pandas.io.formats.style.Styler.highlight_null

Styler.**highlight_null**(*self*, *null_color='red'*)

Shade the background `null_color` for missing values.

Parameters

null_color [str]

Returns

self [Styler]

pandas.io.formats.style.Styler.pipe

Styler.**pipe**(*self*, *func*, **args*, ***kwargs*)

Apply `func(self, *args, **kwargs)`, and return the result.

New in version 0.24.0.

Parameters

func [function] Function to apply to the Styler. Alternatively, a `(callable, keyword)` tuple where `keyword` is a string indicating the keyword of `callable` that expects the Styler.

***args** [optional] Arguments passed to *func*.

****kwargs** [optional] A dictionary of keyword arguments passed into `func`.

Returns

object : The value returned by `func`.

See also:

DataFrame.pipe Analogous method for DataFrame.

Styler.apply Apply a function row-wise, column-wise, or table-wise to modify the dataframe's styling.

Notes

Like `DataFrame.pipe()`, this method can simplify the application of several user-defined functions to a styler. Instead of writing:

```
f(g(df.style.set_precision(3), arg1=a), arg2=b, arg3=c)
```

users can write:

```
(df.style.set_precision(3)
   .pipe(g, arg1=a)
   .pipe(f, arg2=b, arg3=c))
```

In particular, this allows users to define functions that take a styler object, along with other parameters, and return the styler after making styling changes (such as calling *Styler.apply()* or *Styler.set_properties()*). Using `.pipe`, these user-defined style "transformations" can be interleaved with calls to the built-in Styler interface.

Examples

```
>>> def format_conversion(styler):
...     return (styler.set_properties(**{'text-align': 'right'})
...                   .format({'conversion': '{:.1%}'}))
```

The user-defined `format_conversion` function above can be called within a sequence of other style modifications:

```
>>> df = pd.DataFrame({'trial': list(range(5)),
...                    'conversion': [0.75, 0.85, np.nan, 0.7, 0.72]})
>>> (df.style
...    .highlight_min(subset=['conversion'], color='yellow')
...    .pipe(format_conversion)
...    .set_caption("Results with minimum conversion highlighted."))
```

pandas.io.formats.style.Styler.render

`Styler.`**`render`**(*self*, ***kwargs*)

Render the built up styles to HTML.

Parameters

****kwargs** Any additional keyword arguments are passed through to `self.template.render`. This is useful when you need to provide additional variables for a custom template.

Returns

rendered [str] The rendered HTML.

Notes

`Styler` objects have defined the `_repr_html_` method which automatically calls `self.render()` when it's the last item in a Notebook cell. When calling `Styler.render()` directly, wrap the result in `IPython.display.HTML` to view the rendered HTML in the notebook.

Pandas uses the following keys in render. Arguments passed in `**kwargs` take precedence, so think carefully if you want to override them:

- head
- cellstyle
- body
- uuid
- precision
- table_styles
- caption
- table_attributes

pandas.io.formats.style.Styler.set_caption

Styler.**set_caption**(*self*, *caption*)

Set the caption on a Styler.

Parameters

caption [str]

Returns

self [Styler]

pandas.io.formats.style.Styler.set_na_rep

Styler.**set_na_rep**(*self*, *na_rep: str*) → 'Styler'

Set the missing data representation on a Styler.

New in version 1.0.0.

Parameters

na_rep [str]

Returns

self [Styler]

pandas.io.formats.style.Styler.set_precision

Styler.**set_precision**(*self*, *precision*)

Set the precision used to render.

Parameters

precision [int]

Returns

self [Styler]

pandas.io.formats.style.Styler.set_properties

Styler.**set_properties**(*self*, *subset=None*, ***kwargs*)

Method to set one or more non-data dependent properties or each cell.

Parameters

subset [IndexSlice] A valid slice for `data` to limit the style application to.

****kwargs** [dict] A dictionary of property, value pairs to be set for each cell.

Returns

self [Styler]

Examples

```
>>> df = pd.DataFrame(np.random.randn(10, 4))
>>> df.style.set_properties(color="white", align="right")
>>> df.style.set_properties(**{'background-color': 'yellow'})
```

pandas.io.formats.style.Styler.set_table_attributes

Styler.**set_table_attributes**(*self*, *attributes*)

Set the table attributes.

These are the items that show up in the opening <table> tag in addition to to automatic (by default) id.

Parameters

attributes [str]

Returns

self [Styler]

Examples

```
>>> df = pd.DataFrame(np.random.randn(10, 4))
>>> df.style.set_table_attributes('class="pure-table"')
# ... <table class="pure-table"> ...
```

pandas.io.formats.style.Styler.set_table_styles

Styler.**set_table_styles**(*self*, *table_styles*)

Set the table styles on a Styler.

These are placed in a <style> tag before the generated HTML table.

Parameters

table_styles [list] Each individual table_style should be a dictionary with `selector` and `props` keys. `selector` should be a CSS selector that the style will be applied to (automatically prefixed by the table's UUID) and `props` should be a list of tuples with `(attribute, value)`.

Returns

self [Styler]

Examples

```
>>> df = pd.DataFrame(np.random.randn(10, 4))
>>> df.style.set_table_styles(
...     [{'selector': 'tr:hover',
...       'props': [('background-color', 'yellow')]}]
... )
```

pandas.io.formats.style.Styler.set_uuid

Styler.**set_uuid**(*self*, *uuid*)

Set the uuid for a Styler.

Parameters

uuid [str]

Returns

self [Styler]

pandas.io.formats.style.Styler.to_excel

Styler.**to_excel**(*self*, *excel_writer*, *sheet_name='Sheet1'*, *na_rep=''*, *float_format=None*, *columns=None*, *header=True*, *index=True*, *index_label=None*, *startrow=0*, *startcol=0*, *engine=None*, *merge_cells=True*, *encoding=None*, *inf_rep='inf'*, *verbose=True*, *freeze_panes=None*)

Write Styler to an Excel sheet.

To write a single Styler to an Excel .xlsx file it is only necessary to specify a target file name. To write to multiple sheets it is necessary to create an *ExcelWriter* object with a target file name, and specify a sheet in the file to write to.

Multiple sheets may be written to by specifying unique *sheet_name*. With all data written to the file it is necessary to save the changes. Note that creating an *ExcelWriter* object with a file name that already exists will result in the contents of the existing file being erased.

Parameters

excel_writer [str or ExcelWriter object] File path or existing ExcelWriter.

sheet_name [str, default 'Sheet1'] Name of sheet which will contain DataFrame.

na_rep [str, default ''] Missing data representation.

float_format [str, optional] Format string for floating point numbers. For example `float_format="%.2f"` will format 0.1234 to 0.12.

columns [sequence or list of str, optional] Columns to write.

header [bool or list of str, default True] Write out the column names. If a list of string is given it is assumed to be aliases for the column names.

index [bool, default True] Write row names (index).

index_label [str or sequence, optional] Column label for index column(s) if desired. If not specified, and *header* and *index* are True, then the index names are used. A sequence should be given if the DataFrame uses MultiIndex.

startrow [int, default 0] Upper left cell row to dump data frame.

startcol [int, default 0] Upper left cell column to dump data frame.

engine [str, optional] Write engine to use, 'openpyxl' or 'xlsxwriter'. You can also set this via the options `io.excel.xlsx.writer`, `io.excel.xls.writer`, and `io.excel.xlsm.writer`.

merge_cells [bool, default True] Write MultiIndex and Hierarchical Rows as merged cells.

encoding [str, optional] Encoding of the resulting excel file. Only necessary for xlwt, other writers support unicode natively.

inf_rep [str, default 'inf'] Representation for infinity (there is no native representation for infinity in Excel).

verbose [bool, default True] Display more information in the error logs.

freeze_panes [tuple of int (length 2), optional] Specifies the one-based bottommost row and rightmost column that is to be frozen.

See also:

`to_csv` Write DataFrame to a comma-separated values (csv) file.

`ExcelWriter` Class for writing DataFrame objects into excel sheets.

`read_excel` Read an Excel file into a pandas DataFrame.

`read_csv` Read a comma-separated values (csv) file into DataFrame.

Notes

For compatibility with `to_csv()`, to_excel serializes lists and dicts to strings before writing.

Once a workbook has been saved it is not possible write further data without rewriting the whole workbook.

Examples

Create, write to and save a workbook:

```
>>> df1 = pd.DataFrame([['a', 'b'], ['c', 'd']],
...                    index=['row 1', 'row 2'],
...                    columns=['col 1', 'col 2'])
>>> df1.to_excel("output.xlsx")
```

To specify the sheet name:

```
>>> df1.to_excel("output.xlsx",
...              sheet_name='Sheet_name_1')
```

If you wish to write to more than one sheet in the workbook, it is necessary to specify an ExcelWriter object:

```
>>> df2 = df1.copy()
>>> with pd.ExcelWriter('output.xlsx') as writer:
...     df1.to_excel(writer, sheet_name='Sheet_name_1')
...     df2.to_excel(writer, sheet_name='Sheet_name_2')
```

ExcelWriter can also be used to append to an existing Excel file:

```
>>> with pd.ExcelWriter('output.xlsx',
...                     mode='a') as writer:
...     df.to_excel(writer, sheet_name='Sheet_name_3')
```

To set the library that is used to write the Excel file, you can pass the *engine* keyword (the default engine is automatically chosen depending on the file extension):

```
>>> df1.to_excel('output1.xlsx', engine='xlsxwriter')
```

pandas.io.formats.style.Styler.use

`Styler.use`(*self*, *styles*)

Set the styles on the current Styler.

Possibly uses styles from `Styler.export`.

Parameters

styles [list] List of style functions.

Returns

self [Styler]

See also:

Styler.export

pandas.io.formats.style.Styler.where

`Styler.where`(*self*, *cond*, *value*, *other=None*, *subset=None*, ***kwargs*)

Apply a function elementwise.

Updates the HTML representation with a style which is selected in accordance with the return value of a function.

New in version 0.21.0.

Parameters

cond [callable] `cond` should take a scalar and return a boolean.

value [str] Applied when `cond` returns true.

other [str] Applied when `cond` returns false.

subset [IndexSlice] A valid indexer to limit `data` to *before* applying the function. Consider using a pandas.IndexSlice.

****kwargs** [dict] Pass along to `cond`.

Returns

self [Styler]

See also:

Styler.applymap

3.13.2 Styler properties

Styler.env	
Styler.template	
Styler.loader	

pandas.io.formats.style.Styler.env

Styler.**env = <jinja2.environment.Environment object>**

pandas.io.formats.style.Styler.template

Styler.**template = <Template 'html.tpl'>**

pandas.io.formats.style.Styler.loader

Styler.**loader = <jinja2.loaders.PackageLoader object>**

3.13.3 Style application

Styler.apply(self, func[, axis, subset])	Apply a function column-wise, row-wise, or table-wise.
Styler.applymap(self, func[, subset])	Apply a function elementwise.
Styler.where(self, cond, value[, other, subset])	Apply a function elementwise.
Styler.format(self, formatter[, subset])	Format the text display value of cells.
Styler.set_precision(self, precision)	Set the precision used to render.
Styler.set_table_styles(self, table_styles)	Set the table styles on a Styler.
Styler.set_table_attributes(self, attributes)	Set the table attributes.
Styler.set_caption(self, caption)	Set the caption on a Styler.
Styler.set_properties(self[, subset])	Method to set one or more non-data dependent properties or each cell.
Styler.set_uuid(self, uuid)	Set the uuid for a Styler.
Styler.set_na_rep(self, na_rep)	Set the missing data representation on a Styler.
Styler.clear(self)	Reset the styler, removing any previously applied styles.
Styler.pipe(self, func, *args, **kwargs)	Apply `func(self, *args, **kwargs)`, and return the result.

3.13.4 Builtin styles

Styler.highlight_max(self[, subset, color, axis])	Highlight the maximum by shading the background.
Styler.highlight_min(self[, subset, color, axis])	Highlight the minimum by shading the background.
Styler.highlight_null(self[, null_color])	Shade the background `null_color` for missing values.

continues on next page

Table 414 – continued from previous page

Styler.background_gradient(self[, cmap, ...])	Color the background in a gradient style.
Styler.bar(self[, subset, axis, color, ...])	Draw bar chart in the cell backgrounds.

3.13.5 Style export and import

Styler.render(self, **kwargs)	Render the built up styles to HTML.
Styler.export(self)	Export the styles to applied to the current Styler.
Styler.use(self, styles)	Set the styles on the current Styler.
Styler.to_excel(self, excel_writer[, ...])	Write Styler to an Excel sheet.

3.14 Plotting

The following functions are contained in the *pandas.plotting* module.

andrews_curves(frame, class_column[, ax, ...])	Generate a matplotlib plot of Andrews curves, for visualising clusters of multivariate data.
autocorrelation_plot(series[, ax])	Autocorrelation plot for time series.
bootstrap_plot(series[, fig, size, samples])	Bootstrap plot on mean, median and mid-range statistics.
boxplot(data[, column, by, ax, fontsize, ...])	Make a box plot from DataFrame columns.
deregister_matplotlib_converters()	Remove pandas' formatters and converters.
lag_plot(series[, lag, ax])	Lag plot for time series.
parallel_coordinates(frame, class_column[, ...])	Parallel coordinates plotting.
plot_params	Stores pandas plotting options.
radviz(frame, class_column[, ax, color, ...])	Plot a multidimensional dataset in 2D.
register_matplotlib_converters()	Register Pandas Formatters and Converters with matplotlib.
scatter_matrix(frame[, alpha, figsize, ax, ...])	Draw a matrix of scatter plots.
table(ax, data[, rowLabels, colLabels])	Helper function to convert DataFrame and Series to matplotlib.table.

3.14.1 pandas.plotting.andrews_curves

pandas.plotting.**andrews_curves**(*frame*, *class_column*, *ax=None*, *samples=200*, *color=None*, *colormap=None*, ***kwargs*)

Generate a matplotlib plot of Andrews curves, for visualising clusters of multivariate data.

Andrews curves have the functional form:
f(t) = x_1/sqrt(2) + x_2 sin(t) + x_3 cos(t) + x_4 sin(2t) + x_5 cos(2t) + ...
Where x coefficients correspond to the values of each dimension and t is linearly spaced between -pi and +pi. Each row of frame then corresponds to a single curve.

Parameters

frame [DataFrame] Data to be plotted, preferably normalized to (0.0, 1.0).

class_column [Name of the column containing class names]

ax [matplotlib axes object, default None]

samples [Number of points to plot in each curve]

color [list or tuple, optional] Colors to use for the different classes.

colormap [str or matplotlib colormap object, default None] Colormap to select colors from. If string, load colormap with that name from matplotlib.

****kwargs** Options to pass to matplotlib plotting method.

Returns

class:*matplotlip.axis.Axes*

3.14.2 pandas.plotting.autocorrelation_plot

pandas.plotting.**autocorrelation_plot**(*series*, *ax=None*, ***kwargs*)

Autocorrelation plot for time series.

Parameters

series [Time series]

ax [Matplotlib axis object, optional]

****kwargs** Options to pass to matplotlib plotting method.

Returns

class:*matplotlib.axis.Axes*

3.14.3 pandas.plotting.bootstrap_plot

pandas.plotting.**bootstrap_plot**(*series*, *fig=None*, *size=50*, *samples=500*, ***kwds*)

Bootstrap plot on mean, median and mid-range statistics.

The bootstrap plot is used to estimate the uncertainty of a statistic by relaying on random sampling with replacement [1]. This function will generate bootstrapping plots for mean, median and mid-range statistics for the given number of samples of the given size.

Parameters

series [pandas.Series] Pandas Series from where to get the samplings for the bootstrapping.

fig [matplotlib.figure.Figure, default None] If given, it will use the *fig* reference for plotting instead of creating a new one with default parameters.

size [int, default 50] Number of data points to consider during each sampling. It must be greater or equal than the length of the *series*.

samples [int, default 500] Number of times the bootstrap procedure is performed.

****kwds** Options to pass to matplotlib plotting method.

Returns

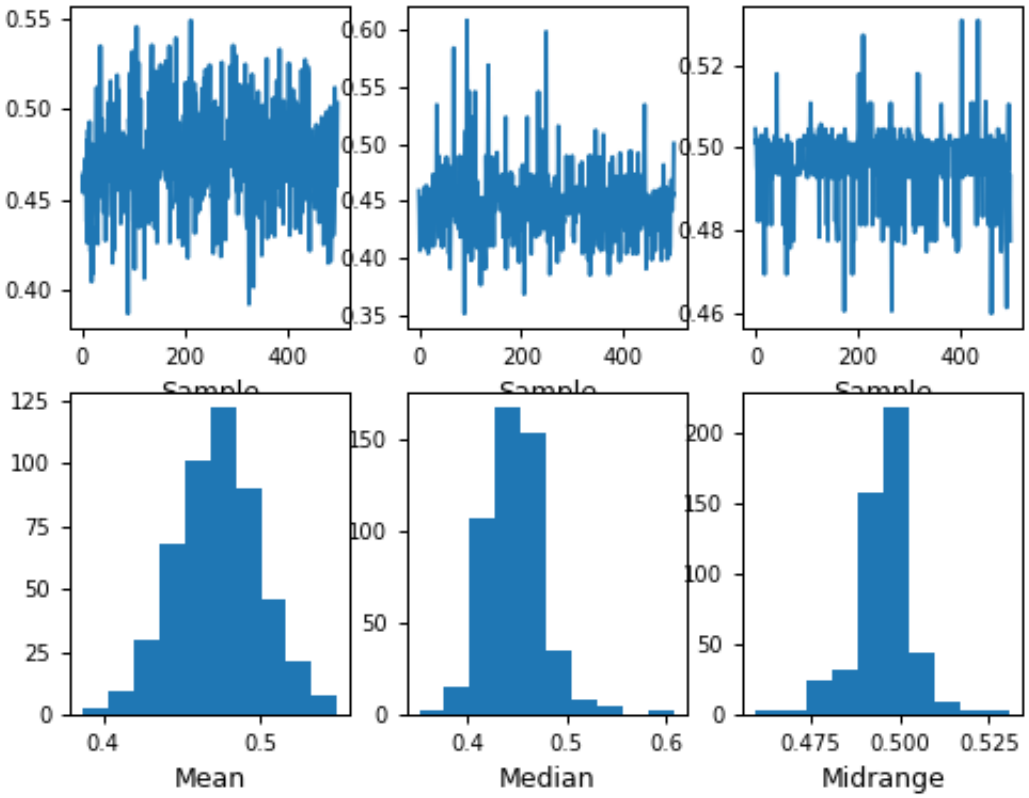
matplotlib.figure.Figure Matplotlib figure.

See also:

DataFrame.plot Basic plotting for DataFrame objects.
Series.plot Basic plotting for Series objects.

Examples

```
>>> s = pd.Series(np.random.uniform(size=100))
>>> fig = pd.plotting.bootstrap_plot(s)
```



3.14.4 pandas.plotting.boxplot

pandas.plotting.**boxplot**(*data*, *column=None*, *by=None*, *ax=None*, *fontsize=None*, *rot=0*, *grid=True*, *figsize=None*, *layout=None*, *return_type=None*, ***kwargs*)

Make a box plot from DataFrame columns.

Make a box-and-whisker plot from DataFrame columns, optionally grouped by some other 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.

Parameters

column [str or list of str, optional] Column name or list of names, or vector. Can be any valid input to *pandas.DataFrame.groupby()*.

by [str or array-like, optional] Column in the DataFrame to *pandas.DataFrame.groupby()*. One box-plot will be done per value of columns in *by*.

ax [object of class matplotlib.axes.Axes, optional] The matplotlib axes to be used by boxplot.

fontsize [float or str] Tick label font size in points or as a string (e.g., *large*).

rot [int or float, default 0] The rotation angle of labels (in degrees) with respect to the screen coordinate system.

grid [bool, default True] Setting this to True will show the grid.

figsize [A tuple (width, height) in inches] The size of the figure to create in matplotlib.

layout [tuple (rows, columns), optional] For example, (3, 5) will display the subplots using 3 columns and 5 rows, starting from the top-left.

return_type [{'axes', 'dict', 'both'} or None, default 'axes'] The kind of object to return. The default is `axes`.

- 'axes' returns the matplotlib axes the boxplot is drawn on.
- 'dict' returns a dictionary whose values are the matplotlib Lines of the boxplot.
- 'both' returns a namedtuple with the axes and dict.
- when grouping with `by`, a Series mapping columns to `return_type` is returned.

  If `return_type` is *None*, a NumPy array of axes with the same shape as `layout` is returned.

****kwargs** All other plotting keyword arguments to be passed to `matplotlib.pyplot.boxplot()`.

Returns

result See Notes.

See also:

`Series.plot.hist` Make a histogram.
`matplotlib.pyplot.boxplot` Matplotlib equivalent plot.

Notes

The return type depends on the *return_type* parameter:

- 'axes' : object of class matplotlib.axes.Axes
- 'dict' : dict of matplotlib.lines.Line2D objects
- 'both' : a namedtuple with structure (ax, lines)

For data grouped with `by`, return a Series of the above or a numpy array:

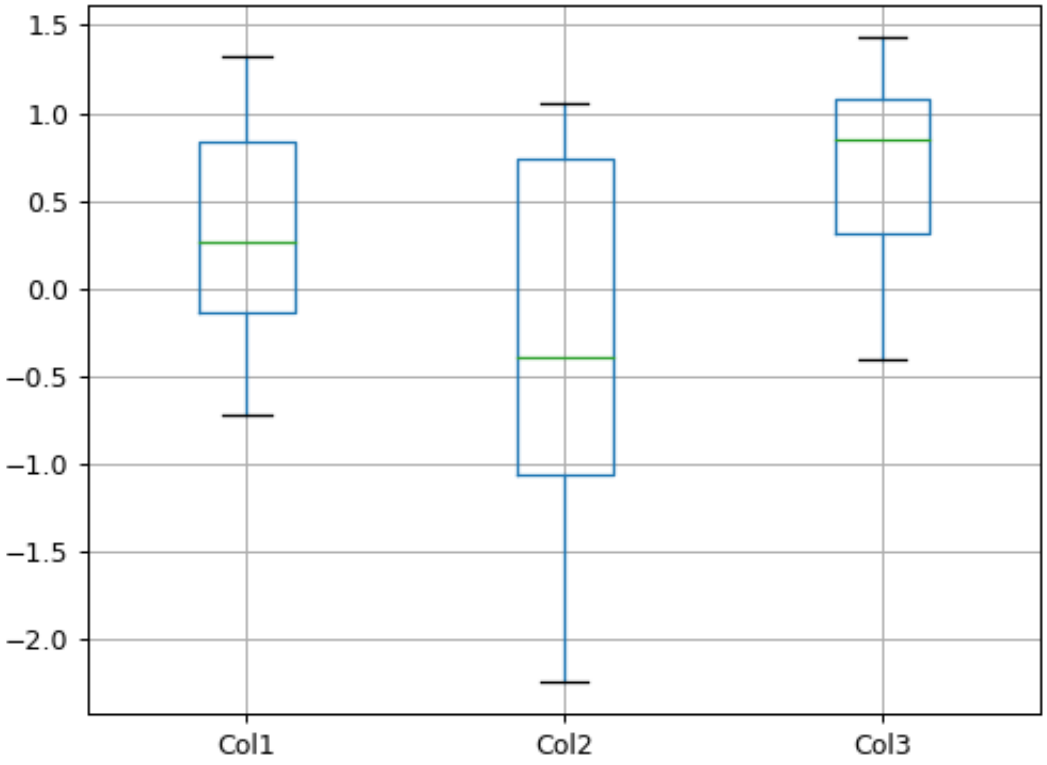
- *Series*
- `array` (for `return_type = None`)

Use `return_type='dict'` when you want to tweak the appearance of the lines after plotting. In this case a dict containing the Lines making up the boxes, caps, fliers, medians, and whiskers is returned.

Examples

Boxplots can be created for every column in the dataframe by `df.boxplot()` or indicating the columns to be used:

```
>>> np.random.seed(1234)
>>> df = pd.DataFrame(np.random.randn(10, 4),
...                   columns=['Col1', 'Col2', 'Col3', 'Col4'])
>>> boxplot = df.boxplot(column=['Col1', 'Col2', 'Col3'])
```



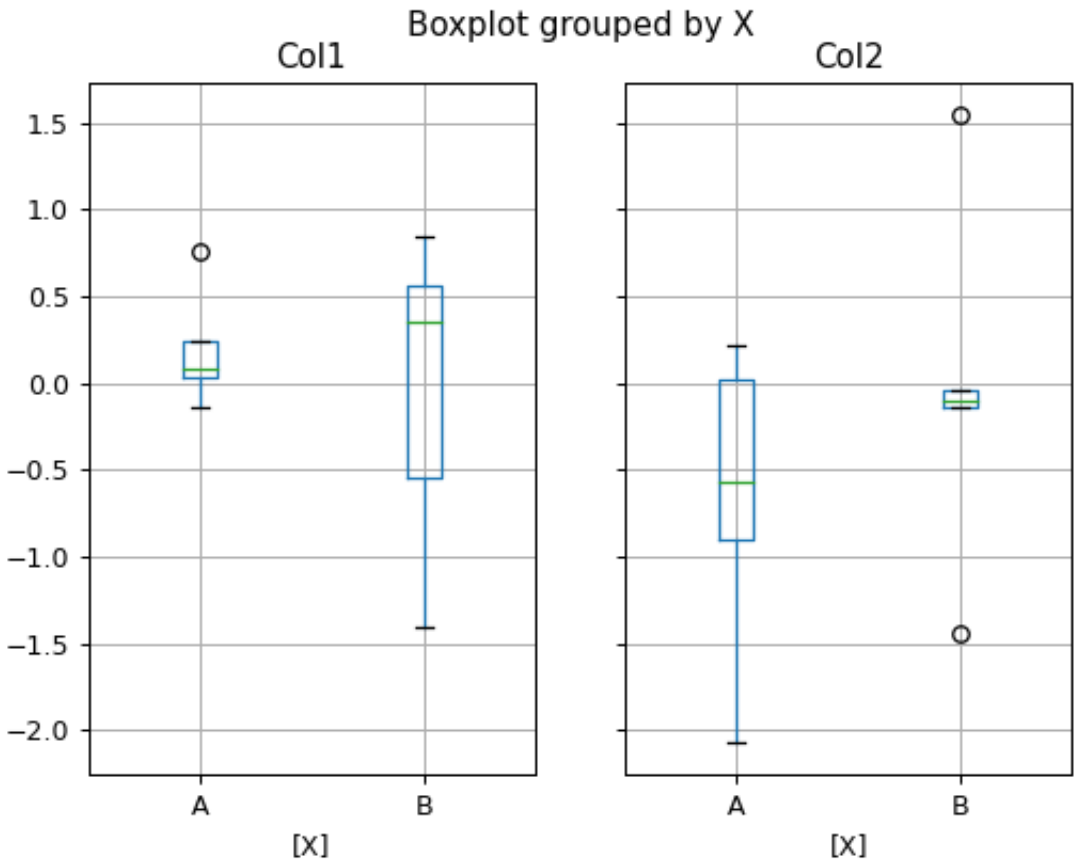
Boxplots of variables distributions grouped by the values of a third variable can be created using the option `by`. For instance:

```
>>> df = pd.DataFrame(np.random.randn(10, 2),
...                   columns=['Col1', 'Col2'])
>>> df['X'] = pd.Series(['A', 'A', 'A', 'A', 'A',
...                      'B', 'B', 'B', 'B', 'B'])
>>> boxplot = df.boxplot(by='X')
```

A list of strings (i.e. `['X', 'Y']`) can be passed to boxplot in order to group the data by combination of the variables in the x-axis:

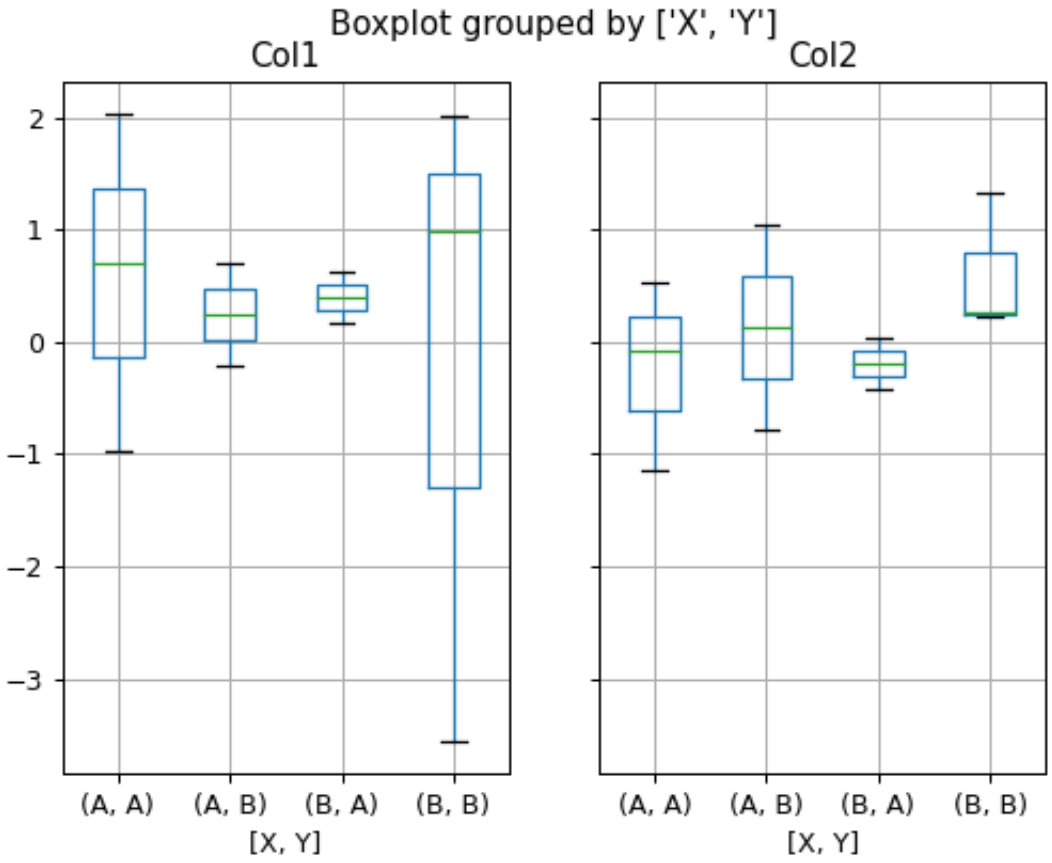
```
>>> df = pd.DataFrame(np.random.randn(10, 3),
...                   columns=['Col1', 'Col2', 'Col3'])
```

(continues on next page)


Boxplot grouped by X
Col1
Col2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
A
B
[X]
A
B
[X]

(continued from previous page)

```
>>> df['X'] = pd.Series(['A', 'A', 'A', 'A', 'A',
...                      'B', 'B', 'B', 'B', 'B'])
>>> df['Y'] = pd.Series(['A', 'B', 'A', 'B', 'A',
...                      'B', 'A', 'B', 'A', 'B'])
>>> boxplot = df.boxplot(column=['Col1', 'Col2'], by=['X', 'Y'])
```



The layout of boxplot can be adjusted giving a tuple to `layout`:

```
>>> boxplot = df.boxplot(column=['Col1', 'Col2'], by='X',
...                      layout=(2, 1))
```

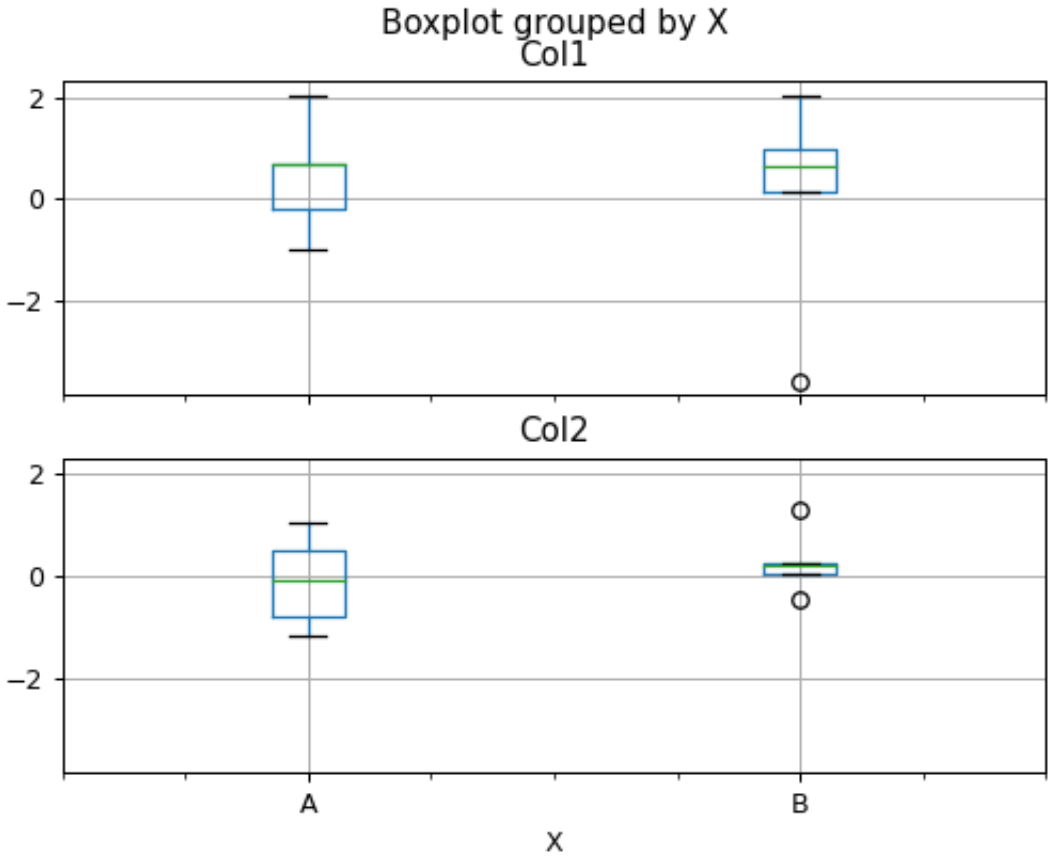
Additional formatting can be done to the boxplot, like suppressing the grid (`grid=False`), rotating the labels in the x-axis (i.e. `rot=45`) or changing the fontsize (i.e. `fontsize=15`):

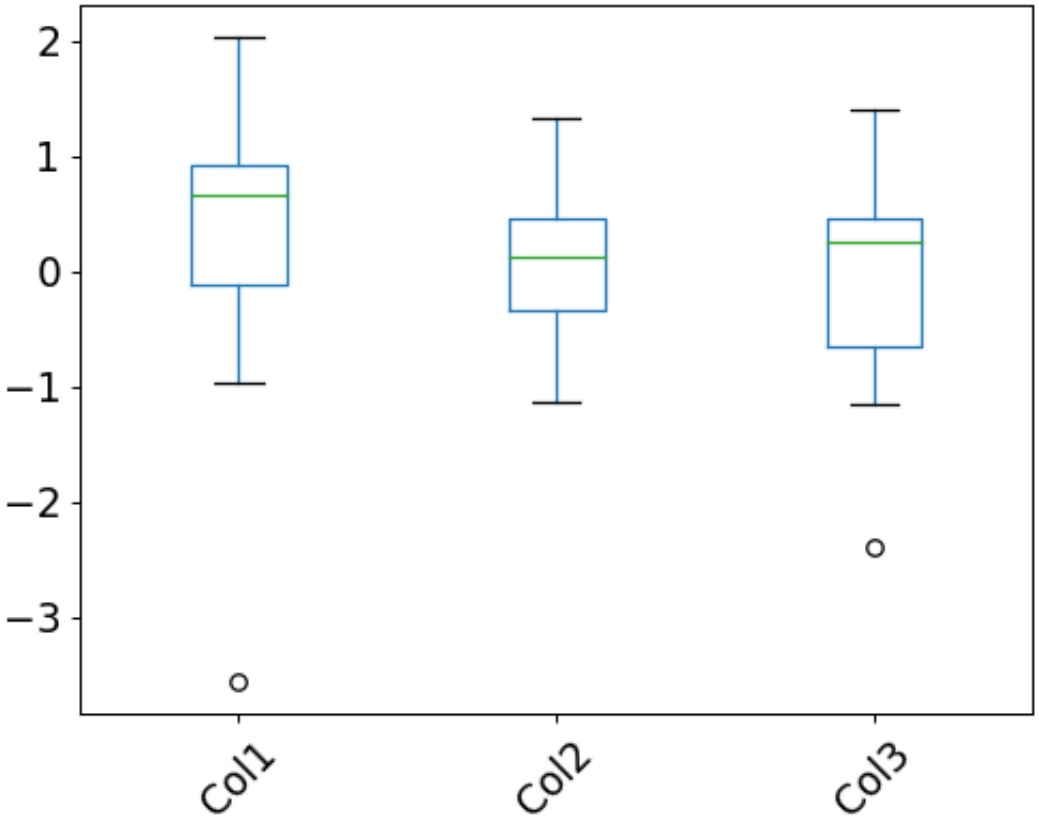
```
>>> boxplot = df.boxplot(grid=False, rot=45, fontsize=15)
```

The parameter `return_type` can be used to select the type of element returned by *boxplot*. When `return_type='axes'` is selected, the matplotlib axes on which the boxplot is drawn are returned:

```
>>> boxplot = df.boxplot(column=['Col1', 'Col2'], return_type='axes')
>>> type(boxplot)
<class 'matplotlib.axes._subplots.AxesSubplot'>
```

When grouping with `by`, a Series mapping columns to `return_type` is returned:


Boxplot grouped by X
Col1
2
0
−2
Col2
2
0
−2
A
B
X


2
1
0
−1
−2
−3
Col1
Col2
Col3

```
>>> boxplot = df.boxplot(column=['Col1', 'Col2'], by='X',
...                      return_type='axes')
>>> type(boxplot)
<class 'pandas.core.series.Series'>
```

If `return_type` is *None*, a NumPy array of axes with the same shape as `layout` is returned:

```
>>> boxplot = df.boxplot(column=['Col1', 'Col2'], by='X',
...                      return_type=None)
>>> type(boxplot)
<class 'numpy.ndarray'>
```

3.14.5 pandas.plotting.deregister_matplotlib_converters

pandas.plotting.**deregister_matplotlib_converters**()

Remove pandas' formatters and converters.

Removes the custom converters added by `register()`. This attempts to set the state of the registry back to the state before pandas registered its own units. Converters for pandas' own types like Timestamp and Period are removed completely. Converters for types pandas overwrites, like `datetime.datetime`, are restored to their original value.

See also:

register_matplotlib_converters

3.14.6 pandas.plotting.lag_plot

pandas.plotting.**lag_plot**(*series*, *lag=1*, *ax=None*, ***kwds*)

Lag plot for time series.

Parameters

series [Time series]

lag [lag of the scatter plot, default 1]

ax [Matplotlib axis object, optional]

****kwds** Matplotlib scatter method keyword arguments.

Returns

class:*matplotlib.axis.Axes*

3.14.7 pandas.plotting.parallel_coordinates

pandas.plotting.**parallel_coordinates**(*frame*, *class_column*, *cols=None*, *ax=None*, *color=None*, *use_columns=False*, *xticks=None*, *colormap=None*, *axvlines=True*, *axvlines_kwds=None*, *sort_labels=False*, ***kwargs*)

Parallel coordinates plotting.

Parameters

frame [DataFrame]

class_column [str] Column name containing class names.

cols [list, optional] A list of column names to use.

ax [matplotlib.axis, optional] Matplotlib axis object.

color [list or tuple, optional] Colors to use for the different classes.

use_columns [bool, optional] If true, columns will be used as xticks.

xticks [list or tuple, optional] A list of values to use for xticks.

colormap [str or matplotlib colormap, default None] Colormap to use for line colors.

axvlines [bool, optional] If true, vertical lines will be added at each xtick.

axvlines_kwds [keywords, optional] Options to be passed to axvline method for vertical lines.

sort_labels [bool, default False] Sort class_column labels, useful when assigning colors.

****kwargs** Options to pass to matplotlib plotting method.

Returns

class:*matplotlib.axis.Axes*

Examples

```
>>> from matplotlib import pyplot as plt
>>> df = pd.read_csv('https://raw.github.com/pandas-dev/pandas/master'
                     '/pandas/tests/data/csv/iris.csv')
>>> pd.plotting.parallel_coordinates(
        df, 'Name',
        color=('#556270', '#4ECDC4', '#C7F464'))
>>> plt.show()
```

3.14.8 pandas.plotting.plot_params

pandas.plotting.**plot_params = {'xaxis.compat': False}**

Stores pandas plotting options.

Allows for parameter aliasing so you can just use parameter names that are the same as the plot function parameters, but is stored in a canonical format that makes it easy to breakdown into groups later.

3.14.9 pandas.plotting.radviz

pandas.plotting.**radviz**(*frame*, *class_column*, *ax=None*, *color=None*, *colormap=None*, ***kwds*)

Plot a multidimensional dataset in 2D.

Each Series in the DataFrame is represented as a evenly distributed slice on a circle. Each data point is rendered in the circle according to the value on each Series. Highly correlated *Series* in the *DataFrame* are placed closer on the unit circle.

RadViz allow to project a N-dimensional data set into a 2D space where the influence of each dimension can be interpreted as a balance between the influence of all dimensions.

More info available at the original article describing RadViz.

Parameters

frame [*DataFrame*] Pandas object holding the data.

class_column [str] Column name containing the name of the data point category.

ax [matplotlib.axes.Axes, optional] A plot instance to which to add the information.

color [list[str] or tuple[str], optional] Assign a color to each category. Example: ['blue', 'green'].

colormap [str or matplotlib.colors.Colormap, default None] Colormap to select colors from. If string, load colormap with that name from matplotlib.

****kwds** Options to pass to matplotlib scatter plotting method.

Returns

class:*matplotlib.axes.Axes*

See also:

plotting.andrews_curves Plot clustering visualization.

Examples

```
>>> df = pd.DataFrame({
...          'SepalLength': [6.5, 7.7, 5.1, 5.8, 7.6, 5.0, 5.4, 4.6,
...                          6.7, 4.6],
...          'SepalWidth': [3.0, 3.8, 3.8, 2.7, 3.0, 2.3, 3.0, 3.2,
...                         3.3, 3.6],
...          'PetalLength': [5.5, 6.7, 1.9, 5.1, 6.6, 3.3, 4.5, 1.4,
...                         5.7, 1.0],
...          'PetalWidth': [1.8, 2.2, 0.4, 1.9, 2.1, 1.0, 1.5, 0.2,
...                         2.1, 0.2],
...          'Category': ['virginica', 'virginica', 'setosa',
...                       'virginica', 'virginica', 'versicolor',
...                       'versicolor', 'setosa', 'virginica',
...                       'setosa']
...      })
>>> rad_viz = pd.plotting.radviz(df, 'Category')
```

3.14.10 pandas.plotting.register_matplotlib_converters

pandas.plotting.**register_matplotlib_converters**()

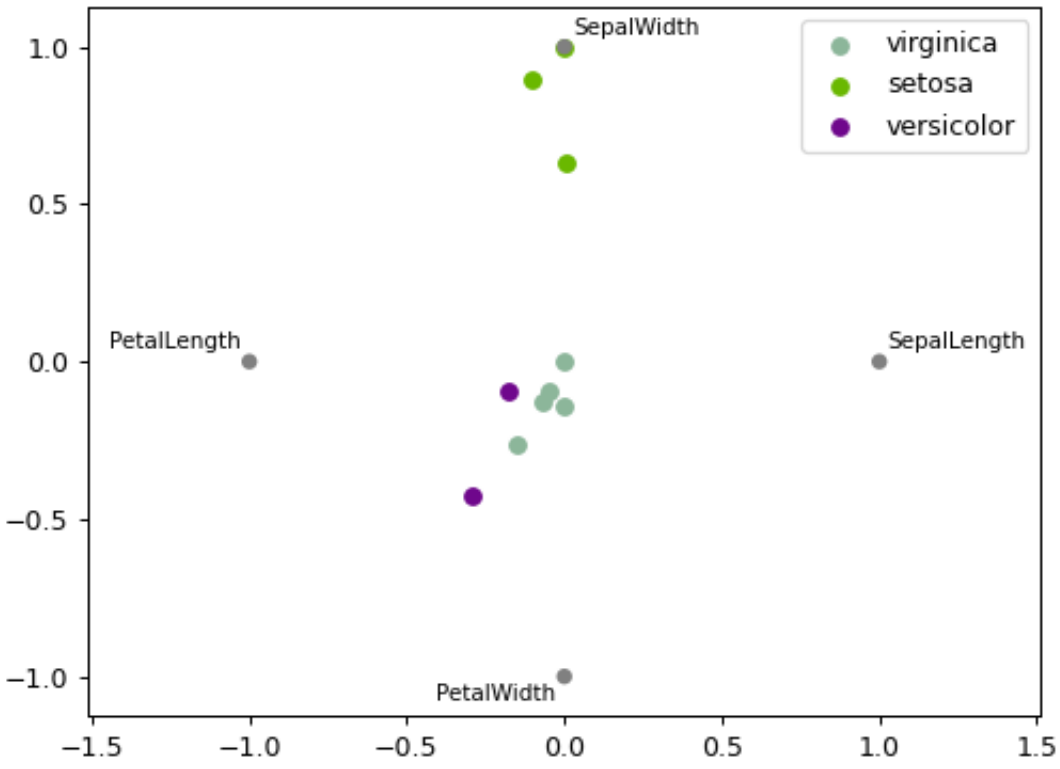
Register Pandas Formatters and Converters with matplotlib.

This function modifies the global matplotlib.units.registry dictionary. Pandas adds custom converters for

- pd.Timestamp
- pd.Period
- np.datetime64
- datetime.datetime
- datetime.date
- datetime.time

See also:

deregister_matplotlib_converters


SepalWidth
virginica
setosa
versicolor
PetalLength
SepalLength
PetalWidth
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

3.14.11 pandas.plotting.scatter_matrix

pandas.plotting.**scatter_matrix**(*frame*, *alpha=0.5*, *figsize=None*, *ax=None*, *grid=False*, *diagonal='hist'*, *marker='.'*, *density_kwds=None*, *hist_kwds=None*, *range_padding=0.05*, ***kwargs*)

Draw a matrix of scatter plots.

Parameters

frame [DataFrame]

alpha [float, optional] Amount of transparency applied.

figsize [(float,float), optional] A tuple (width, height) in inches.

ax [Matplotlib axis object, optional]

grid [bool, optional] Setting this to True will show the grid.

diagonal [{'hist', 'kde'}] Pick between 'kde' and 'hist' for either Kernel Density Estimation or Histogram plot in the diagonal.

marker [str, optional] Matplotlib marker type, default '.'.

density_kwds [keywords] Keyword arguments to be passed to kernel density estimate plot.

hist_kwds [keywords] Keyword arguments to be passed to hist function.

range_padding [float, default 0.05] Relative extension of axis range in x and y with respect to (x_max - x_min) or (y_max - y_min).

****kwargs** Keyword arguments to be passed to scatter function.

Returns

numpy.ndarray A matrix of scatter plots.

Examples

```
>>> df = pd.DataFrame(np.random.randn(1000, 4), columns=['A','B','C','D'])
>>> scatter_matrix(df, alpha=0.2)
```

3.14.12 pandas.plotting.table

pandas.plotting.**table**(*ax*, *data*, *rowLabels=None*, *colLabels=None*, ***kwargs*)

Helper function to convert DataFrame and Series to matplotlib.table.

Parameters

ax [Matplotlib axes object]

data [DataFrame or Series] Data for table contents.

****kwargs** Keyword arguments to be passed to matplotlib.table.table. If *rowLabels* or *colLabels* is not specified, data index or column name will be used.

Returns

matplotlib table object

3.15 General utility functions

3.15.1 Working with options

describe_option(pat[, _print_desc])	Prints the description for one or more registered options.
reset_option(pat)	Reset one or more options to their default value.
get_option(pat)	Retrieves the value of the specified option.
set_option(pat, value)	Sets the value of the specified option.
option_context(*args)	Context manager to temporarily set options in the *with* statement context.

pandas.describe_option

pandas.**describe_option**(*pat*, *_print_desc=False*) **= <pandas._config.config.CallableDynamicDoc object>**

Prints the description for one or more registered options.

Call with not arguments to get a listing for all registered options.

Available options:

- compute.[use_bottleneck, use_numexpr]
- display.[chop_threshold, colheader_justify, column_space, date_dayfirst, date_yearfirst, encoding, expand_frame_repr, float_format]
- display.html.[border, table_schema, use_mathjax]
- display.[large_repr]
- display.latex.[escape, longtable, multicolumn, multicolumn_format, multirow, repr]
- display.[max_categories, max_columns, max_colwidth, max_info_columns, max_info_rows, max_rows, max_seq_items, memory_usage, min_rows, multi_sparse, notebook_repr_html, pprint_nest_depth, precision, show_dimensions]
- display.unicode.[ambiguous_as_wide, east_asian_width]
- display.[width]
- io.excel.ods.[reader]
- io.excel.xls.[reader, writer]
- io.excel.xlsb.[reader]
- io.excel.xlsm.[reader, writer]
- io.excel.xlsx.[reader, writer]
- io.hdf.[default_format, dropna_table]
- io.parquet.[engine]
- mode.[chained_assignment, sim_interactive, use_inf_as_na, use_inf_as_null]
- plotting.[backend]
- plotting.matplotlib.[register_converters]

Parameters

pat [str] Regexp pattern. All matching keys will have their description displayed.

_print_desc [bool, default True] If True (default) the description(s) will be printed to stdout. Otherwise, the description(s) will be returned as a unicode string (for testing).

Returns

None by default, the description(s) as a unicode string if _print_desc

is False

Notes

The available options with its descriptions:

compute.use_bottleneck [bool] Use the bottleneck library to accelerate if it is installed, the default is True Valid values: False,True [default: True] [currently: True]

compute.use_numexpr [bool] Use the numexpr library to accelerate computation if it is installed, the default is True Valid values: False,True [default: True] [currently: True]

display.chop_threshold [float or None] if set to a float value, all float values smaller then the given threshold will be displayed as exactly 0 by repr and friends. [default: None] [currently: None]

display.colheader_justify ['left'/'right'] Controls the justification of column headers. used by DataFrameFormatter. [default: right] [currently: right]

display.column_space No description available. [default: 12] [currently: 12]

display.date_dayfirst [boolean] When True, prints and parses dates with the day first, eg 20/01/2005 [default: False] [currently: False]

display.date_yearfirst [boolean] When True, prints and parses dates with the year first, eg 2005/01/20 [default: False] [currently: False]

display.encoding [str/unicode] Defaults to the detected encoding of the console. Specifies the encoding to be used for strings returned by to_string, these are generally strings meant to be displayed on the console. [default: UTF-8] [currently: UTF-8]

display.expand_frame_repr [boolean] Whether to print out the full DataFrame repr for wide DataFrames across multiple lines, *max_columns* is still respected, but the output will wrap-around across multiple "pages" if its width exceeds *display.width*. [default: True] [currently: True]

display.float_format [callable] The callable should accept a floating point number and return a string with the desired format of the number. This is used in some places like SeriesFormatter. See formats.format.EngFormatter for an example. [default: None] [currently: None]

display.html.border [int] A `border=value` attribute is inserted in the `<table>` tag for the DataFrame HTML repr. [default: 1] [currently: 1]

display.html.table_schema [boolean] Whether to publish a Table Schema representation for frontends that support it. (default: False) [default: False] [currently: False]

display.html.use_mathjax [boolean] When True, Jupyter notebook will process table contents using MathJax, rendering mathematical expressions enclosed by the dollar symbol. (default: True) [default: True] [currently: True]

display.large_repr ['truncate'/'info'] For DataFrames exceeding max_rows/max_cols, the repr (and HTML repr) can show a truncated table (the default from 0.13), or switch to the view from df.info() (the behaviour in earlier versions of pandas). [default: truncate] [currently: truncate]

display.latex.escape [bool] This specifies if the to_latex method of a Dataframe uses escapes special characters. Valid values: False,True [default: True] [currently: True]

display.latex.longtable :bool This specifies if the to_latex method of a Dataframe uses the longtable format. Valid values: False,True [default: False] [currently: False]

display.latex.multicolumn [bool] This specifies if the to_latex method of a Dataframe uses multicolumns to pretty-print MultiIndex columns. Valid values: False,True [default: True] [currently: True]

display.latex.multicolumn_format [bool] This specifies if the to_latex method of a Dataframe uses multicolumns to pretty-print MultiIndex columns. Valid values: False,True [default: l] [currently: l]

display.latex.multirow [bool] This specifies if the to_latex method of a Dataframe uses multirows to pretty-print MultiIndex rows. Valid values: False,True [default: False] [currently: False]

display.latex.repr [boolean] Whether to produce a latex DataFrame representation for jupyter environments that support it. (default: False) [default: False] [currently: False]

display.max_categories [int] This sets the maximum number of categories pandas should output when printing out a *Categorical* or a Series of dtype "category". [default: 8] [currently: 8]

display.max_columns [int] If max_cols is exceeded, switch to truncate view. Depending on *large_repr*, objects are either centrally truncated or printed as a summary view. 'None' value means unlimited.

In case python/IPython is running in a terminal and *large_repr* equals 'truncate' this can be set to 0 and pandas will auto-detect the width of the terminal and print a truncated object which fits the screen width.

The IPython notebook, IPython qtconsole, or IDLE do not run in a terminal and hence it is not possible to do correct auto-detection. [default: 0] [currently: 0]

display.max_colwidth [int or None] The maximum width in characters of a column in the repr of a pandas data structure. When the column overflows, a "..." placeholder is embedded in the output. A 'None' value means unlimited. [default: 50] [currently: 50]

display.max_info_columns [int] max_info_columns is used in DataFrame.info method to decide if per column information will be printed. [default: 100] [currently: 100]

display.max_info_rows [int or None] df.info() will usually show null-counts for each column. For large frames this can be quite slow. max_info_rows and max_info_cols limit this null check only to frames with smaller dimensions than specified. [default: 1690785] [currently: 1690785]

display.max_rows [int] If max_rows is exceeded, switch to truncate view. Depending on *large_repr*, objects are either centrally truncated or printed as a summary view. 'None' value means unlimited.

In case python/IPython is running in a terminal and *large_repr* equals 'truncate' this can be set to 0 and pandas will auto-detect the height of the terminal and print a truncated object which fits the screen height. The IPython notebook, IPython qtconsole, or IDLE do not run in a terminal and hence it is not possible to do correct auto-detection. [default: 60] [currently: 15]

display.max_seq_items [int or None] when pretty-printing a long sequence, no more then *max_seq_items* will be printed. If items are omitted, they will be denoted by the addition of "..." to the resulting string.

If set to None, the number of items to be printed is unlimited. [default: 100] [currently: 100]

display.memory_usage [bool, string or None] This specifies if the memory usage of a DataFrame should be displayed when df.info() is called. Valid values True,False,'deep' [default: True] [currently: True]

display.min_rows [int] The numbers of rows to show in a truncated view (when *max_rows* is exceeded). Ignored when *max_rows* is set to None or 0. When set to None, follows the value of *max_rows*. [default: 10] [currently: 10]

display.multi_sparse [boolean] "sparsify" MultiIndex display (don't display repeated elements in outer levels within groups) [default: True] [currently: True]

display.notebook_repr_html [boolean] When True, IPython notebook will use html representation for pandas objects (if it is available). [default: True] [currently: True]

display.pprint_nest_depth [int] Controls the number of nested levels to process when pretty-printing [default: 3] [currently: 3]

display.precision [int] Floating point output precision (number of significant digits). This is only a suggestion [default: 6] [currently: 6]

display.show_dimensions [boolean or 'truncate'] Whether to print out dimensions at the end of DataFrame repr. If 'truncate' is specified, only print out the dimensions if the frame is truncated (e.g. not display all rows and/or columns) [default: truncate] [currently: truncate]

display.unicode.ambiguous_as_wide [boolean] Whether to use the Unicode East Asian Width to calculate the display text width. Enabling this may affect to the performance (default: False) [default: False] [currently: False]

display.unicode.east_asian_width [boolean] Whether to use the Unicode East Asian Width to calculate the display text width. Enabling this may affect to the performance (default: False) [default: False] [currently: False]

display.width [int] Width of the display in characters. In case python/IPython is running in a terminal this can be set to None and pandas will correctly auto-detect the width. Note that the IPython notebook, IPython qtconsole, or IDLE do not run in a terminal and hence it is not possible to correctly detect the width. [default: 80] [currently: 80]

io.excel.ods.reader [string] The default Excel reader engine for 'ods' files. Available options: auto, odf. [default: auto] [currently: auto]

io.excel.xls.reader [string] The default Excel reader engine for 'xls' files. Available options: auto, xlrd. [default: auto] [currently: auto]

io.excel.xls.writer [string] The default Excel writer engine for 'xls' files. Available options: auto, xlwt. [default: auto] [currently: auto]

io.excel.xlsb.reader [string] The default Excel reader engine for 'xlsb' files. Available options: auto, pyxlsb. [default: auto] [currently: auto]