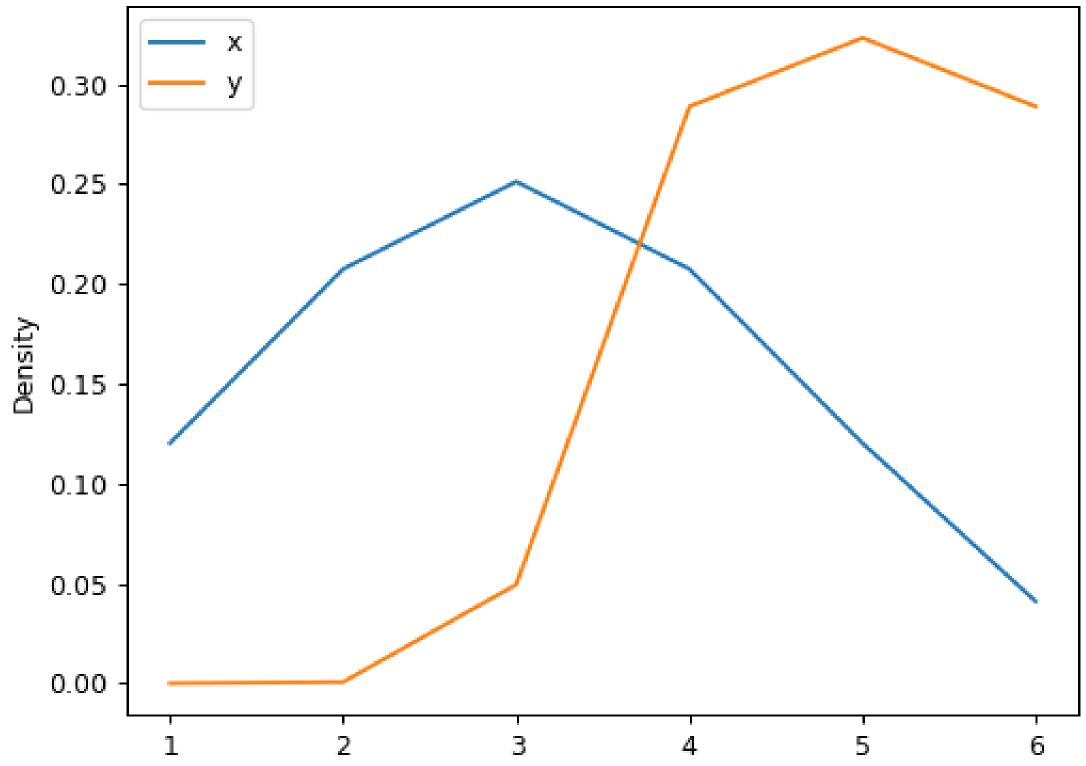

x
y
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
1
2
3
4
5
6

pandas.DataFrame.plot.hexbin

`DataFrame.plot.hexbin`(*self*, *x*, *y*, *C=None*, *reduce_C_function=None*, *gridsize=None*, ***kwargs*)

Generate a hexagonal binning plot.

Generate a hexagonal binning plot of *x* versus *y*. If *C* is *None* (the default), this is a histogram of the number of occurrences of the observations at `(x[i], y[i])`.

If *C* is specified, specifies values at given coordinates `(x[i], y[i])`. These values are accumulated for each hexagonal bin and then reduced according to *reduce_C_function*, having as default the NumPy's mean function (`numpy.mean()`). (If *C* is specified, it must also be a 1-D sequence of the same length as *x* and *y*, or a column label.)

Parameters

x [int or str] The column label or position for x points.

y [int or str] The column label or position for y points.

C [int or str, optional] The column label or position for the value of *(x, y)* point.

reduce_C_function [callable, default *np.mean*] Function of one argument that reduces all the values in a bin to a single number (e.g. *np.mean*, *np.max*, *np.sum*, *np.std*).

gridsize [int or tuple of (int, int), default 100] The number of hexagons in the x-direction. The corresponding number of hexagons in the y-direction is chosen in a way that the hexagons are approximately regular. Alternatively, gridsize can be a tuple with two elements specifying the number of hexagons in the x-direction and the y-direction.

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

Returns

matplotlib.AxesSubplot The matplotlib `Axes` on which the hexbin is plotted.

See also:

`DataFrame.plot` Make plots of a DataFrame.

`matplotlib.pyplot.hexbin` Hexagonal binning plot using matplotlib, the matplotlib function that is used under the hood.

Examples

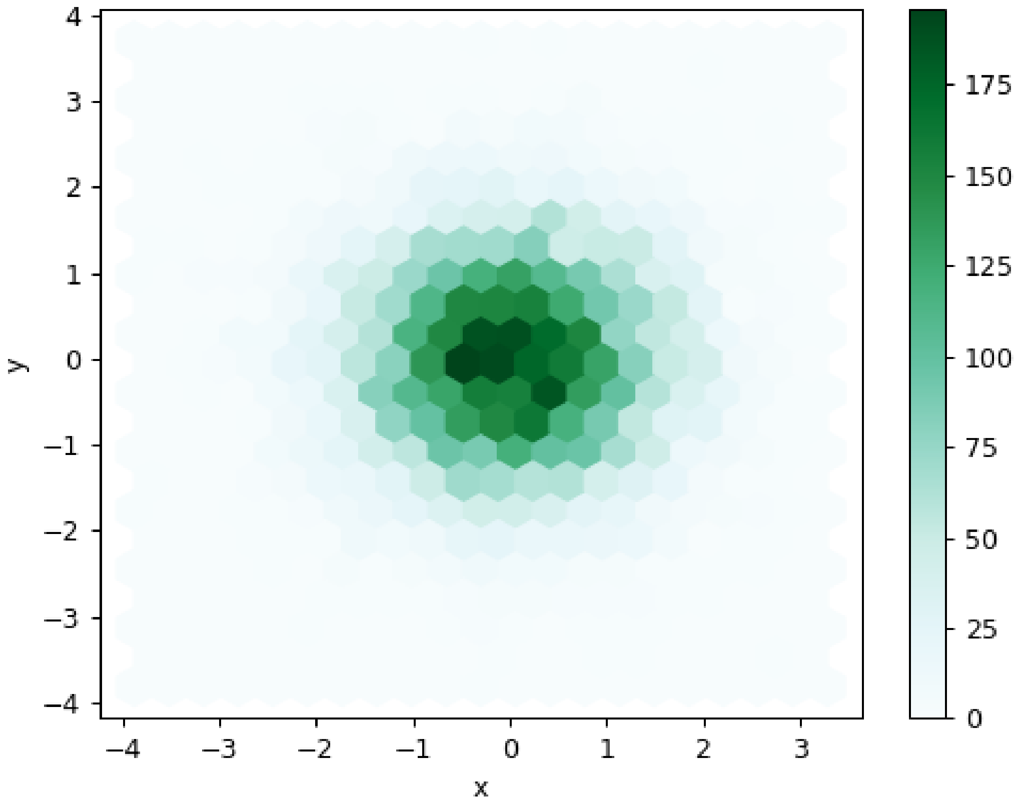
The following examples are generated with random data from a normal distribution.

```
>>> n = 10000
>>> df = pd.DataFrame({'x': np.random.randn(n),
...                    'y': np.random.randn(n)})
>>> ax = df.plot.hexbin(x='x', y='y', gridsize=20)
```

The next example uses *C* and *np.sum* as *reduce_C_function*. Note that *'observations'* values ranges from 1 to 5 but the result plot shows values up to more than 25. This is because of the *reduce_C_function*.

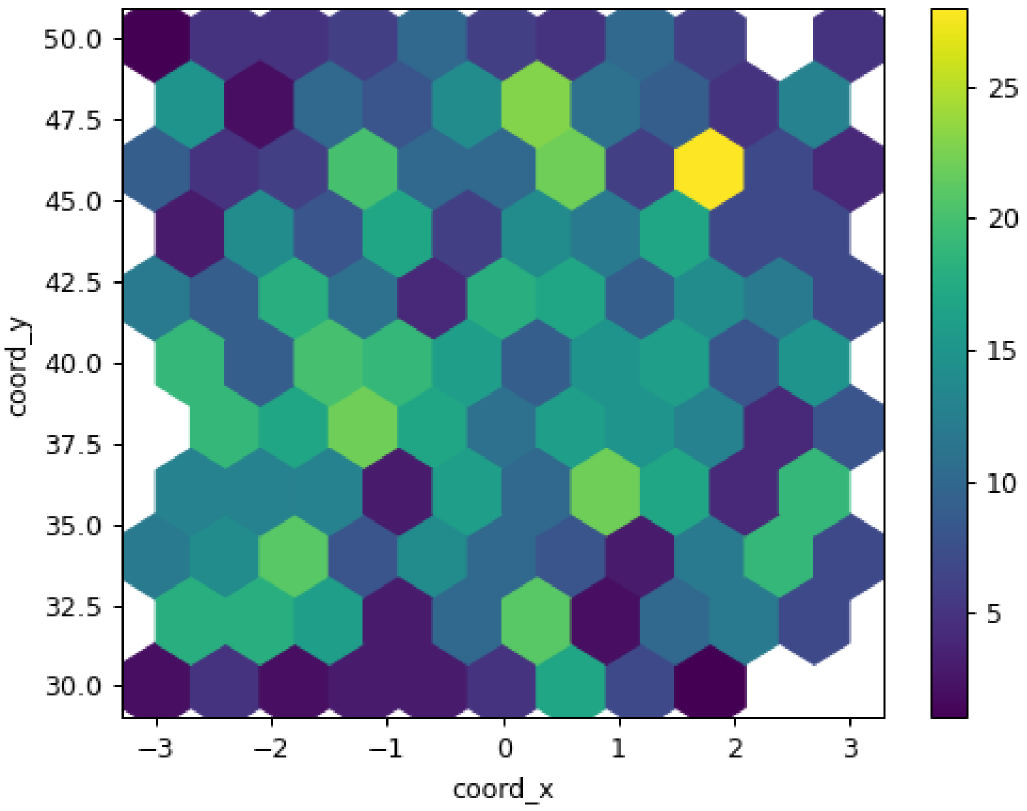
```
>>> n = 500
>>> df = pd.DataFrame({
...     'coord_x': np.random.uniform(-3, 3, size=n),
...     'coord_y': np.random.uniform(30, 50, size=n),
...     'observations': np.random.randint(1,5, size=n)
...     })
>>> ax = df.plot.hexbin(x='coord_x',
...                     y='coord_y',
```

(continues on next page)


y
x
4
3
2
1
0
−1
−2
−3
−4
175
150
125
100
75
50
25
0

(continued from previous page)

```
...                          C='observations',
...                          reduce_C_function=np.sum,
...                          gridsize=10,
...                          cmap="viridis")
```



pandas.DataFrame.plot.hist

`DataFrame.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.

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

Returns

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

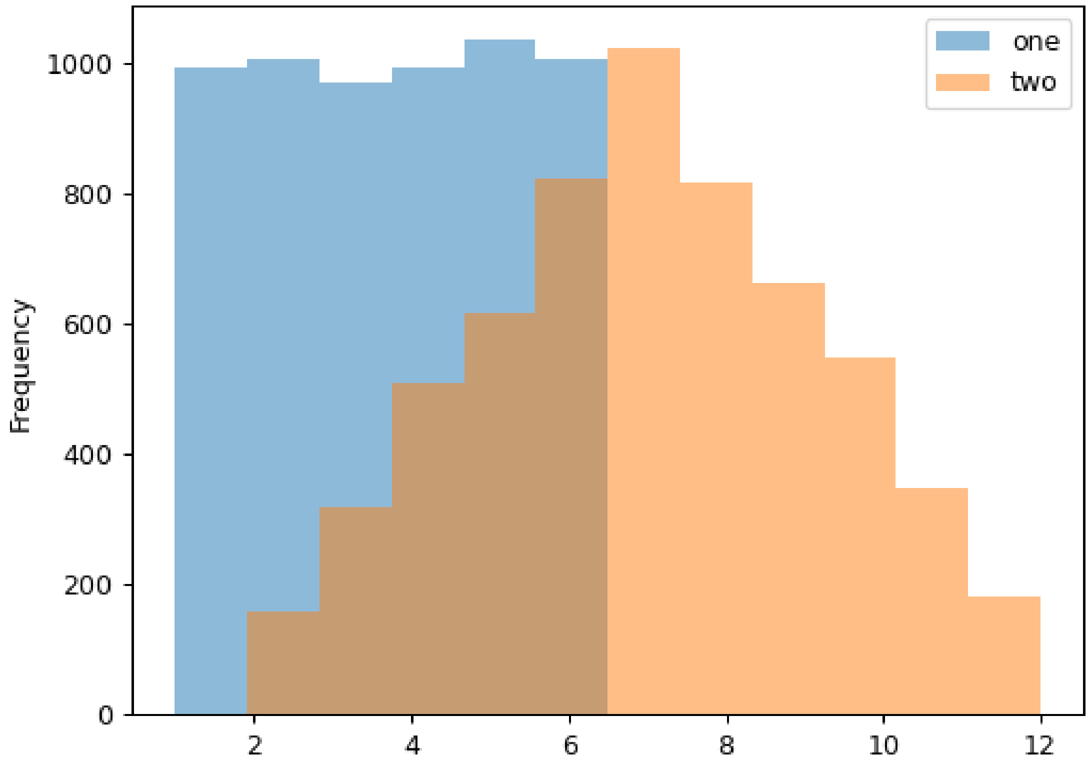
See also:

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.DataFrame.plot.kde

`DataFrame.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)
```

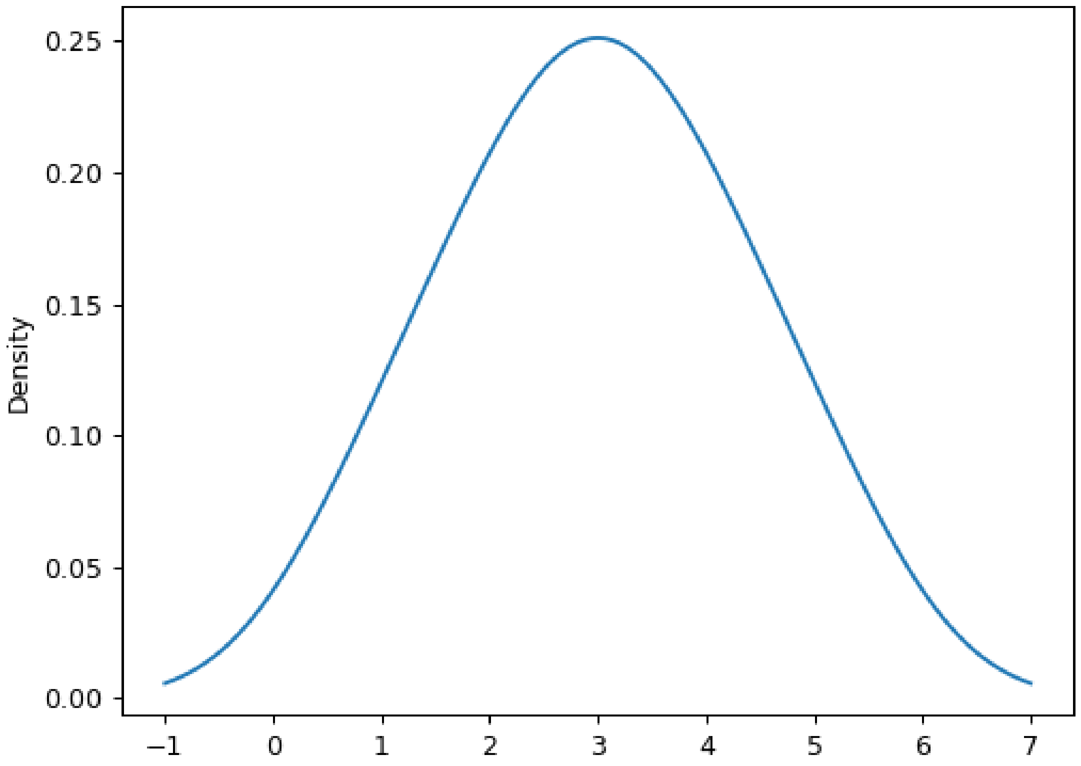
```
>>> ax = s.plot.kde(bw_method=3)
```

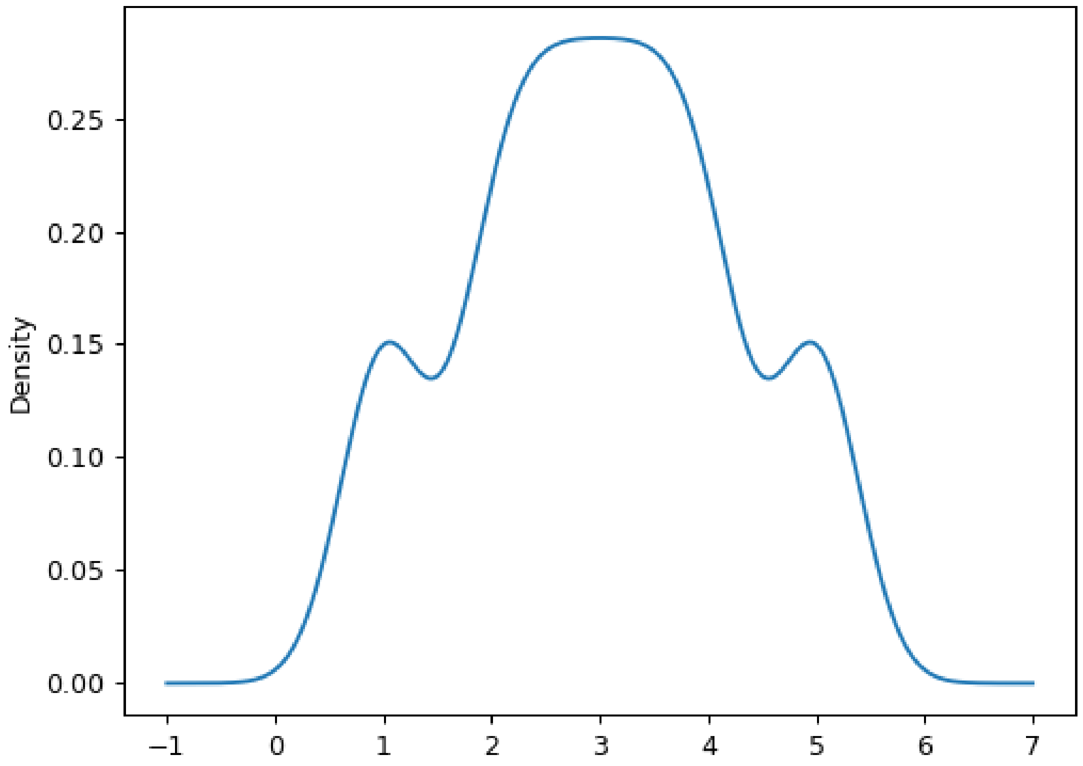
Finally, the *ind* parameter determines the evaluation points for the plot of the estimated PDF:

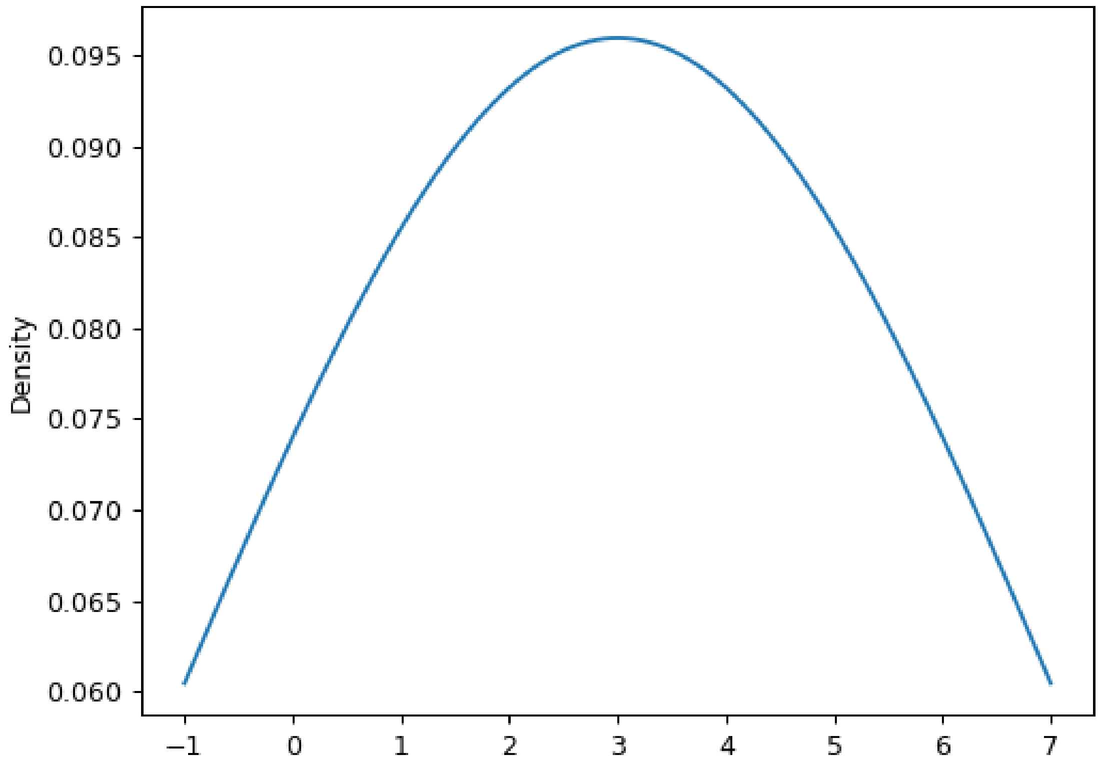
```
>>> ax = s.plot.kde(ind=[1, 2, 3, 4, 5])
```

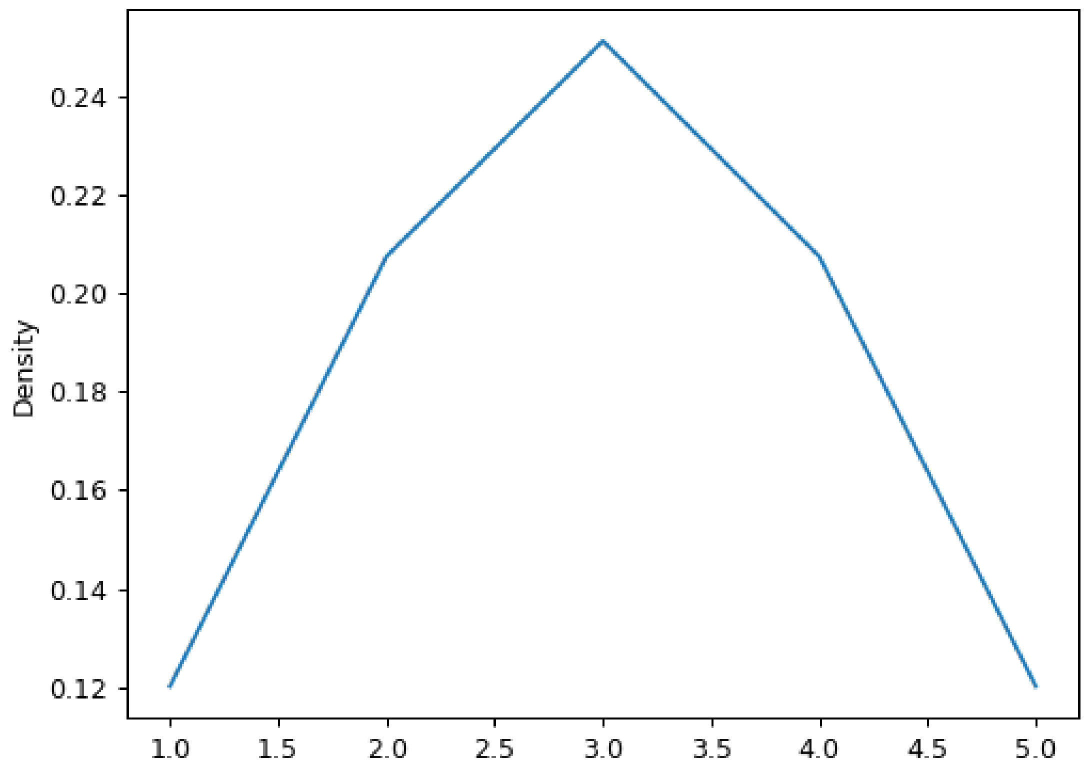
For DataFrame, it works in the same way:

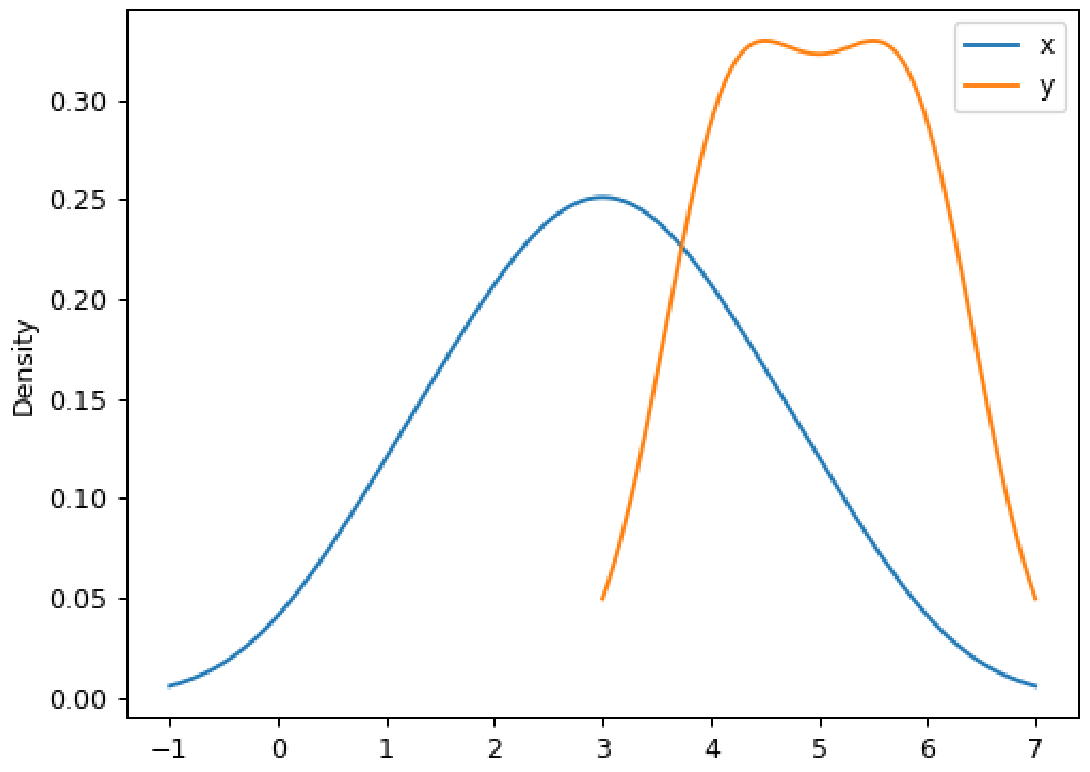
```
>>> df = pd.DataFrame({
...     'x': [1, 2, 2.5, 3, 3.5, 4, 5],
...     'y': [4, 4, 4.5, 5, 5.5, 6, 6],
... })
>>> ax = df.plot.kde()
```


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


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

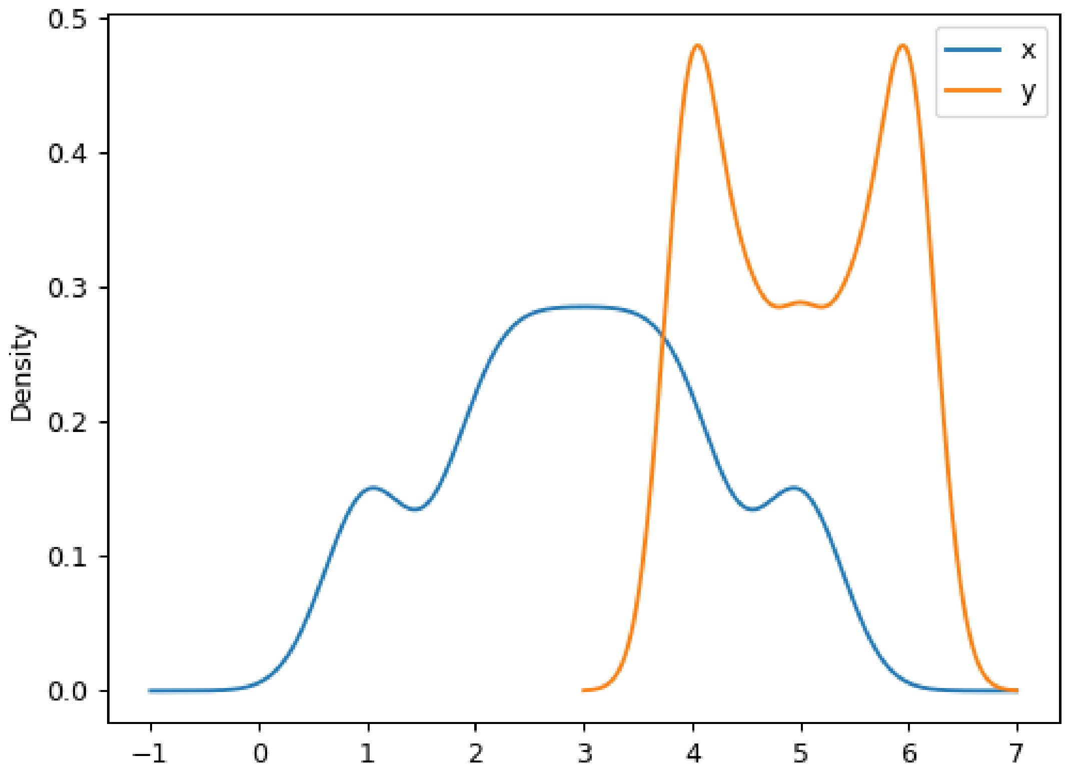

Density
0.095
0.090
0.085
0.080
0.075
0.070
0.065
0.060
−1
0
1
2
3
4
5
6
7


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
0.00
0.05
0.10
0.15
0.20
0.25
0.30
−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.DataFrame.plot.line

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

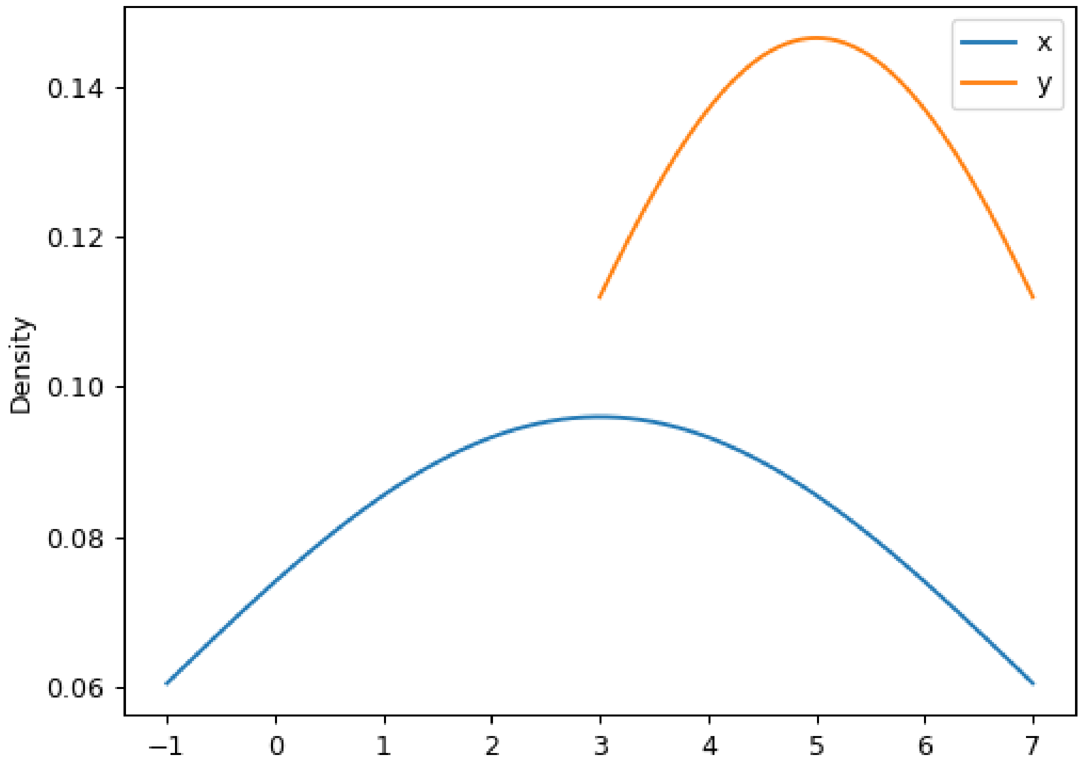
Plot Series or DataFrame as lines.

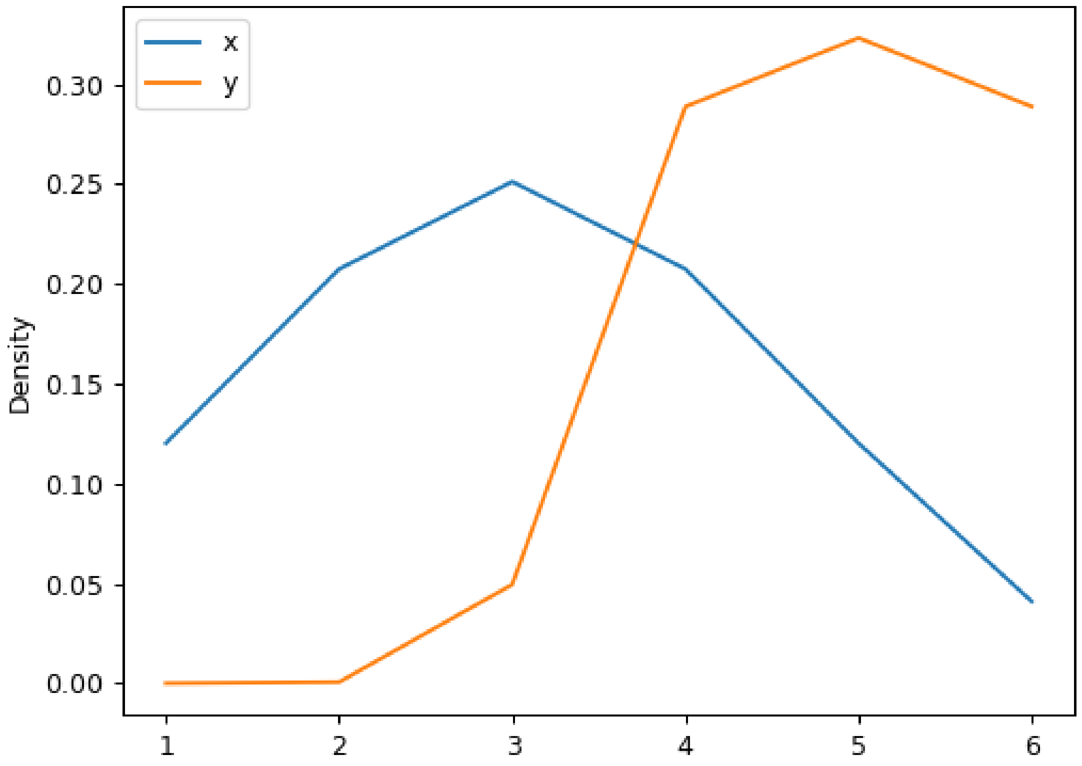
This function is useful to plot lines using DataFrame's values as coordinates.

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


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

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

Returns

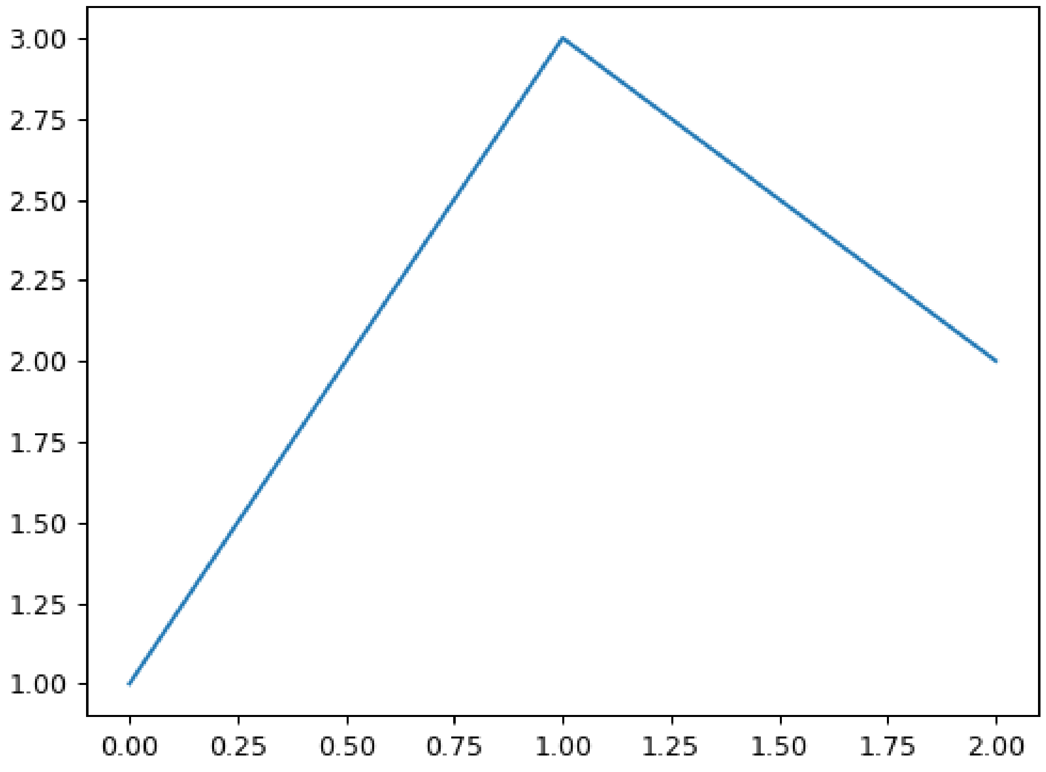
`matplotlib.axes.Axes` or `numpy.ndarray` Return an ndarray when `subplots=True`.

See also:

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

Examples

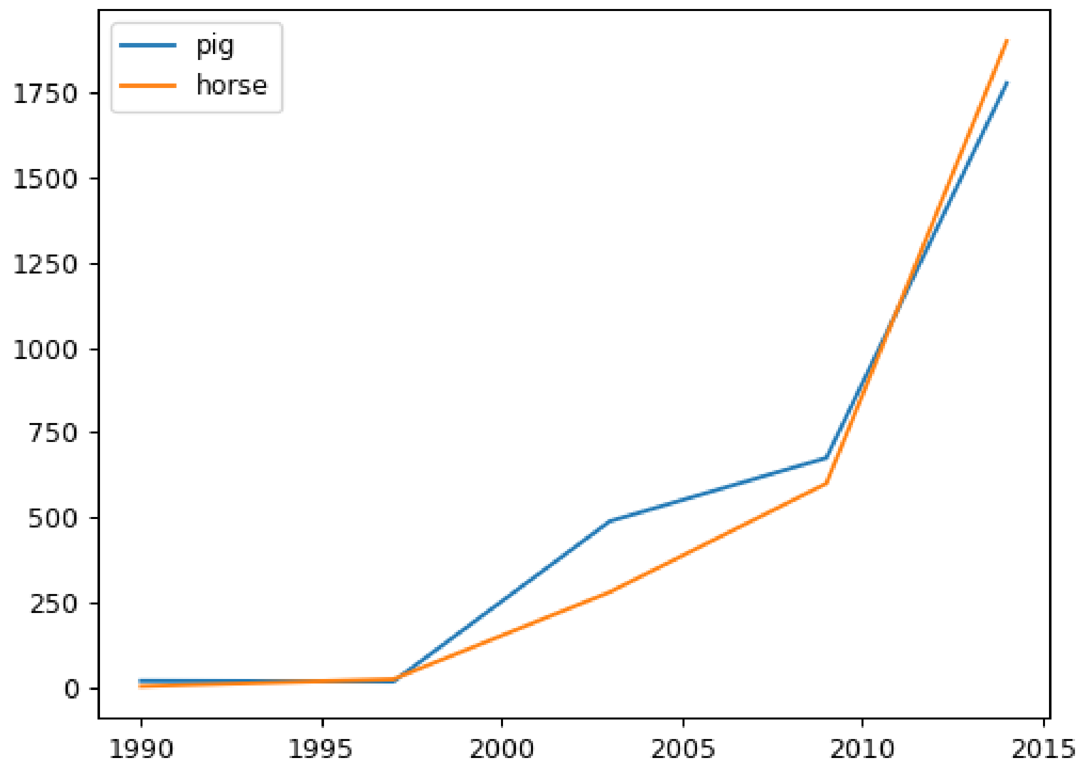
```
>>> s = pd.Series([1, 3, 2])
>>> s.plot.line()
```



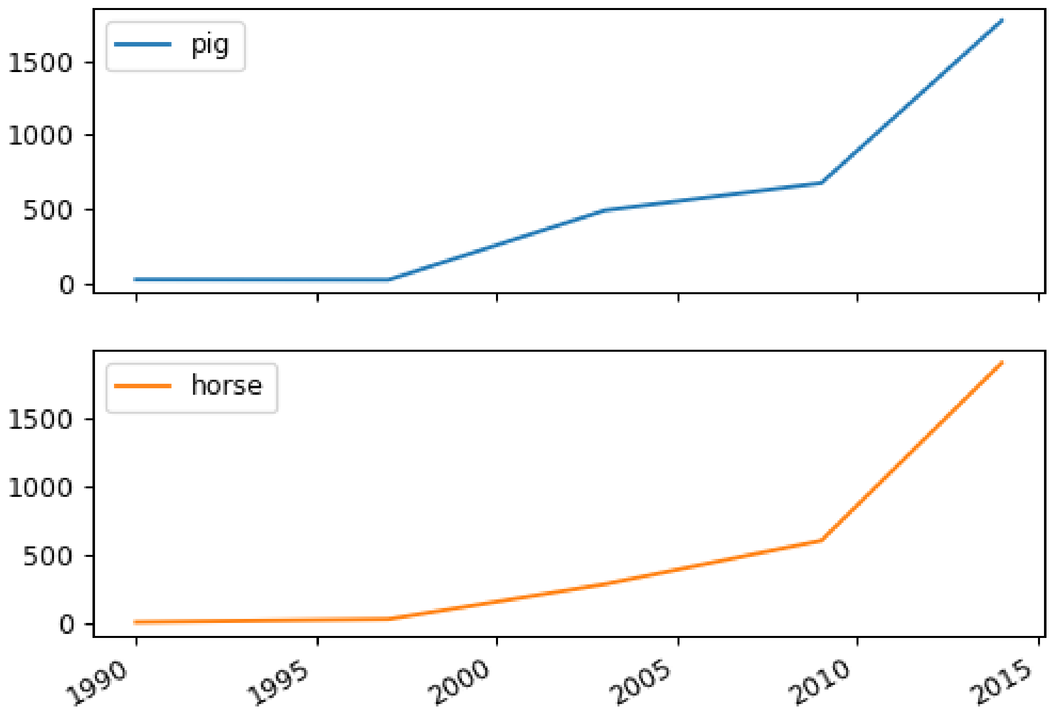
The following example shows the populations for some animals over the years.

```
>>> 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.DataFrame.plot.pie

`DataFrame.plot.pie(self, **kwargs)`
Generate a pie plot.

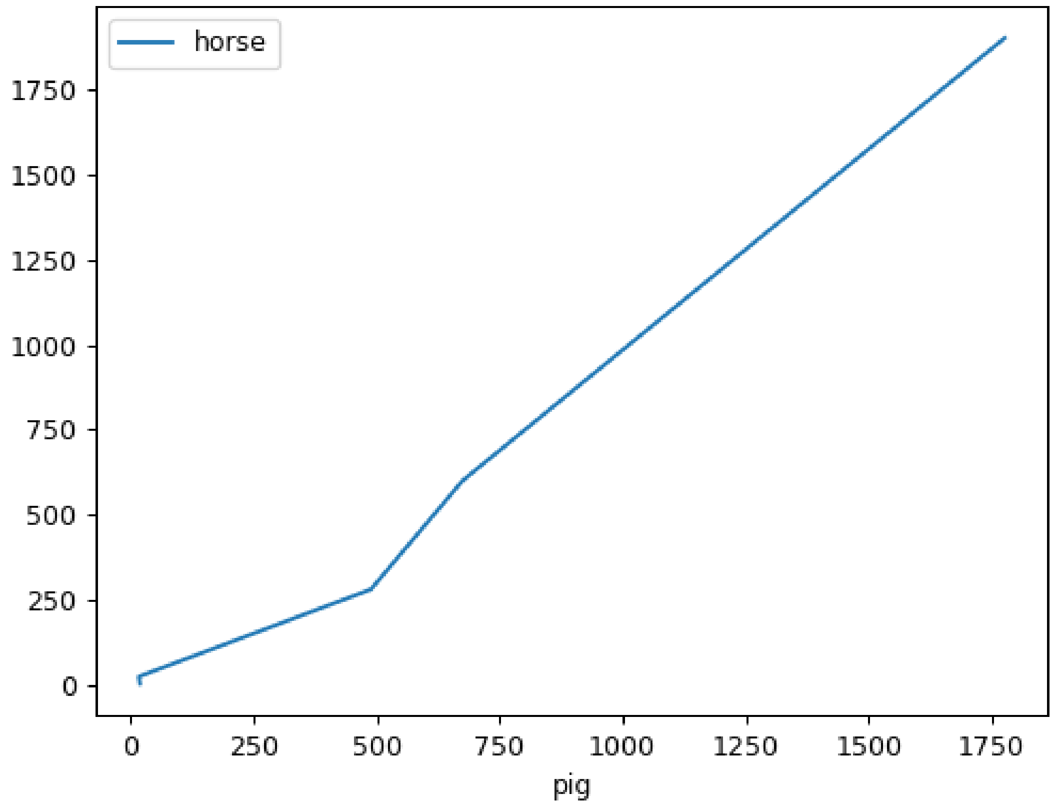
A pie plot is a proportional representation of the numerical data in a column. This function wraps `matplotlib.pyplot.pie()` for the specified column. If no column reference is passed and `subplots=True` a pie plot is drawn for each numerical column independently.

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
0
250
500
750
1000
1250
1500
1750
pig

matplotlib.axes.Axes or np.ndarray of them A NumPy array is returned when *subplots* is True.

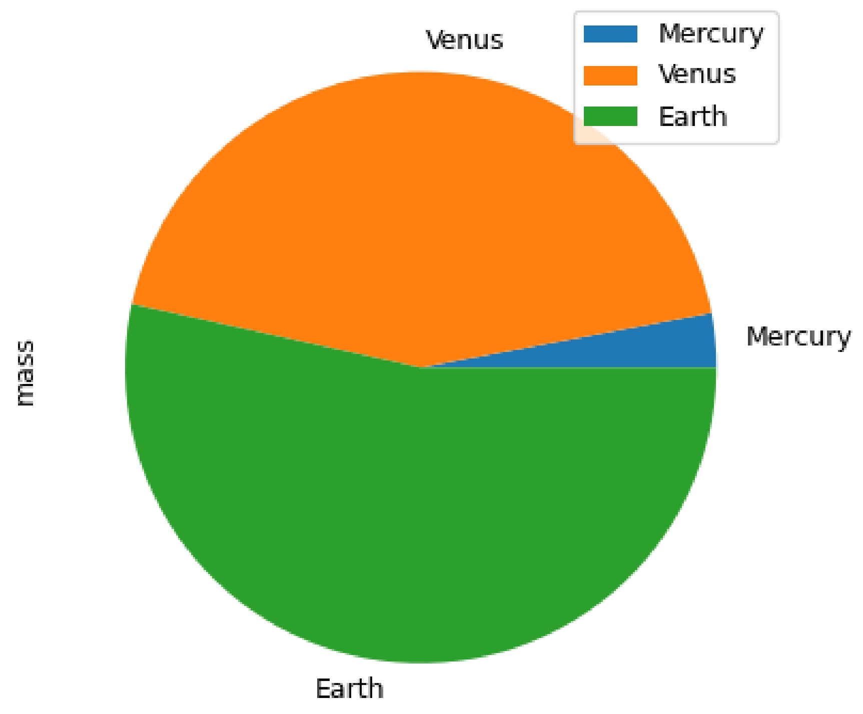
See also:

`Series.plot.pie` Generate a pie plot for a Series.
`DataFrame.plot` Make plots of a DataFrame.

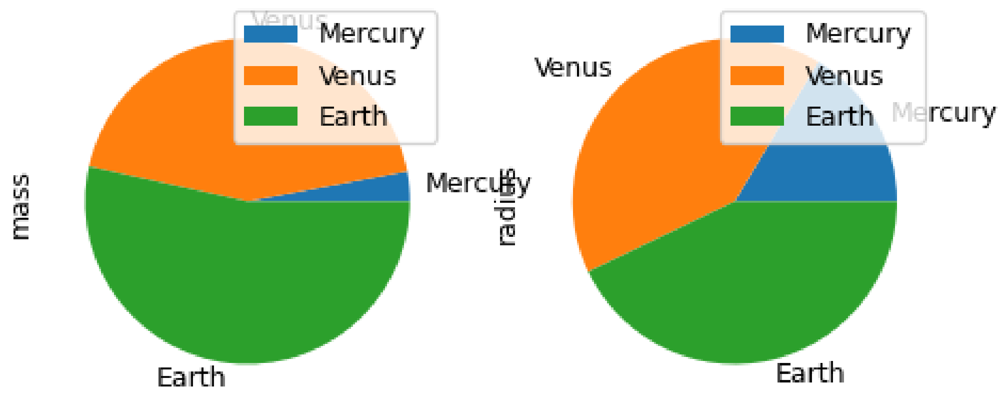
Examples

In the example below we have a DataFrame with the information about planet's mass and radius. We pass the the 'mass' column to the pie function to get a pie plot.

```
>>> df = pd.DataFrame({'mass': [0.330, 4.87 , 5.97],
...                    'radius': [2439.7, 6051.8, 6378.1]},
...                   index=['Mercury', 'Venus', 'Earth'])
>>> plot = df.plot.pie(y='mass', figsize=(5, 5))
```



```
>>> plot = df.plot.pie(subplots=True, figsize=(6, 3))
```



pandas.DataFrame.plot.scatter

`DataFrame.plot.scatter`(*self*, *x*, *y*, *s=None*, *c=None*, ***kwargs*)

Create a scatter plot with varying marker point size and color.

The coordinates of each point are defined by two dataframe columns and filled circles are used to represent each point. This kind of plot is useful to see complex correlations between two variables. Points could be for instance natural 2D coordinates like longitude and latitude in a map or, in general, any pair of metrics that can be plotted against each other.

Parameters

x [int or str] The column name or column position to be used as horizontal coordinates for each point.

y [int or str] The column name or column position to be used as vertical coordinates for each point.

s [scalar or array_like, optional] The size of each point. Possible values are:

- A single scalar so all points have the same size.
- A sequence of scalars, which will be used for each point's size recursively. For instance, when passing [2,14] all points size will be either 2 or 14, alternatively.

c [str, int or array_like, optional] The color of each point. Possible values are:

- A single color string referred to by name, RGB or RGBA code, for instance 'red' or '#a98d19'.
- A sequence of color strings referred to by name, RGB or RGBA code, which will be used for each point's color recursively. For instance ['green','yellow'] all points will be filled in green or yellow, alternatively.
- A column name or position whose values will be used to color the marker points according to a colormap.

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

Returns

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

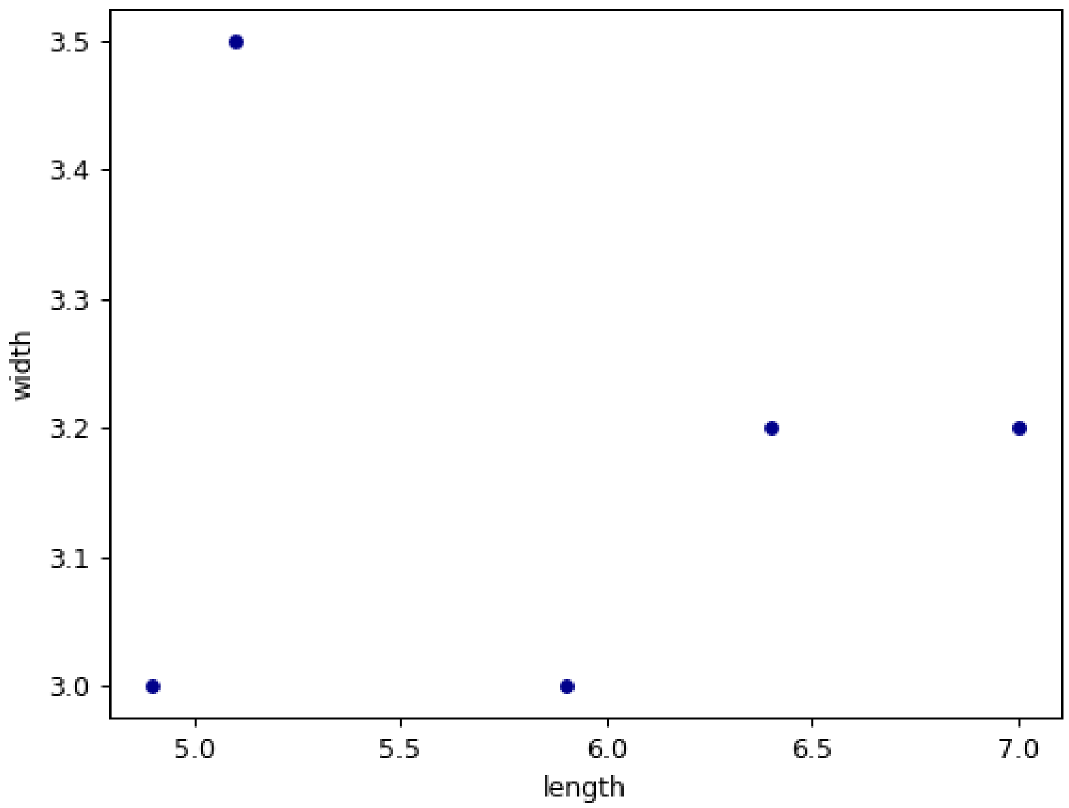
See also:

`matplotlib.pyplot.scatter` Scatter plot using multiple input data formats.

Examples

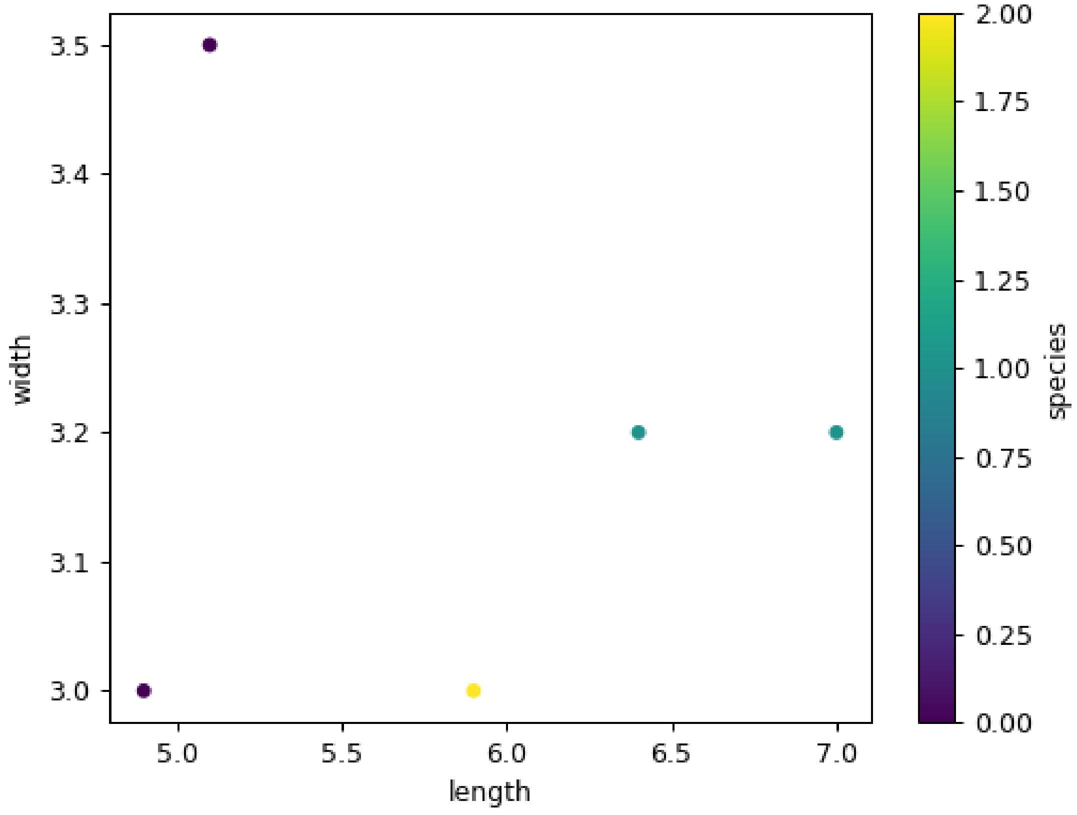
Let's see how to draw a scatter plot using coordinates from the values in a DataFrame's columns.

```
>>> df = pd.DataFrame([[5.1, 3.5, 0], [4.9, 3.0, 0], [7.0, 3.2, 1],
...                    [6.4, 3.2, 1], [5.9, 3.0, 2]],
...                   columns=['length', 'width', 'species'])
>>> ax1 = df.plot.scatter(x='length',
...                       y='width',
...                       c='DarkBlue')
```



And now with the color determined by a column as well.

```
>>> ax2 = df.plot.scatter(x='length',
...                       y='width',
...                       c='species',
...                       colormap='viridis')
```


width
3.5
3.4
3.3
3.2
3.1
3.0
5.0
5.5
6.0
6.5
7.0
length
species
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

DataFrame.boxplot(self[, column, by, ax, . . .])	Make a box plot from DataFrame columns.
DataFrame.hist(data[, column, by, grid, . . .])	Make a histogram of the DataFrame's.

3.4.15 Sparse accessor

Sparse-dtype specific methods and attributes are provided under the `DataFrame.sparse` accessor.

DataFrame.sparse.density	Ratio of non-sparse points to total (dense) data points.

pandas.DataFrame.sparse.density

`DataFrame.sparse.`**`density`**

Ratio of non-sparse points to total (dense) data points.

DataFrame.sparse.from_spmatrix(data[, . . .])	Create a new DataFrame from a scipy sparse matrix.
DataFrame.sparse.to_coo(self)	Return the contents of the frame as a sparse SciPy COO matrix.
DataFrame.sparse.to_dense(self)	Convert a DataFrame with sparse values to dense.

pandas.DataFrame.sparse.from_spmatrix

classmethod `DataFrame.sparse.`**`from_spmatrix`**(*data*, *index=None*, *columns=None*)

Create a new DataFrame from a scipy sparse matrix.

New in version 0.25.0.

Parameters

data [scipy.sparse.spmatrix] Must be convertible to csc format.

index, columns [Index, optional] Row and column labels to use for the resulting DataFrame. Defaults to a RangeIndex.

Returns

DataFrame Each column of the DataFrame is stored as a *arrays.SparseArray*.

Examples

```
>>> import scipy.sparse
>>> mat = scipy.sparse.eye(3)
>>> pd.DataFrame.sparse.from_spmatrix(mat)
     0    1    2
0  1.0  0.0  0.0
1  0.0  1.0  0.0
2  0.0  0.0  1.0
```

pandas.DataFrame.sparse.to_coo

`DataFrame.sparse.to_coo(self)`

Return the contents of the frame as a sparse SciPy COO matrix.

New in version 0.25.0.

Returns

coo_matrix [scipy.sparse.spmatrix] If the caller is heterogeneous and contains booleans or objects, the result will be of dtype=object. See Notes.

Notes

The dtype will be the lowest-common-denominator type (implicit upcasting); that is to say if the dtypes (even of numeric types) are mixed, the one that accommodates all will be chosen.

e.g. If the dtypes are float16 and float32, dtype will be upcast to float32. By numpy.find_common_type convention, mixing int64 and and uint64 will result in a float64 dtype.

pandas.DataFrame.sparse.to_dense

`DataFrame.sparse.to_dense(self)`

Convert a DataFrame with sparse values to dense.

New in version 0.25.0.

Returns

DataFrame A DataFrame with the same values stored as dense arrays.

Examples

```
>>> df = pd.DataFrame({"A": pd.arrays.SparseArray([0, 1, 0])})
>>> df.sparse.to_dense()
   A
0  0
1  1
2  0
```

3.4.16 Serialization / IO / conversion

`DataFrame.from_dict`(data[, orient, dtype, ...])	Construct DataFrame from dict of array-like or dicts.
`DataFrame.from_records`(data[, index, ...])	Convert structured or record ndarray to DataFrame.
`DataFrame.info`(self[, verbose, buf, ...])	Print a concise summary of a DataFrame.
`DataFrame.to_parquet`(self, path[, engine, ...])	Write a DataFrame to the binary parquet format.
`DataFrame.to_pickle`(self, path, compression, ...)	Pickle (serialize) object to file.
`DataFrame.to_csv`(self, path_or_buf, ...)	Write object to a comma-separated values (csv) file.
`DataFrame.to_hdf`(self, path_or_buf, key, ...)	Write the contained data to an HDF5 file using HDFStore.
`DataFrame.to_sql`(self, name, con[, schema, ...])	Write records stored in a DataFrame to a SQL database.
`DataFrame.to_dict`(self[, orient, into])	Convert the DataFrame to a dictionary.

continues on next page

Table 78 – continued from previous page

`DataFrame.to_excel`(self, excel_writer[, . . .])	Write object to an Excel sheet.
`DataFrame.to_json`(self, path_or_buf, . . .)	Convert the object to a JSON string.
`DataFrame.to_html`(self[, buf, columns, . . .])	Render a DataFrame as an HTML table.
`DataFrame.to_feather`(self, path)	Write out the binary feather-format for DataFrames.
`DataFrame.to_latex`(self[, buf, columns, . . .])	Render object to a LaTeX tabular, longtable, or nested table/tabular.
`DataFrame.to_stata`(self, path[, . . .])	Export DataFrame object to Stata dta format.
`DataFrame.to_gbq`(self, destination_table[, . . .])	Write a DataFrame to a Google BigQuery table.
`DataFrame.to_records`(self[, index, . . .])	Convert DataFrame to a NumPy record array.
`DataFrame.to_string`(self, buf, pathlib.Path, . . .)	Render a DataFrame to a console-friendly tabular output.
`DataFrame.to_clipboard`(self, excel, sep, . . .)	Copy object to the system clipboard.
`DataFrame.to_markdown`(self, buf, . . .)	Print DataFrame in Markdown-friendly format.
`DataFrame.style`	Returns a Styler object.

3.5 Pandas arrays

For most data types, pandas uses NumPy arrays as the concrete objects contained with a `Index`, `Series`, or `DataFrame`.

For some data types, pandas extends NumPy's type system. String aliases for these types can be found at *dtypes*.

Kind of Data	Pandas Data Type	Scalar	Array
TZ-aware datetime	`DatetimeTZDtype`	`Timestamp`	*Datetime data*
Timedeltas	(none)	`Timedelta`	*Timedelta data*
Period (time spans)	`PeriodDtype`	`Period`	*Timespan data*
Intervals	`IntervalDtype`	`Interval`	*Interval data*
Nullable Integer	`Int64Dtype`, . . .	(none)	*Nullable integer*
Categorical	`CategoricalDtype`	(none)	*Categorical data*
Sparse	`SparseDtype`	(none)	*Sparse data*
Strings	`StringDtype`	`str`	*Text data*
Boolean (with NA)	`BooleanDtype`	`bool`	*Boolean data with missing values*

Pandas and third-party libraries can extend NumPy's type system (see *Extension types*). The top-level `array()` method can be used to create a new array, which may be stored in a `Series`, `Index`, or as a column in a `DataFrame`.

`array`(data, dtype, numpy.dtype, . . .)	Create an array.

3.5.1 pandas.array

`pandas.`**`array`**(*data: Sequence[object], dtype: Union[str, numpy.dtype, pandas.core.dtypes.base.ExtensionDtype, NoneType] = None, copy: bool = True*) → pandas.core.dtypes.generic.ABCExtensionArray

Create an array.

New in version 0.24.0.

Parameters

data [Sequence of objects] The scalars inside *data* should be instances of the scalar type for *dtype*. It's expected that *data* represents a 1-dimensional array of data.