Table 69 – continued from previous page

DataFrame.reorder_levels(self, order[, axis])	Rearrange index levels using input order.
DataFrame.sort_values(self, by[, axis, ...])	Sort by the values along either axis.
DataFrame.sort_index(self[, axis, level, ...])	Sort object by labels (along an axis).
DataFrame.nlargest(self, n, columns[, keep])	Return the first *n* rows ordered by *columns* in descending order.
DataFrame.nsmallest(self, n, columns[, keep])	Return the first *n* rows ordered by *columns* in ascending order.
DataFrame.swaplevel(self[, i, j, axis])	Swap levels i and j in a MultiIndex on a particular axis.
DataFrame.stack(self[, level, dropna])	Stack the prescribed level(s) from columns to index.
DataFrame.unstack(self[, level, fill_value])	Pivot a level of the (necessarily hierarchical) index labels.
DataFrame.swapaxes(self, axis1, axis2[, copy])	Interchange axes and swap values axes appropriately.
DataFrame.melt(self[, id_vars, value_vars, ...])	Unpivot a DataFrame from wide to long format, optionally leaving identifiers set.
DataFrame.explode(self, column, Tuple])	Transform each element of a list-like to a row, replicating index values.
DataFrame.squeeze(self[, axis])	Squeeze 1 dimensional axis objects into scalars.
DataFrame.to_xarray(self)	Return an xarray object from the pandas object.
DataFrame.T	Transpose index and columns.
DataFrame.transpose(self, *args, copy)	Transpose index and columns.

3.4.11 Combining / joining / merging

DataFrame.append(self, other[, ...])	Append rows of *other* to the end of caller, returning a new object.
DataFrame.assign(self, **kwargs)	Assign new columns to a DataFrame.
DataFrame.join(self, other[, on, how, ...])	Join columns of another DataFrame.
DataFrame.merge(self, right[, how, on, ...])	Merge DataFrame or named Series objects with a database-style join.
DataFrame.update(self, other[, join, ...])	Modify in place using non-NA values from another DataFrame.

3.4.12 Time series-related

DataFrame.asfreq(self, freq[, method, ...])	Convert TimeSeries to specified frequency.
DataFrame.asof(self, where[, subset])	Return the last row(s) without any NaNs before *where*.
DataFrame.shift(self[, periods, freq, axis, ...])	Shift index by desired number of periods with an optional time *freq*.
DataFrame.slice_shift(self, periods[, axis])	Equivalent to *shift* without copying data.
DataFrame.tshift(self, periods[, freq, axis])	Shift the time index, using the index's frequency if available.
DataFrame.first_valid_index(self)	Return index for first non-NA/null value.
DataFrame.last_valid_index(self)	Return index for last non-NA/null value.
DataFrame.resample(self, rule[, axis, ...])	Resample time-series data.
DataFrame.to_period(self[, freq, axis, copy])	Convert DataFrame from DatetimeIndex to PeriodIndex.

continues on next page

Table 71 – continued from previous page

DataFrame.to_timestamp(self[, freq, how, ...])	Cast to DatetimeIndex of timestamps, at *beginning* of period.
DataFrame.tz_convert(self, tz[, axis, level])	Convert tz-aware axis to target time zone.
DataFrame.tz_localize(self, tz[, axis, ...])	Localize tz-naive index of a Series or DataFrame to target time zone.

3.4.13 Metadata

DataFrame.attrs is a dictionary for storing global metadata for this DataFrame.

Warning: DataFrame.attrs is considered experimental and may change without warning.

DataFrame.attrs	Dictionary of global attributes on this object.

3.4.14 Plotting

DataFrame.plot is both a callable method and a namespace attribute for specific plotting methods of the form DataFrame.plot.<kind>.

DataFrame.plot([x, y, kind, ax, ...])	DataFrame plotting accessor and method

DataFrame.plot.area(self[, x, y])	Draw a stacked area plot.
DataFrame.plot.bar(self[, x, y])	Vertical bar plot.
DataFrame.plot.barh(self[, x, y])	Make a horizontal bar plot.
DataFrame.plot.box(self[, by])	Make a box plot of the DataFrame columns.
DataFrame.plot.density(self[, bw_method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
DataFrame.plot.hexbin(self, x, y[, C, ...])	Generate a hexagonal binning plot.
DataFrame.plot.hist(self[, by, bins])	Draw one histogram of the DataFrame's columns.
DataFrame.plot.kde(self[, bw_method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
DataFrame.plot.line(self[, x, y])	Plot Series or DataFrame as lines.
DataFrame.plot.pie(self, **kwargs)	Generate a pie plot.
DataFrame.plot.scatter(self, x, y[, s, c])	Create a scatter plot with varying marker point size and color.

pandas.DataFrame.plot.area

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

Draw a stacked area plot.

An area plot displays quantitative data visually. This function wraps the matplotlib area function.

Parameters

x [label or position, optional] Coordinates for the X axis. By default uses the index.

y [label or position, optional] Column to plot. By default uses all columns.

stacked [bool, default True] Area plots are stacked by default. Set to False to create a unstacked plot.

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

Returns

matplotlib.axes.Axes or numpy.ndarray Area plot, or array of area plots if subplots is True.

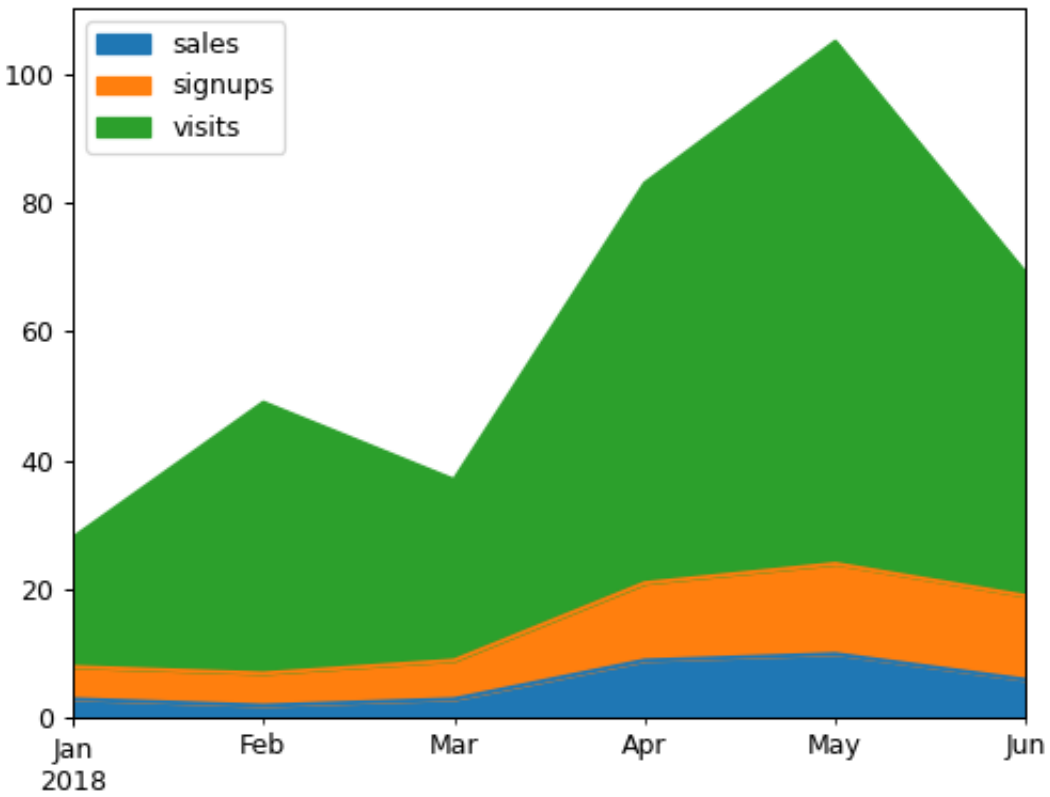
See also:

DataFrame.plot Make plots of DataFrame using matplotlib / pylab.

Examples

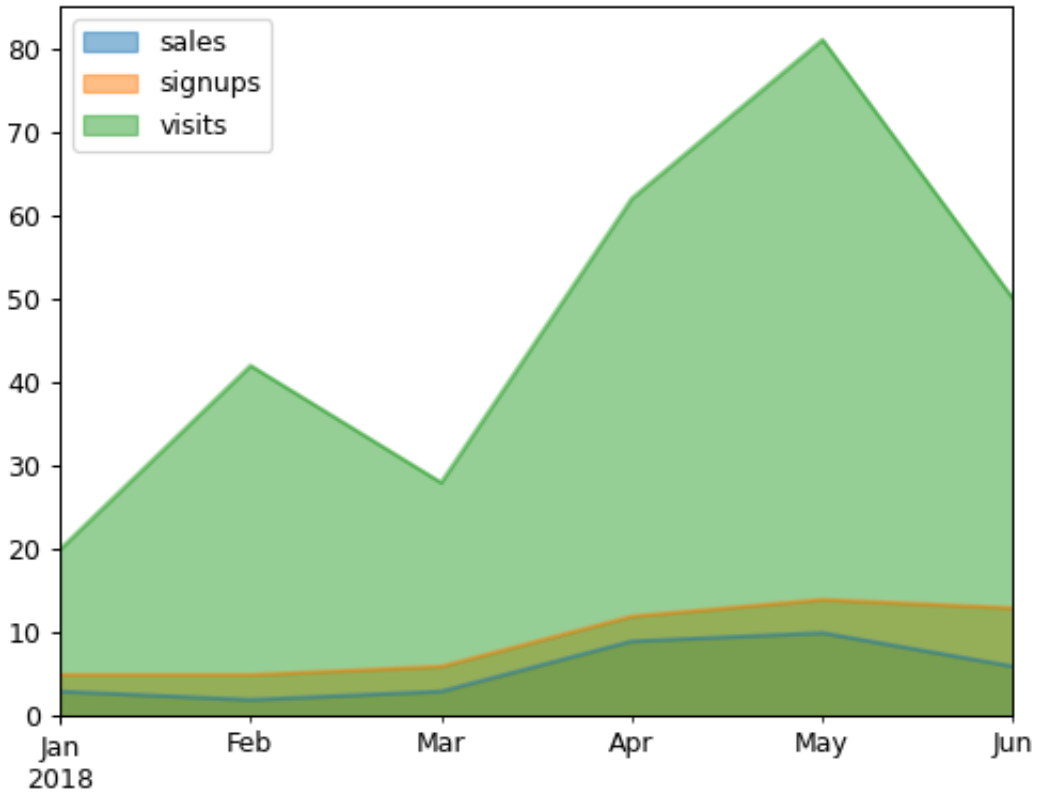
Draw an area plot based on basic business metrics:

```
>>> df = pd.DataFrame({
...     'sales': [3, 2, 3, 9, 10, 6],
...     'signups': [5, 5, 6, 12, 14, 13],
...     'visits': [20, 42, 28, 62, 81, 50],
... }, index=pd.date_range(start='2018/01/01', end='2018/07/01',
...                        freq='M'))
>>> ax = df.plot.area()
```



Area plots are stacked by default. To produce an unstacked plot, pass `stacked=False`:

```
>>> ax = df.plot.area(stacked=False)
```

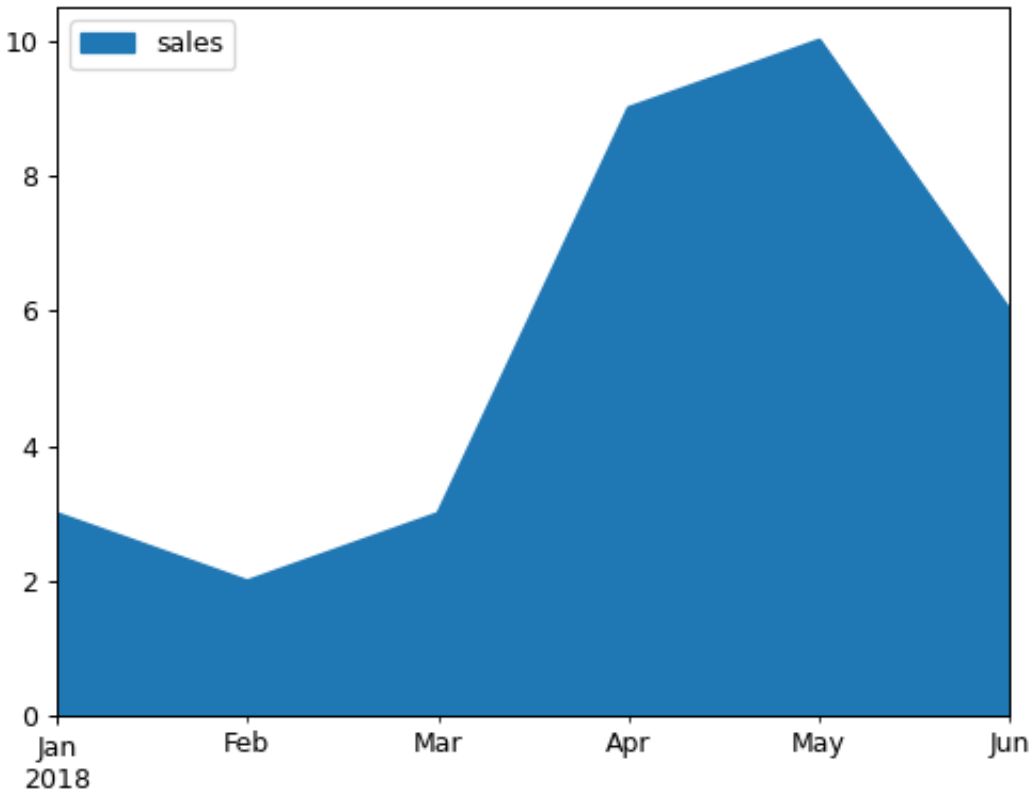


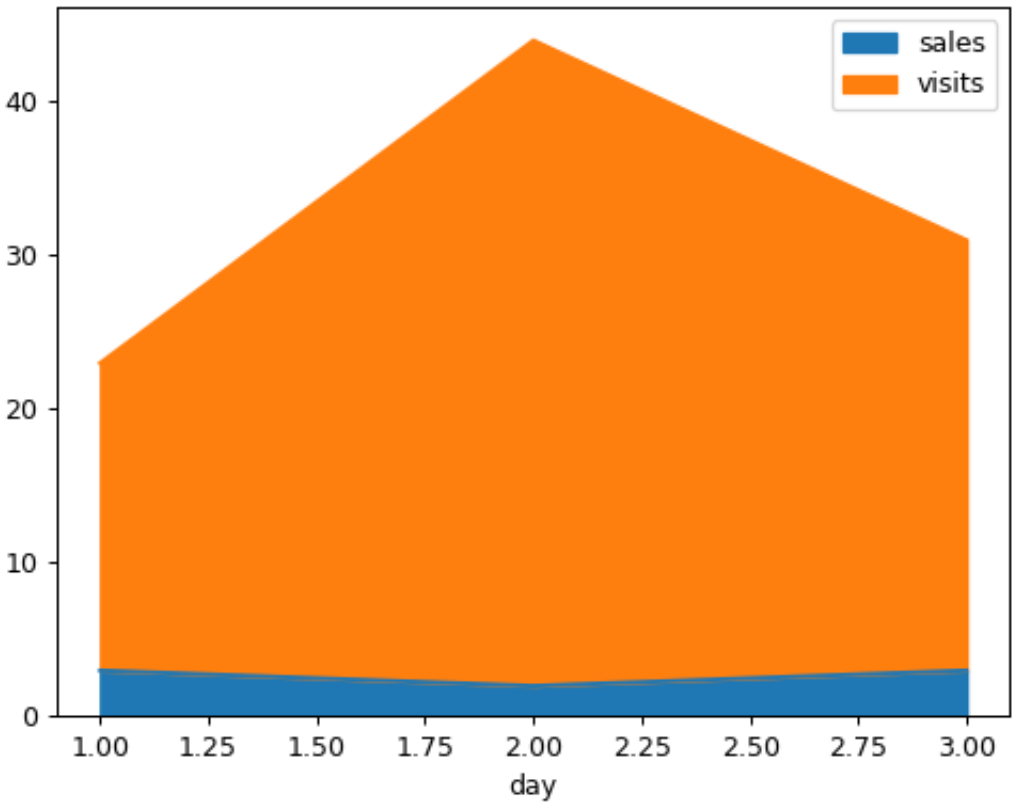
Draw an area plot for a single column:

```
>>> ax = df.plot.area(y='sales')
```

Draw with a different *x*:

```
>>> df = pd.DataFrame({
...     'sales': [3, 2, 3],
...     'visits': [20, 42, 28],
...     'day': [1, 2, 3],
... })
>>> ax = df.plot.area(x='day')
```


sales
10
8
6
4
2
0
Jan
2018
Feb
Mar
Apr
May
Jun


sales
visits
40
30
20
10
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
day

pandas.DataFrame.plot.bar

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

Vertical bar plot.

A bar plot is a plot that presents categorical data with rectangular bars with lengths proportional to the values that they represent. A bar plot shows comparisons among discrete categories. One axis of the plot shows the specific categories being compared, and the other axis represents a measured value.

Parameters

x [label or position, optional] Allows plotting of one column versus another. If not specified, the index of the DataFrame is used.

y [label or position, optional] Allows plotting of one column versus another. If not specified, all numerical columns are used.

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

Returns

matplotlib.axes.Axes or np.ndarray of them An ndarray is returned with one matplotlib.axes.Axes per column when subplots=True.

See also:

DataFrame.plot.barh Horizontal bar plot.
DataFrame.plot Make plots of a DataFrame.
matplotlib.pyplot.bar Make a bar plot with matplotlib.

Examples

Basic plot.

```
>>> df = pd.DataFrame({'lab':['A', 'B', 'C'], 'val':[10, 30, 20]})
>>> ax = df.plot.bar(x='lab', y='val', rot=0)
```

Plot a whole dataframe to a bar plot. Each column is assigned a distinct color, and each row is nested in a group along the horizontal axis.

```
>>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
>>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
>>> index = ['snail', 'pig', 'elephant',
...          'rabbit', 'giraffe', 'coyote', 'horse']
>>> df = pd.DataFrame({'speed': speed,
...                    'lifespan': lifespan}, index=index)
>>> ax = df.plot.bar(rot=0)
```

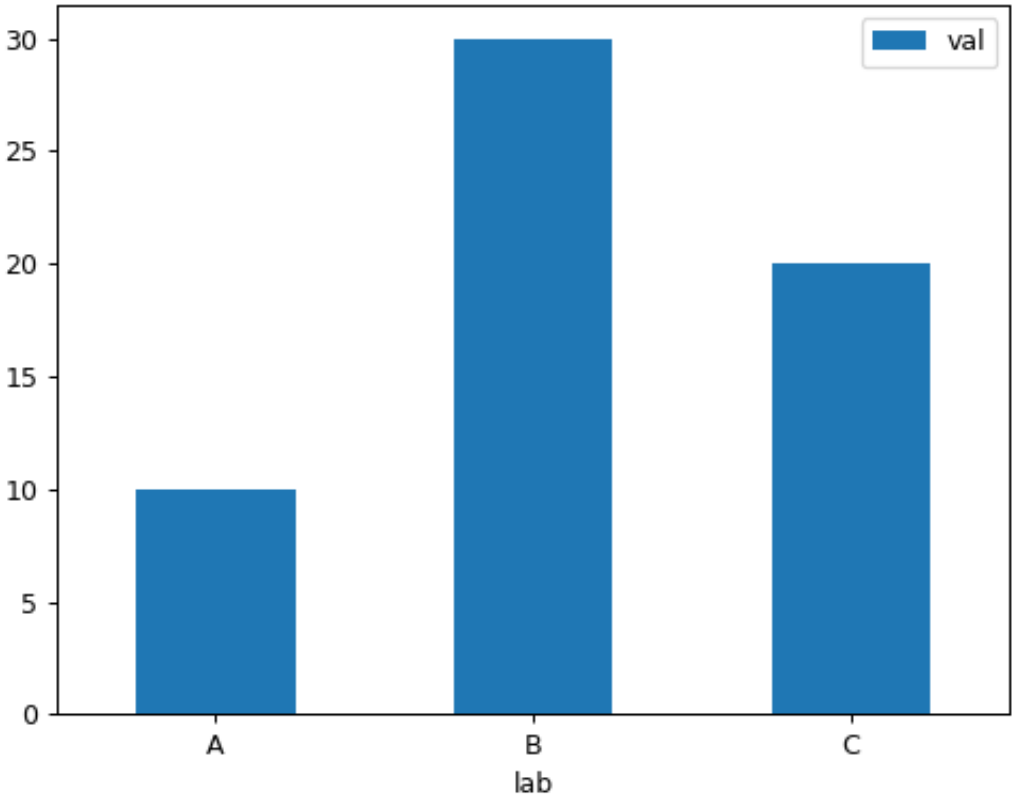
Instead of nesting, the figure can be split by column with subplots=True. In this case, a numpy.ndarray of matplotlib.axes.Axes are returned.

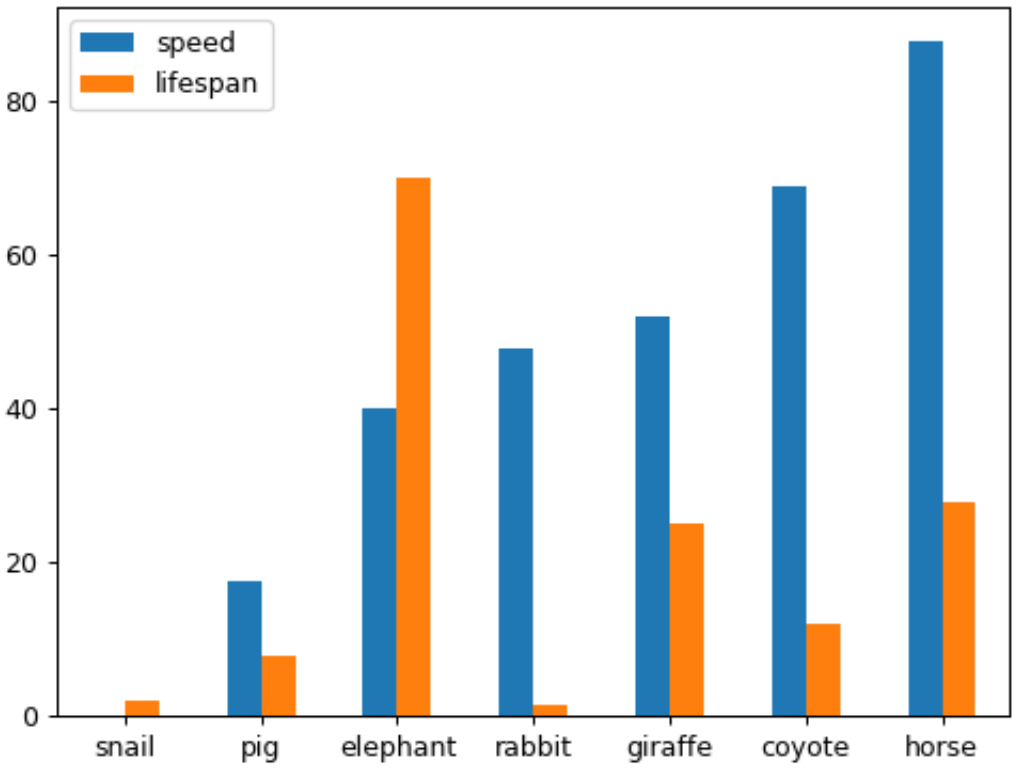
```
>>> axes = df.plot.bar(rot=0, subplots=True)
>>> axes[1].legend(loc=2)
```

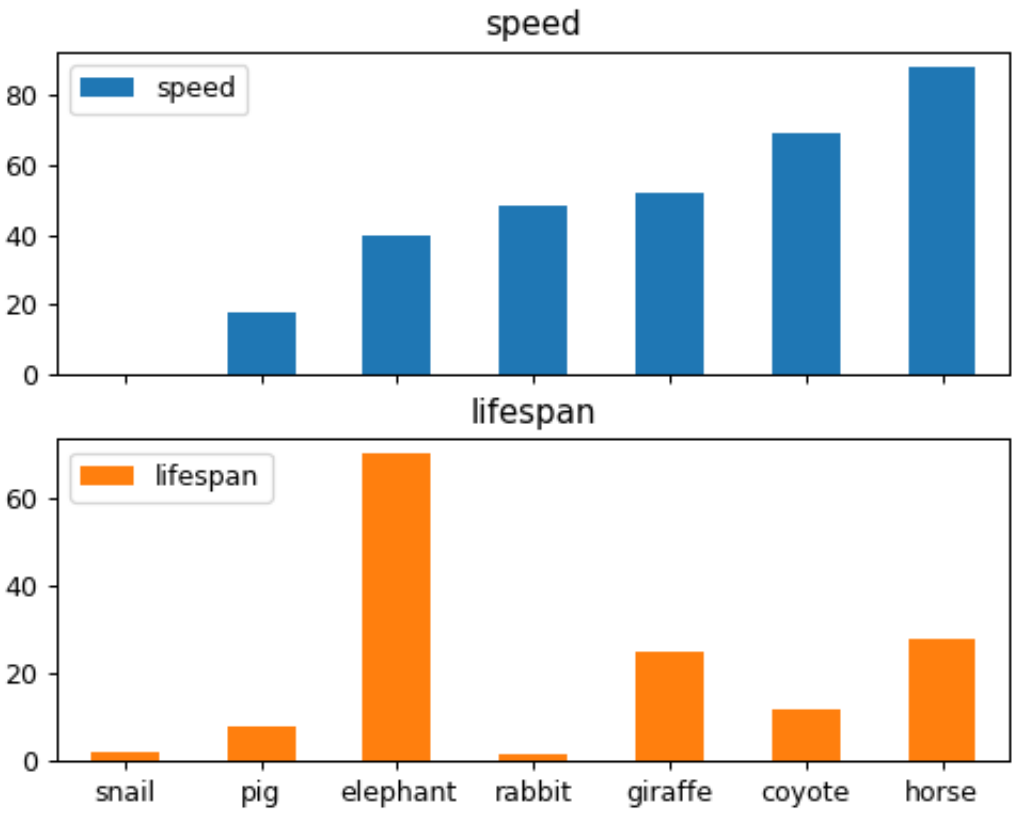
Plot a single column.

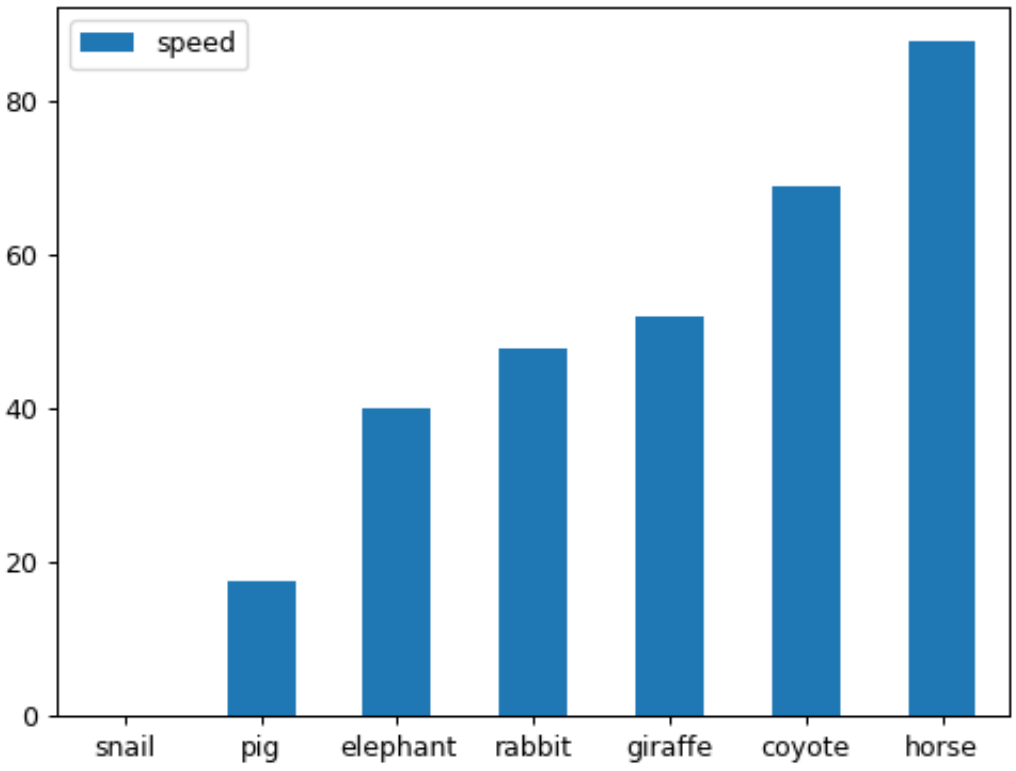
```
>>> ax = df.plot.bar(y='speed', rot=0)
```

Plot only selected categories for the DataFrame.

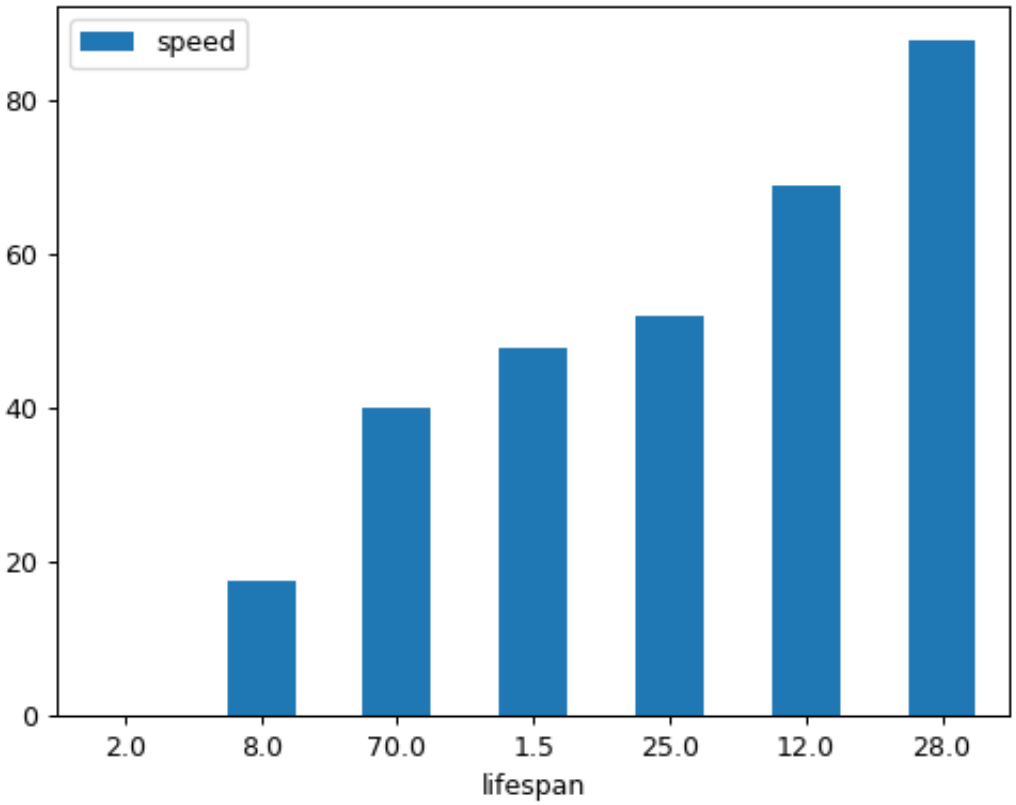

val
30
25
20
15
10
5
0
A
B
C
lab


speed
lifespan
80
60
40
20
0
snail
pig
elephant
rabbit
giraffe
coyote
horse


speed
speed
80
60
40
20
0
lifespan
lifespan
60
40
20
0
snail
pig
elephant
rabbit
giraffe
coyote
horse


speed
80
60
40
20
0
snail
pig
elephant
rabbit
giraffe
coyote
horse

```
>>> ax = df.plot.bar(x='lifespan', rot=0)
```



pandas.DataFrame.plot.barh

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

Make a horizontal bar plot.

A horizontal bar plot is a plot that presents quantitative data with rectangular bars with lengths proportional to the values that they represent. A bar plot shows comparisons among discrete categories. One axis of the plot shows the specific categories being compared, and the other axis represents a measured value.

Parameters

x [label or position, default DataFrame.index] Column to be used for categories.

y [label or position, default All numeric columns in dataframe] Columns to be plotted from the DataFrame.

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

Returns

matplotlib.axes.Axes or numpy.ndarray of them

See also:

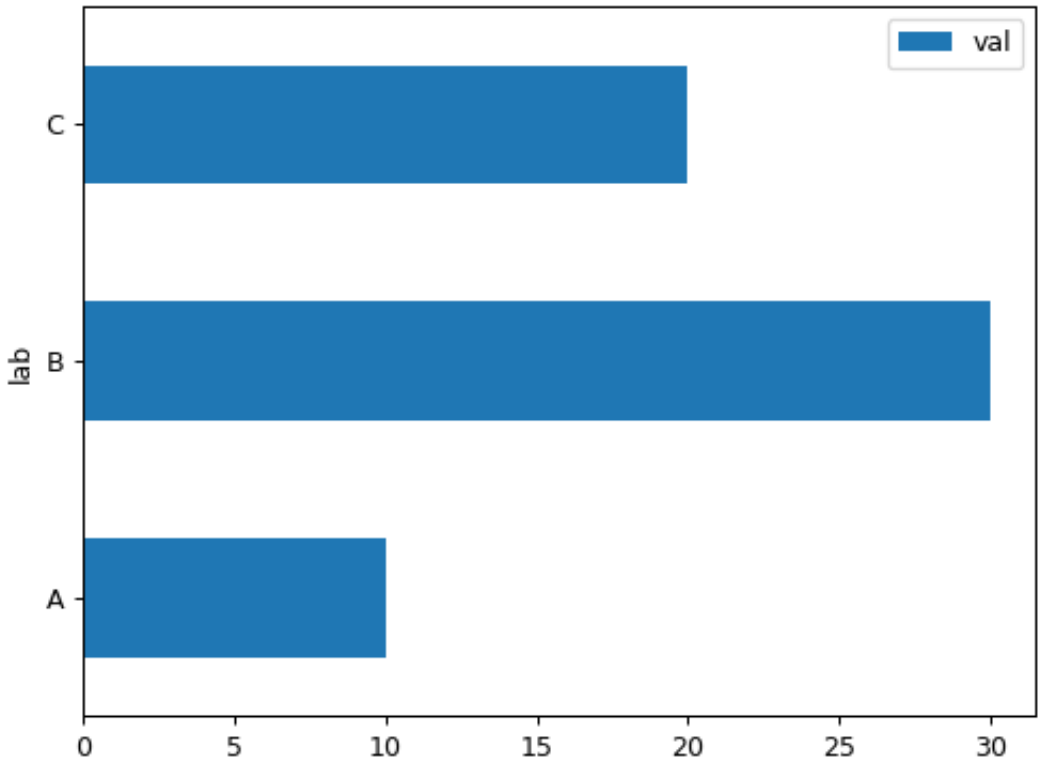
DataFrame.plot.bar Vertical bar plot.

DataFrame.plot Make plots of DataFrame using matplotlib.
matplotlib.axes.Axes.bar Plot a vertical bar plot using matplotlib.

Examples

Basic example

```
>>> df = pd.DataFrame({'lab': ['A', 'B', 'C'], 'val': [10, 30, 20]})
>>> ax = df.plot.barh(x='lab', y='val')
```



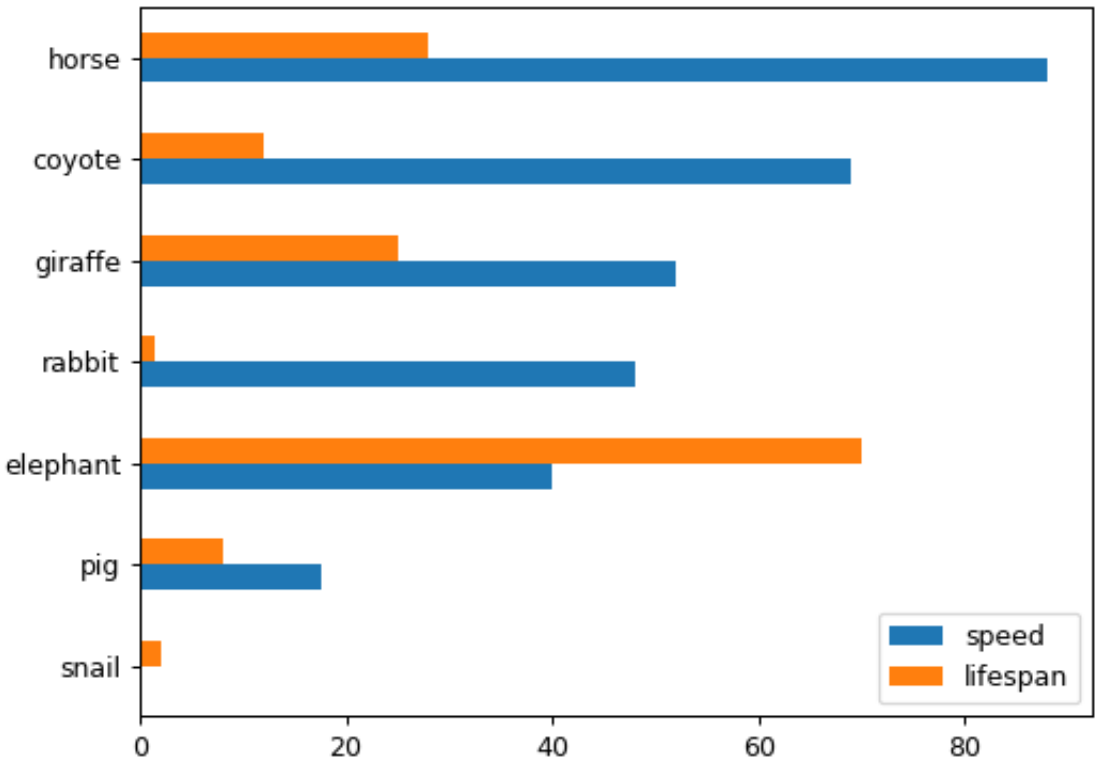
Plot a whole DataFrame to a horizontal bar plot

```
>>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
>>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
>>> index = ['snail', 'pig', 'elephant',
...          'rabbit', 'giraffe', 'coyote', 'horse']
>>> df = pd.DataFrame({'speed': speed,
...                    'lifespan': lifespan}, index=index)
>>> ax = df.plot.barh()
```

Plot a column of the DataFrame to a horizontal bar plot

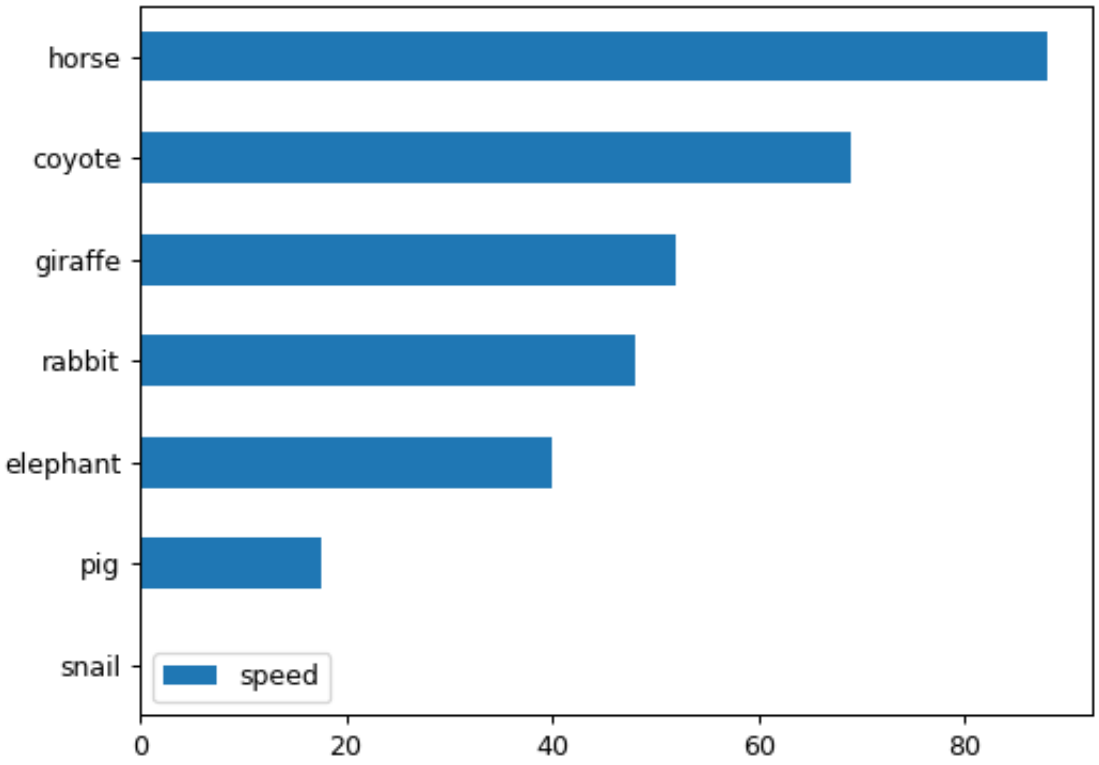
```
>>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
>>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
```

(continues on next page)


horse
coyote
giraffe
rabbit
elephant
pig
snail
0
20
40
60
80
speed
lifespan

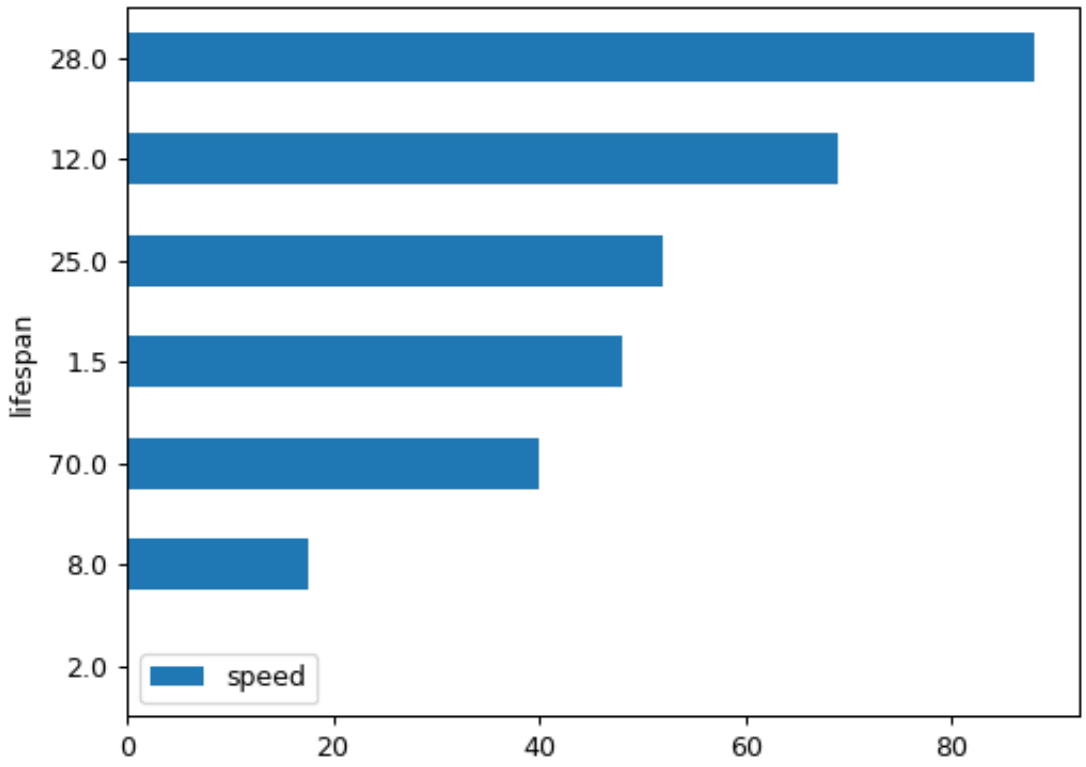
(continued from previous page)

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



Plot DataFrame versus the desired column

```
>>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
>>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
>>> index = ['snail', 'pig', 'elephant',
...          'rabbit', 'giraffe', 'coyote', 'horse']
>>> df = pd.DataFrame({'speed': speed,
...                    'lifespan': lifespan}, index=index)
>>> ax = df.plot.barh(x='lifespan')
```


28.0
12.0
25.0
1.5
70.0
8.0
2.0
lifespan
speed
0
20
40
60
80

pandas.DataFrame.plot.box

DataFrame.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.DataFrame.plot.density

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

Generate Kernel Density Estimate plot using Gaussian kernels.

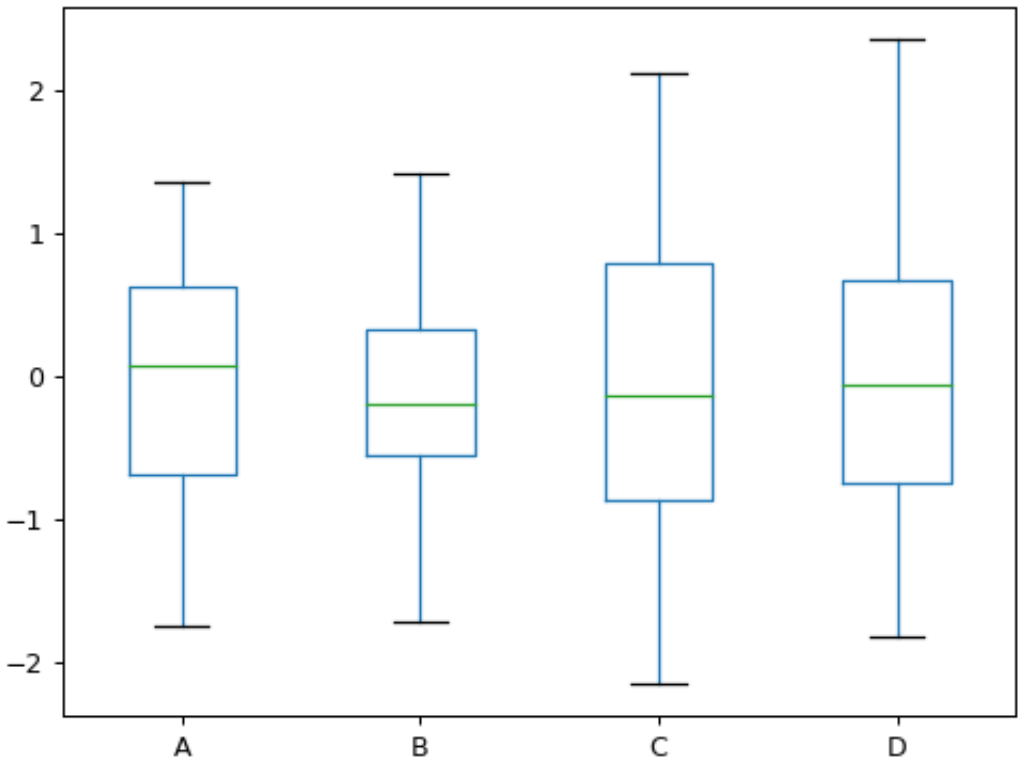
In statistics, kernel density estimation (KDE) is a non-parametric way to estimate the probability density function (PDF) of a random variable. This function uses Gaussian kernels and includes automatic bandwidth determination.

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().


2
1
0
−1
−2
A
B
C
D

Returns

matplotlib.axes.Axes or numpy.ndarray of them

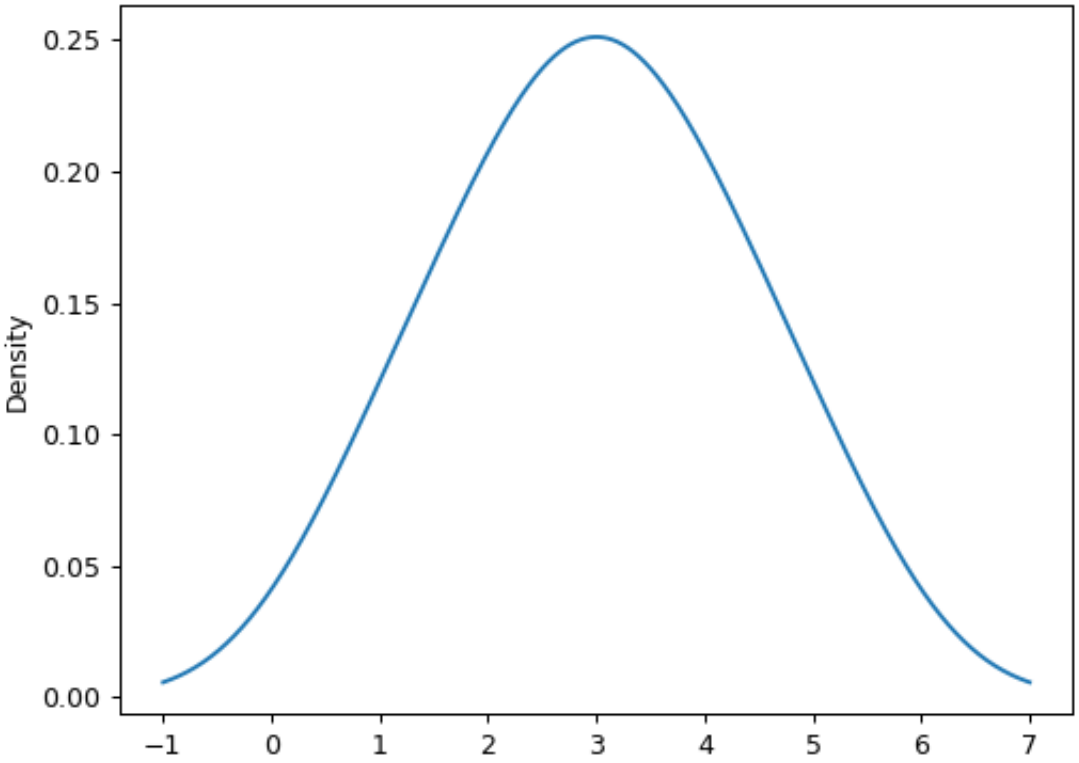
See also:

`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()
```

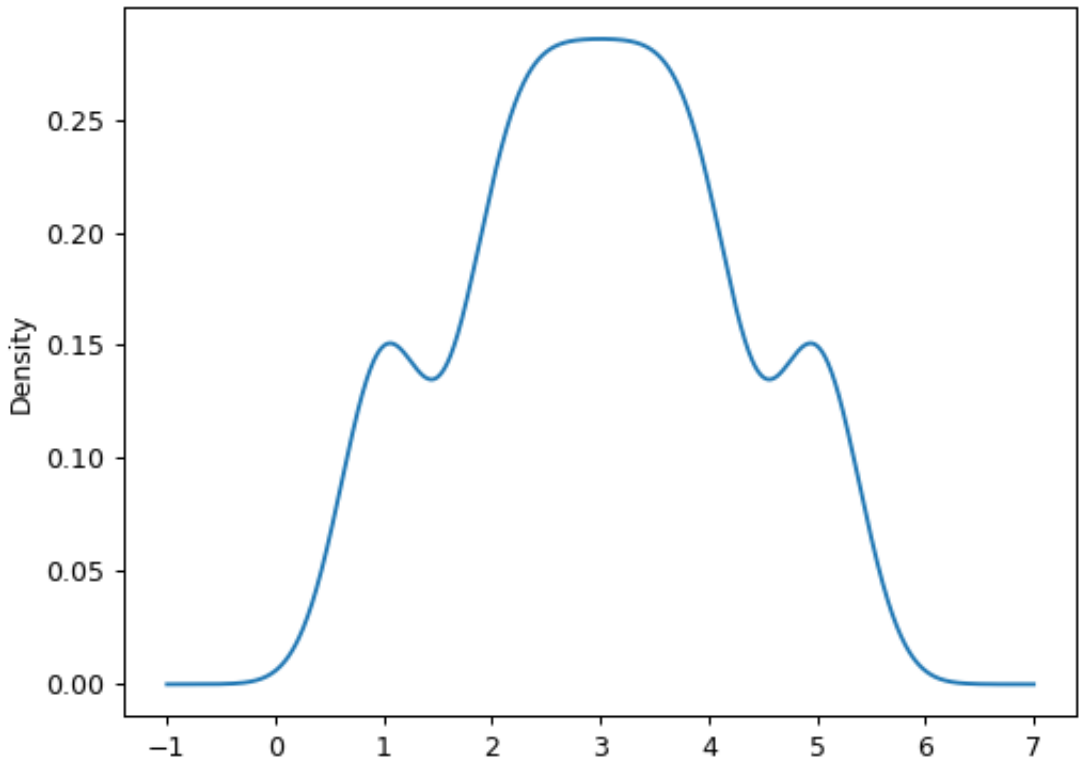


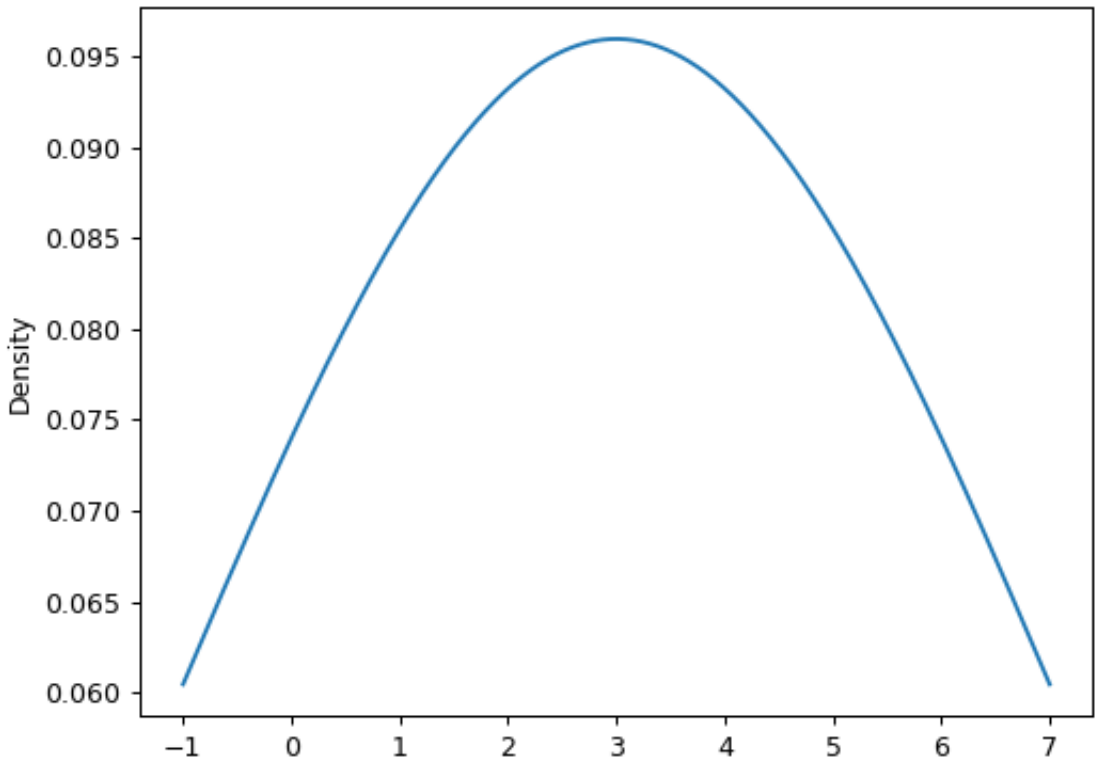
A scalar bandwidth can be specified. Using a small bandwidth value can lead to over-fitting, while using a large bandwidth value may result in under-fitting:

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

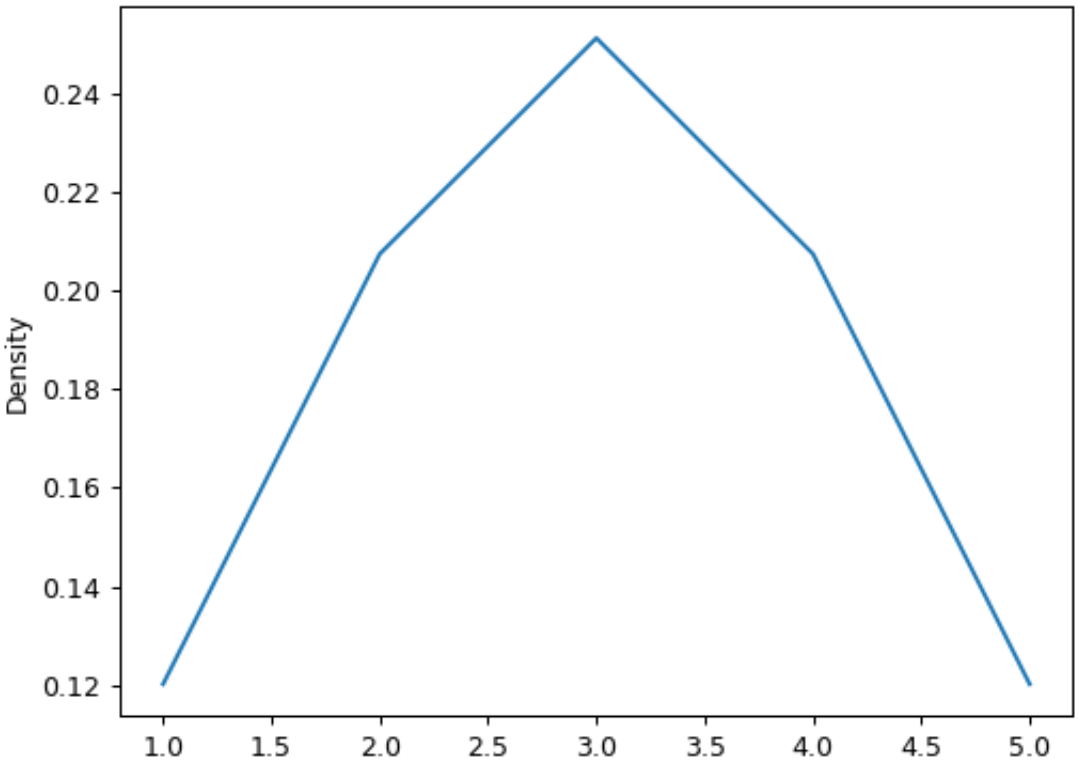
```
>>> ax = s.plot.kde(bw_method=3)
```

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


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


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

```
>>> 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()
```

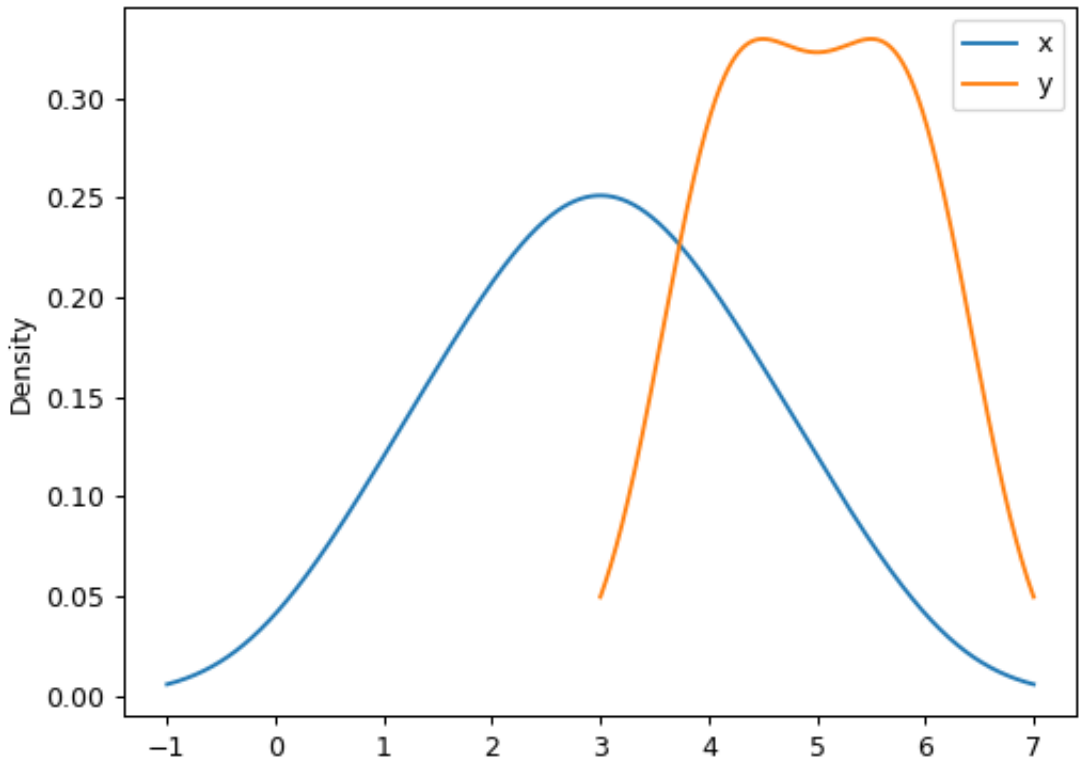
A scalar bandwidth can be specified. Using a small bandwidth value can lead to over-fitting, while using a large bandwidth value may result in under-fitting:

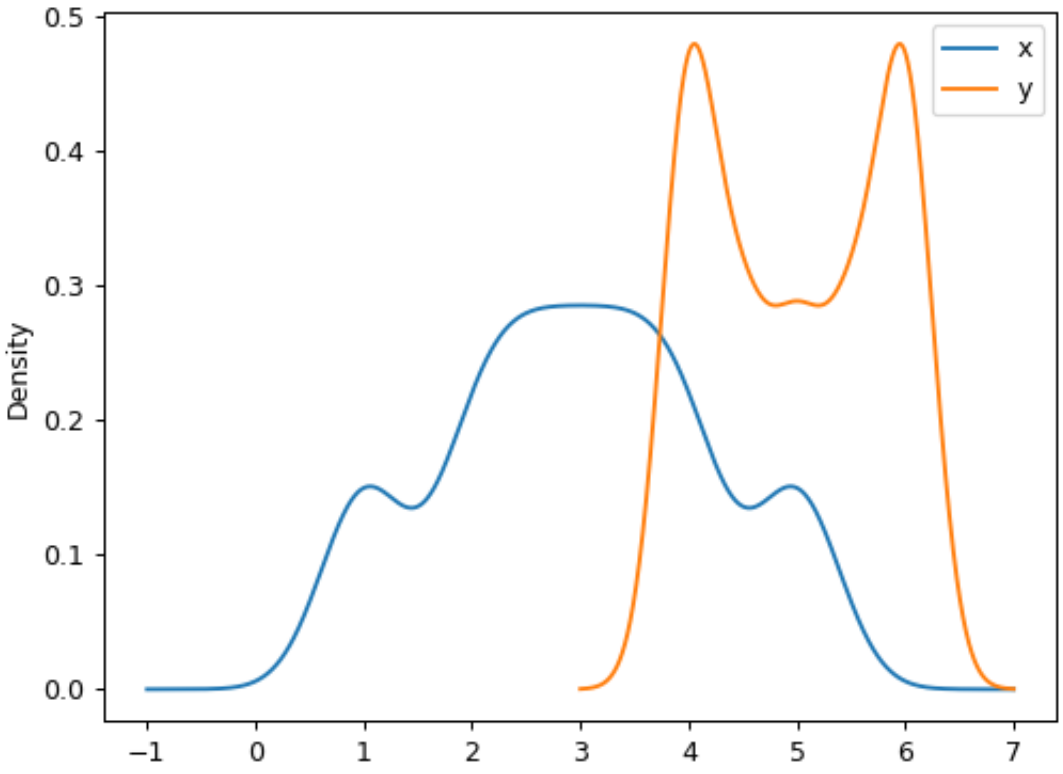
```
>>> ax = df.plot.kde(bw_method=0.3)
```

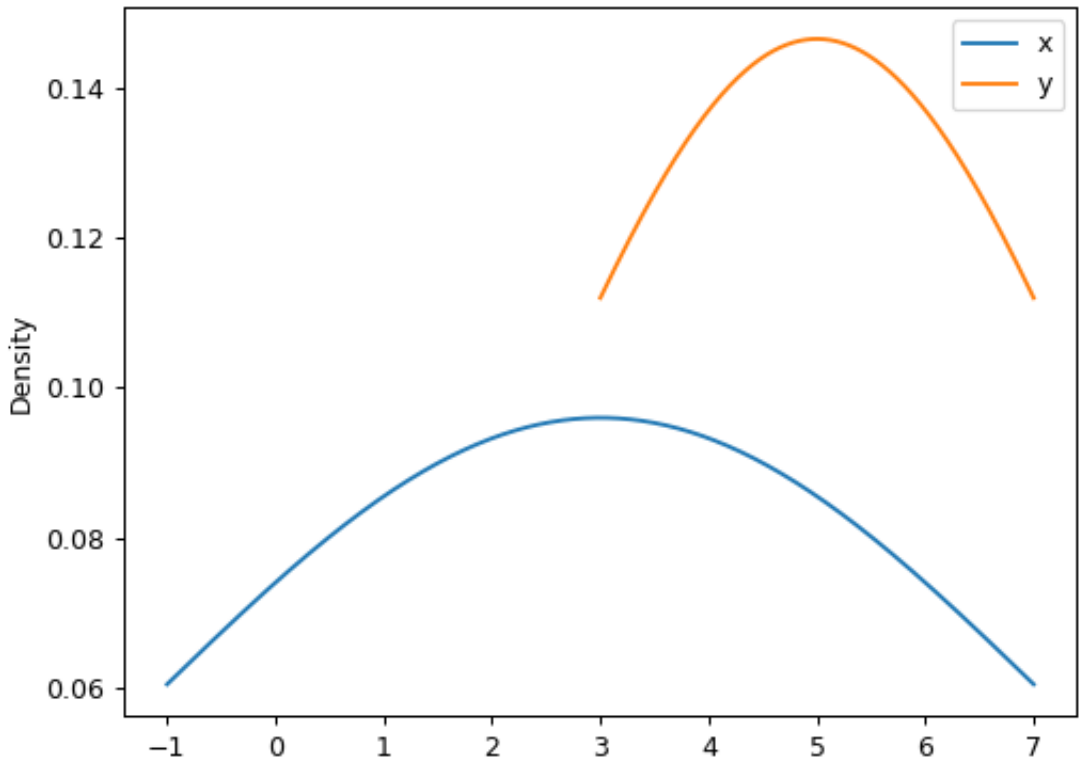
```
>>> ax = df.plot.kde(bw_method=3)
```

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])
```


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


0.5
0.4
0.3
0.2
0.1
0.0
Density
−1
0
1
2
3
4
5
6
7
x
y


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