seaborn.objects.Plot

class seaborn.objects.**Plot**(**args*, *data=None*, *x=None*, *y=None*, *color=None*, *alpha=None*, *fill=None*, *marker=None*, *pointsize=None*, *stroke=None*, *linewidth=None*, *linestyle=None*, *fillcolor=None*, *fillalpha=None*, *edgewidth=None*, *edgestyle=None*, *edgecolor=None*, *edgealpha=None*, *text=None*, *halign=None*, *valign=None*, *offset=None*, *fontsize=None*, *xmin=None*, *xmax=None*, *ymin=None*, *ymax=None*, *group=None*)

An interface for declaratively specifying statistical graphics.

Plots are constructed by initializing this class and adding one or more layers, comprising a `Mark` and optional `Stat` or `Move`. Additionally, faceting variables or variable pairings may be defined to divide the space into multiple subplots. The mappings from data values to visual properties can be parametrized using scales, although the plot will try to infer good defaults when scales are not explicitly defined.

The constructor accepts a data source (a `pandas.DataFrame` or dictionary with columnar values) and variable assignments. Variables can be passed as keys to the data source or directly as data vectors. If multiple data-containing objects are provided, they will be index-aligned.

The data source and variables defined in the constructor will be used for all layers in the plot, unless overridden or disabled when adding a layer.

The following variables can be defined in the constructor:

x, y, color, alpha, fill, marker, pointsize, stroke, linewidth, linestyle, fillcolor, fillalpha, edgewidth, edgestyle, edgecolor, edgealpha, text, halign, valign, offset, fontsize, xmin, xmax, ymin, ymax, group

The `data`, `x`, and `y` variables can be passed as positional arguments or using keywords. Whether the first positional argument is interpreted as a data source or `x` variable depends on its type.

The methods of this class return a copy of the instance; use chaining to build up a plot through multiple calls. Methods can be called in any order.

Most methods only add information to the plot spec; no actual processing happens until the plot is shown or saved. It is also possible to compile the plot without rendering it to access the lower-level representation.

Methods

Specification methods

`add`	Specify a layer of the visualization in terms of mark and data transform(s).
`scale`	Specify mappings from data units to visual properties.

Subplot methods

`facet`	Produce subplots with conditional subsets of the data.
`pair`	Produce subplots by pairing multiple `x` and/or `y` variables.

Customization methods

`layout`	Control the figure size and layout.
`label`	Control the labels and titles for axes, legends, and subplots.
`limit`	Control the range of visible data.
`share`	Control sharing of axis limits and ticks across subplots.
`theme`	Control the appearance of elements in the plot.

Integration methods

`on`	Provide existing Matplotlib figure or axes for drawing the plot.

Output methods

`plot`	Compile the plot spec and return the Plotter object.
`save`	Compile the plot and write it to a buffer or file on disk.
`show`	Compile the plot and display it by hooking into pyplot.

Configuration

The `Plot` object's default behavior can be configured through its `Plot.config` attribute. Notice that this is a property of the class, not a method on an instance.

Theme configuration

Theme changes made through the the `Plot.config` interface will apply to all subsequent `Plot` instances. Use the `Plot.theme()` method to modify the theme on a plot-by-plot basis.

The theme is a dictionary of matplotlib rc parameters. You can set individual parameters directly:

```
so.Plot.config.theme["axes.facecolor"] = "white"
```

To change the overall style of the plot, update the theme with a dictionary of parameters, perhaps from one of seaborn's theming functions:

```
from seaborn import axes_style
so.Plot.config.theme.update(axes_style("whitegrid"))
```

To sync `Plot` with matplotlib's global state, pass the `rcParams` dictionary:

```
import matplotlib as mpl
so.Plot.config.theme.update(mpl.rcParams)
```

The theme can also be reset back to seaborn defaults:

```
so.Plot.config.theme.reset()
```

Display configuration

When returned from the last statement in a notebook cell, a `Plot` will be compiled and embedded in the notebook as an image. By default, the image is rendered as HiDPI PNG. Alternatively, it is possible to display the plots in SVG format:

```
so.Plot.config.display["format"] = "svg"
```

SVG images use vector graphics with "infinite" resolution, so they will appear crisp at any amount of zoom. The downside is that each plot element is drawn separately, so the image data can get very heavy for certain kinds of plots (e.g., for dense scatterplots).

The HiDPI scaling of the default PNG images will also inflate the size of the notebook they are stored in. (Unlike with SVG, PNG size will scale with the dimensions of the plot but not its complexity). When not useful, it can be disabled:

```
so.Plot.config.display["hidpi"] = False
```

The embedded images are scaled down slightly — independently from the figure size or DPI — so that more information can be presented on the screen. The precise scaling factor is also configurable:

```
so.Plot.config.display["scaling"] = 0.7
```


Created using Sphinx and the PyData Theme.

v0.13.2

Archive