
6
7


Density
0.24
0.22
0.20
0.18
0.16
0.14
0.12
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


x
y
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4
5
6
7

A scalar bandwidth can be specified. Using a small bandwidth value can lead to over-fitting, while using a large bandwidth value may result in under-fitting:

```
>>> ax = df.plot.kde(bw_method=0.3)
```




```
>>> ax = df.plot.kde(bw_method=3)
```

Finally, the *ind* parameter determines the evaluation points for the plot of the estimated PDF:

```
>>> ax = df.plot.kde(ind=[1, 2, 3, 4, 5, 6])
```

pandas.Series.plot.line

`Series.plot.`**`line`**(*self*, *x=None*, *y=None*, ***kwargs*)

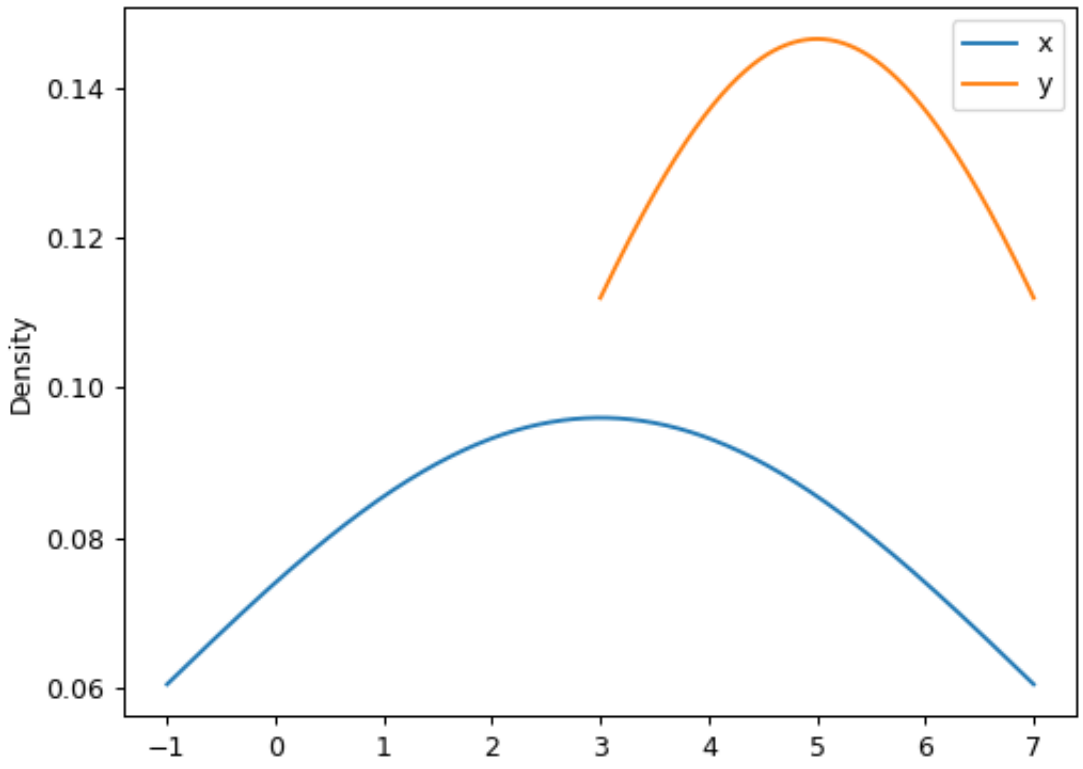
Plot Series or DataFrame as lines.

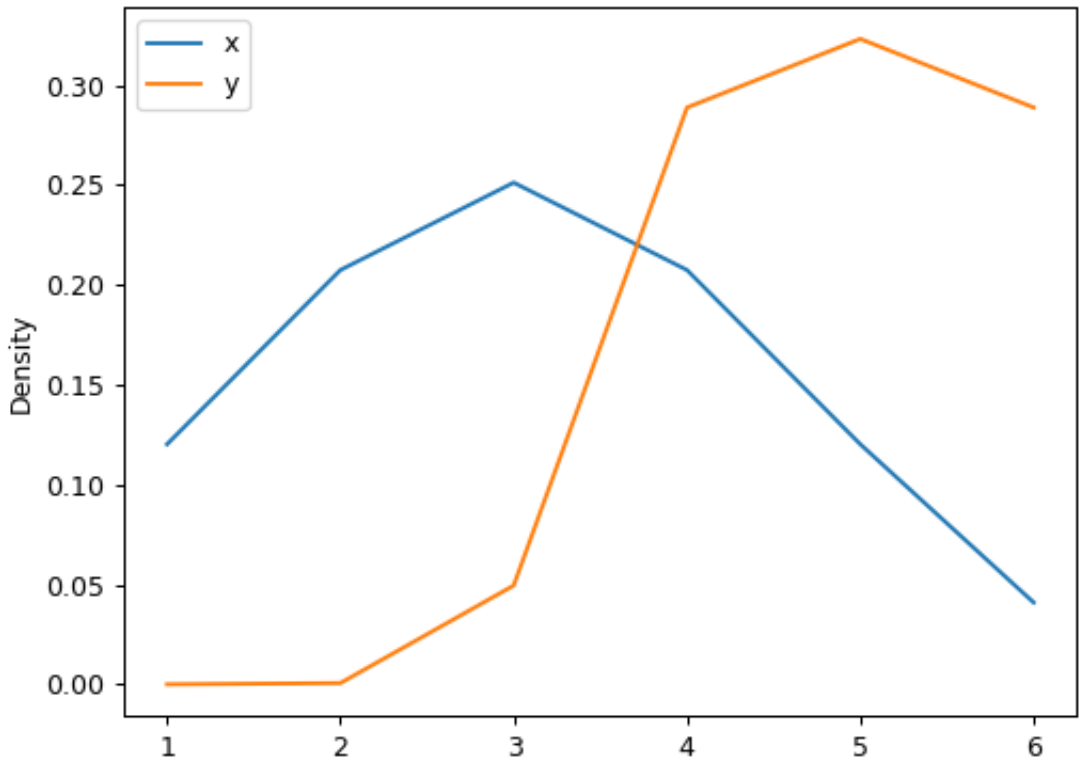
This function is useful to plot lines using DataFrame's values as coordinates.

Parameters

x [int or str, optional] Columns to use for the horizontal axis. Either the location or the label of the columns to be used. By default, it will use the DataFrame indices.

y [int, str, or list of them, optional] The values to be plotted. Either the location or the label of the columns to be used. By default, it will use the remaining DataFrame numeric columns.


x
y
0.14
0.12
0.10
0.08
0.06
Density
−1
0
1
2
3
4
5
6
7


x
y
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6

****kwargs** Keyword arguments to pass on to *DataFrame.plot()*.

Returns

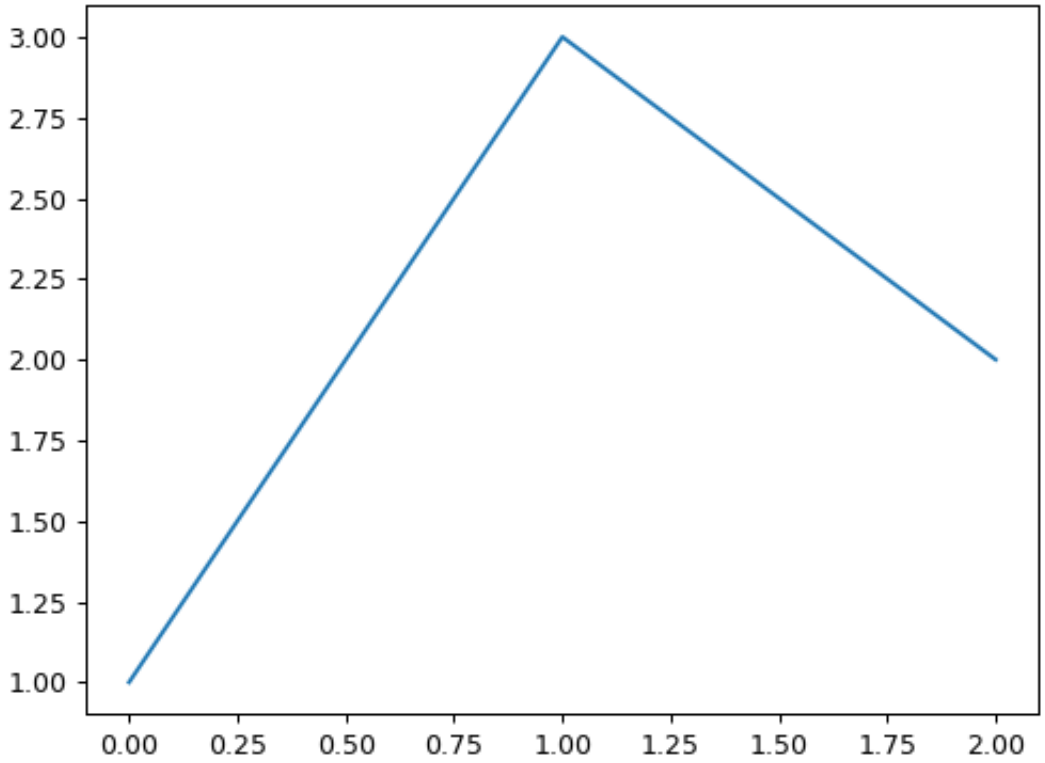
matplotlib.axes.Axes or numpy.ndarray Return an ndarray when `subplots=True`.

See also:

matplotlib.pyplot.plot Plot y versus x as lines and/or markers.

Examples

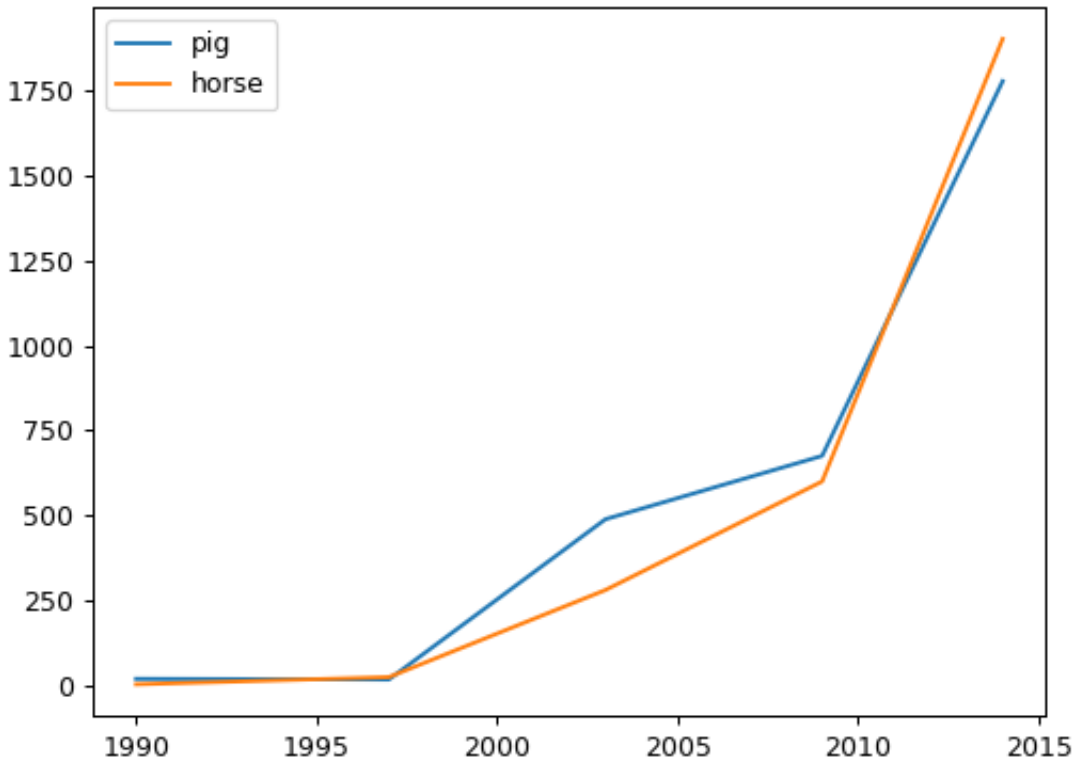
```
>>> s = pd.Series([1, 3, 2])
>>> s.plot.line()
```



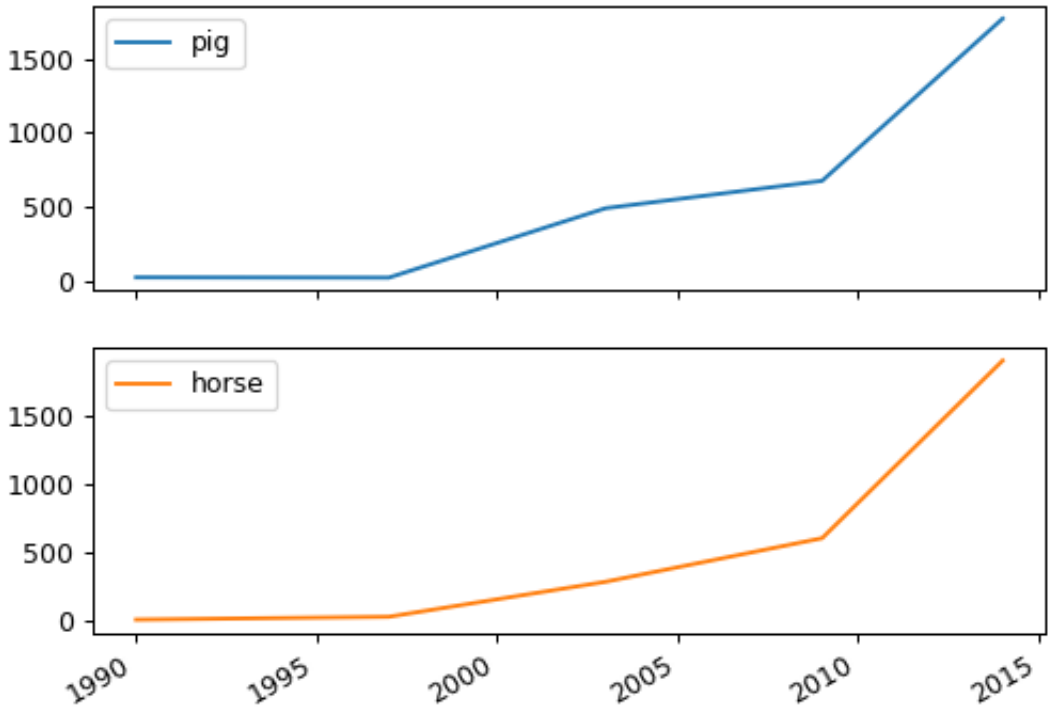
The following example shows the populations for some animals over the years.

```
>>> df = pd.DataFrame({
...     'pig': [20, 18, 489, 675, 1776],
...     'horse': [4, 25, 281, 600, 1900]
...     }, index=[1990, 1997, 2003, 2009, 2014])
>>> lines = df.plot.line()
```

An example with subplots, so an array of axes is returned.


pig
horse
1750
1500
1250
1000
750
500
250
0
1990
1995
2000
2005
2010
2015

```
>>> axes = df.plot.line(subplots=True)
>>> type(axes)
<class 'numpy.ndarray'>
```



The following example shows the relationship between both populations.

```
>>> lines = df.plot.line(x='pig', y='horse')
```

pandas.Series.plot.pie

`Series.plot.`**`pie`**(*self*, ***kwargs*)

Generate a pie plot.

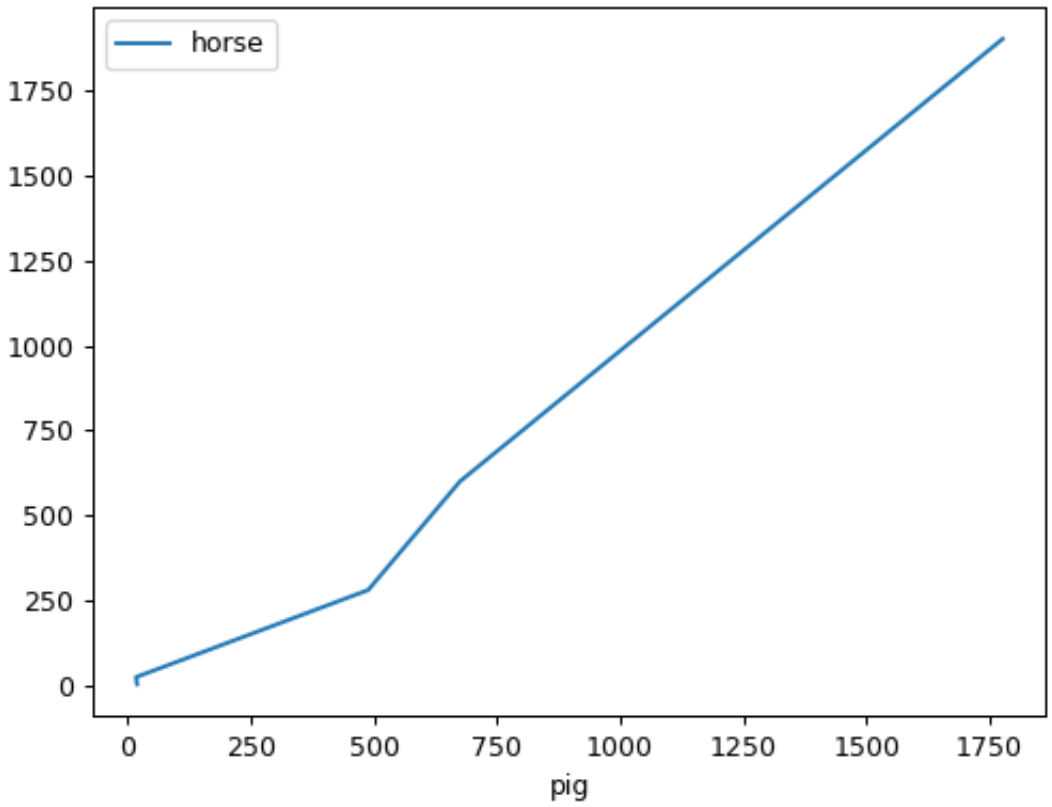
A pie plot is a proportional representation of the numerical data in a column. This function wraps `matplotlib.pyplot.pie()` for the specified column. If no column reference is passed and `subplots=True` a pie plot is drawn for each numerical column independently.

Parameters

y [int or label, optional] Label or position of the column to plot. If not provided, `subplots=True` argument must be passed.

****kwargs** Keyword arguments to pass on to *`DataFrame.plot()`*.

Returns


horse
1750
1500
1250
1000
750
500
250
0
0
250
500
750
1000
1250
1500
1750
pig