Chapter Seven

Data Visualization using Seaborn

Table of Contents

1. Introduction to Seaborn Package

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Here is some of the functionality that seaborn offers

- A dataset-oriented API for examining relationships between multiple variables
- Specialized support for using categorical variables to show aggregate statistics
- Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
- Automatic estimation and plotting of linear regression models for different kinds dependent variables
- Convenient views onto the overall structure of complex datasets
- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Before going to start we need to import the Seaborn Package by using the below code

```
import seaborn as sns
```

2. Setting and Using of Seaborn Styles

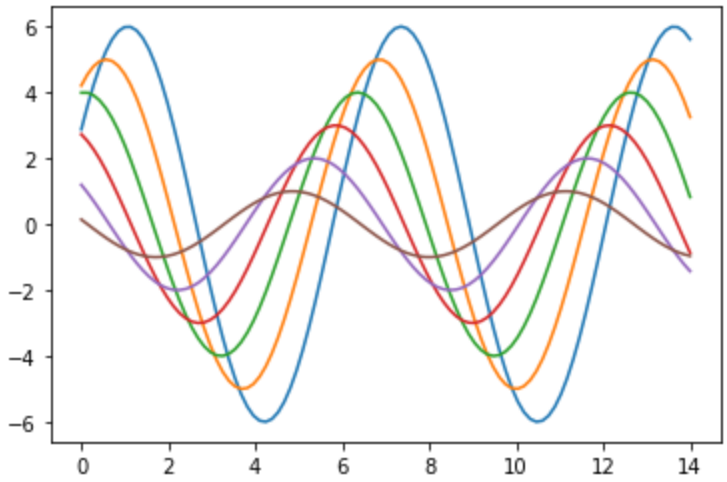
Seaborn comes with a number of customized themes and a high-level interface for controlling the look of matplotlib figures.

Let's define a simple function to plot some offset sine waves, which will help us see the different stylistic parameters we can tweak.

```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)
```

This is what the plot looks like with matplotlib defaults

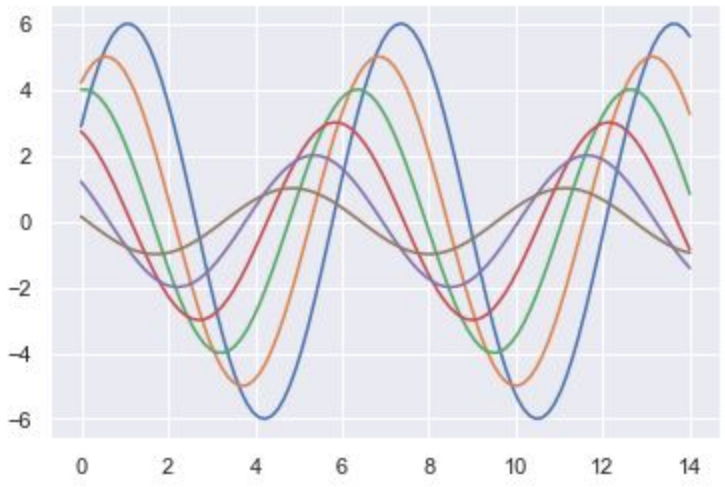
sinplot()



To switch to seaborn defaults, simply call the **set()** function.

sns.set()

sinplot()



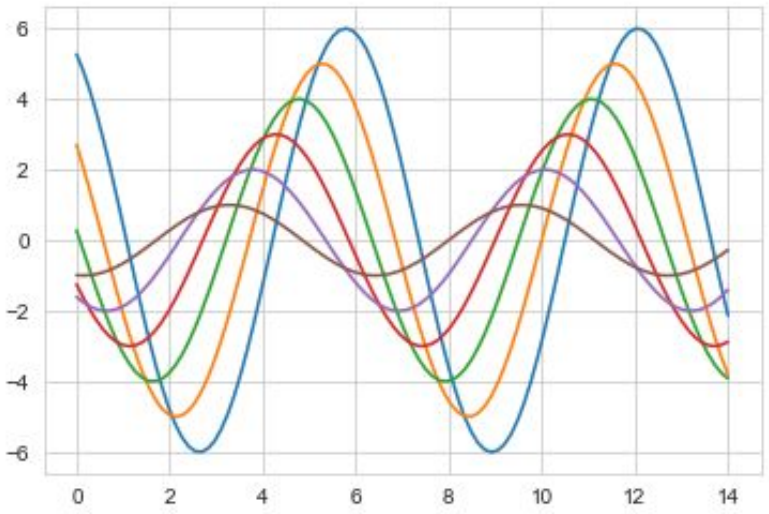
Seaborn figure styles

There are five preset seaborn themes: *darkgrid, whitegrid, dark, white, and ticks*. They are each suited to different applications and personal preferences. The default theme is darkgrid. As mentioned above, the grid helps the plot serve as a lookup table for quantitative information, and the white-on grey helps to keep the grid from competing with lines that represent data. The whitegrid theme is similar, but it is better suited to plots with heavy data elements:

1. **Whitegrid**

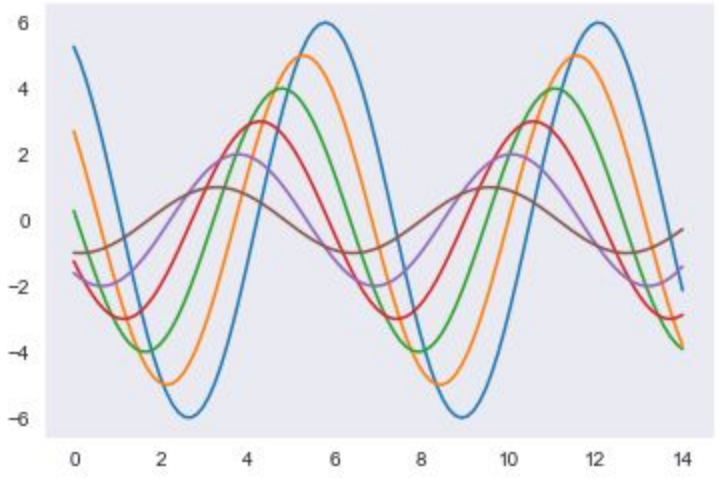
```
sns.set_style("whitegrid")

sinplot()
```

Output:



2. Darkgrid

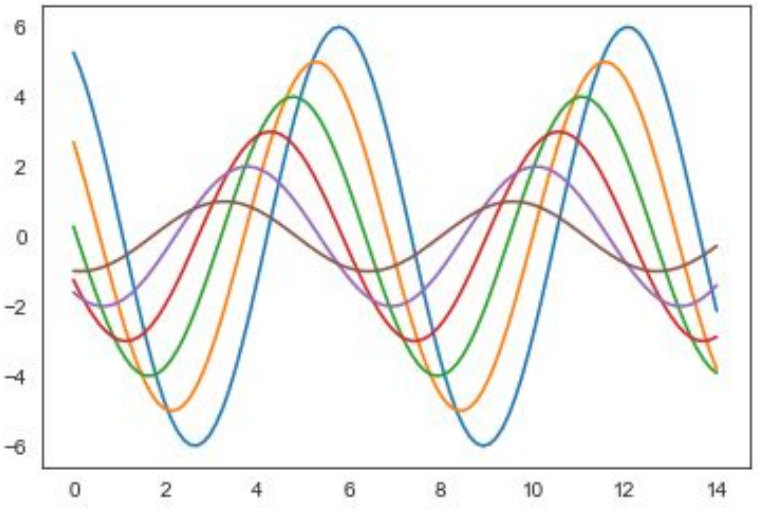
```
sns.set_style("dark")
sinplot()
```



3. White

```
sns.set_style("white")
sinplot()
```

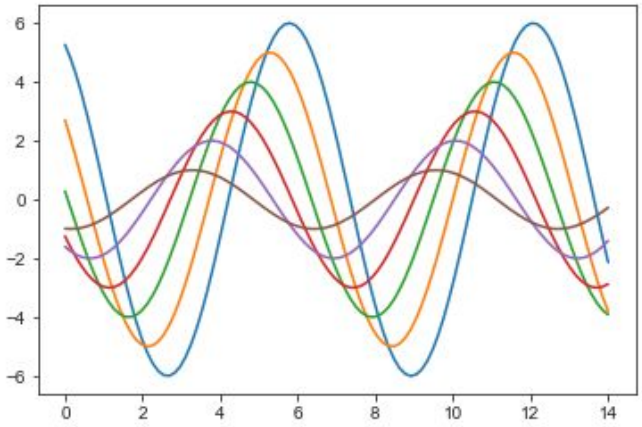
Output:



4. Ticks

```
sns.set_style("ticks")
sinplot()
```

Output:

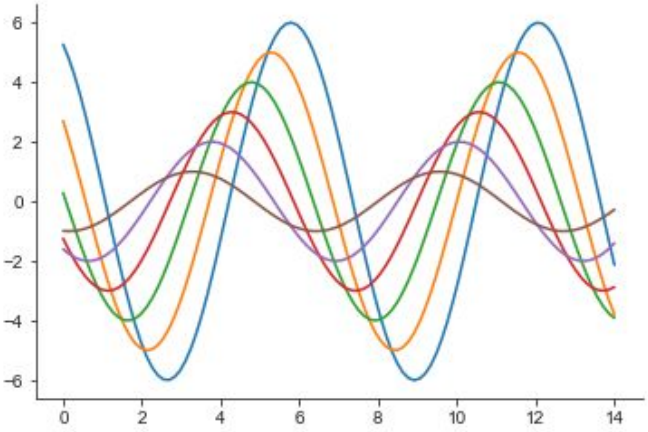


Removing axes spines

Both the white and ticks styles can benefit from removing the top and right axes spines, which are not needed. The seaborn function despine() can be called to remove them:

```
sinplot()

sns.despine()
```

Output:



3. Color Palettes

Color is more important than other aspects of figure style because color can reveal patterns in the data if used effectively or hide those patterns if used poorly. There are a number of great resources to learn about good techniques for using color in visualizations

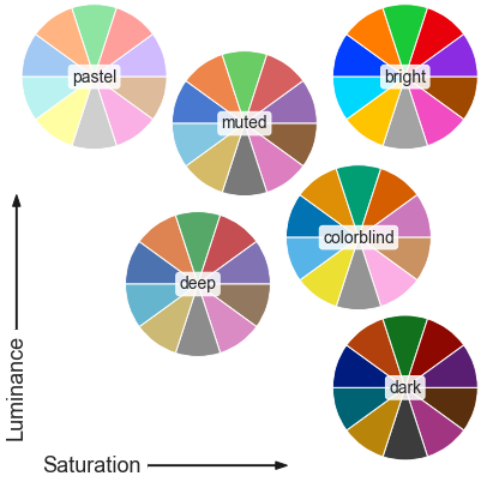
Qualitative color palettes

Qualitative (or categorical) palettes are best when you want to distinguish discrete chunks of data that do not have an inherent ordering.

When importing seaborn, the default color cycle is changed to a set of ten colors that evoke the standard matplotlib color cycle while aiming to be a bit more pleasing to look at.

```
current_palette = sns.color_palette()
sns.palplot(current_palette)
```

There are six variations of the default theme, called ***deep, muted, pastel, bright, dark, and colorblind.***



Using circular color systems

The most common way to do this uses the hls color space, which is a simple transformation of RGB values.

```
sns.palplot(sns.color_palette("hls", 8))
```

There is also the hls_palette() function that lets you control the lightness and saturation of the colors.

```
sns.palplot(sns.hls_palette(8, l=.3, s=.8))
```

To remedy this, seaborn provides an interface to the husl system (since renamed to HSLuv), which also makes it easy to select evenly spaced hues while keeping the apparent brightness and saturation much more uniform

```
sns.palplot(sns.color_palette("husl", 8))
```

Using categorical Colorbrewer palettes

To help you choose palettes from the Color Brewer library, there is the choose_colorbrewer_palette() function.

```
flatui = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c", "#34495e", "#2ecc71"]
sns.palplot(sns.color_palette(flatui))
```

4. Custom sequential palettes

For a simpler interface to custom sequential palettes, you can use light_palette() or dark_palette(), which are both seeded with a single color and produce a palette that ramps either from light or dark desaturated values to that color. These functions are also accompanied by the choose_light_palette() and choose_dark_palette() functions that launch interactive widgets to create these palettes.

```
sns.palplot(sns.light_palette("green"))
```

These palettes can also be reversed.

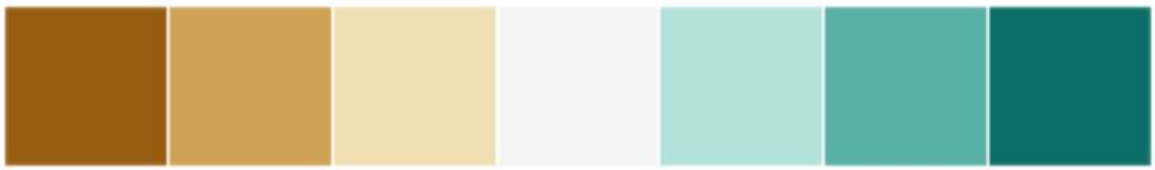
```
sns.palplot(sns.light_palette("navy", reverse=True))
```

Diverging color palettes

The third class of color palettes is called “diverging”. These are used for data where both large low and high values are interesting. There is also usually a well-defined midpoint in the data.

It should not surprise you that the Color Brewer library comes with a set of well-chosen diverging colormaps.

```
sns.palplot(sns.color_palette("BrBG", 7))
```



```
sns.palplot(sns.color_palette("RdBu_r", 7))
```

5. stripplot() and swarmplot()

seaborn.stripplot

seaborn.stripplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, jitter=True, dodge=False, orient=None, color=None, palette=None, size=5, edgecolor='gray', linewidth=0, ax=None, **kwargs)

Draw a scatter plot where one variable is categorical.

A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.

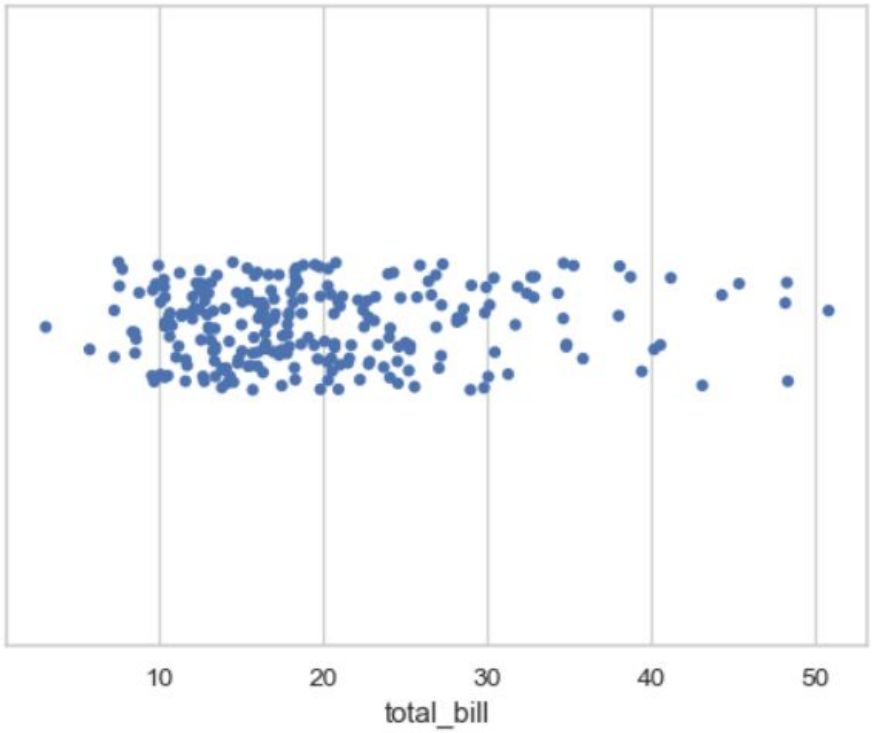
Input data can be passed in a variety of formats, including:

- Vectors of data represented as lists, numpy arrays, or pandas Series objects passed directly to the x, y, and/or hue parameters.
- A “long-form” DataFrame, in which case the x, y, and hue variables will determine how the data are plotted.
- A “wide-form” DataFrame, such that each numeric column will be plotted.
- An array or list of vectors.

Examples

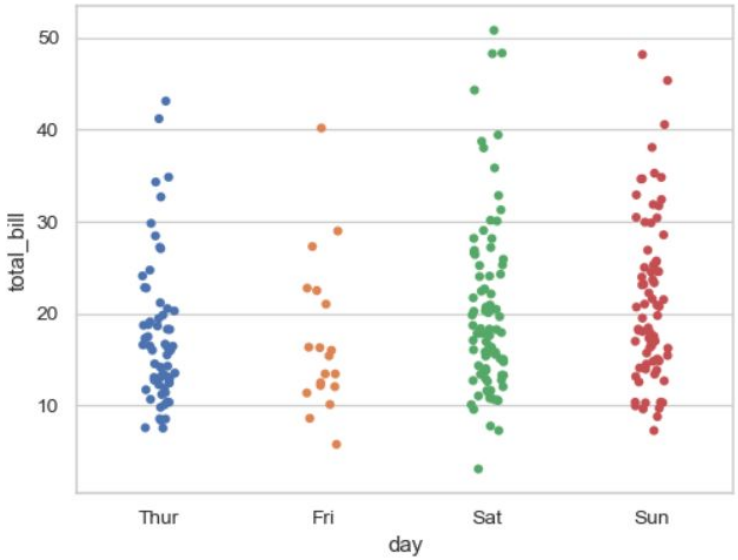
Draw a single horizontal strip plot:

```
>>> import seaborn as sns
>>> sns.set(style="whitegrid")
>>> tips = sns.load_dataset("tips")
>>> ax = sns.stripplot(x=tips["total_bill"])
```



Group the strips by a categorical variable:

```
>>> ax = sns.stripplot(x="day", y="total_bill", data=tips)
```



seaborn.swarmplot

seaborn.swarmplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, dodge=False, orient=None, color=None, palette=None, size=5, edgecolor='gray', linewidth=0, ax=None, **kwargs)

This function is similar to stripplot(), but the points are adjusted (only along the categorical axis) so that they don't overlap. This gives a better representation of the distribution of values, but it does not scale well to large numbers of observations. This style of plot is sometimes called a "beeswarm".

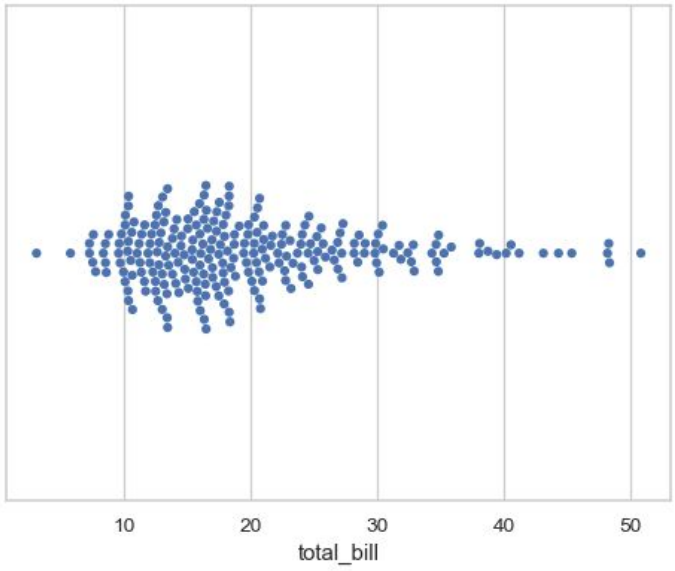
A swarm plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.

Examples

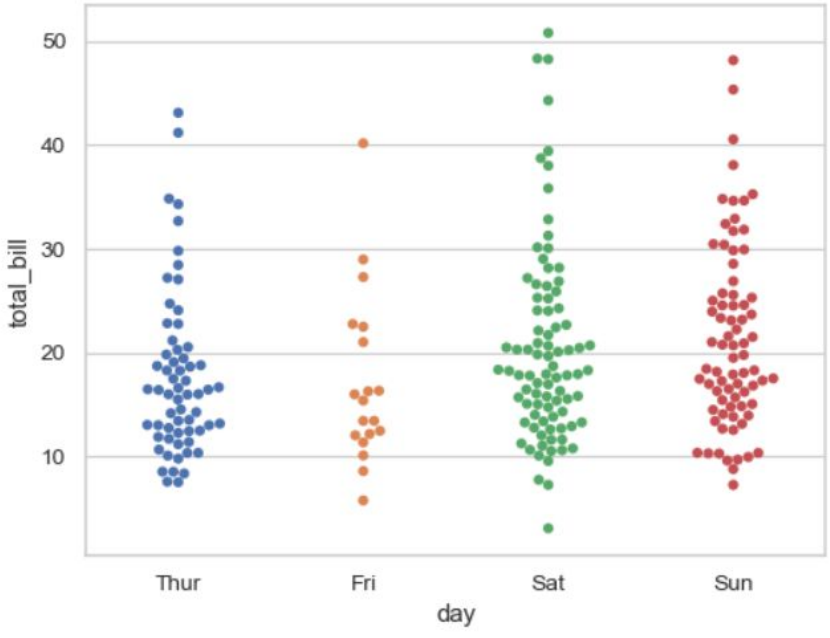
Draw a single horizontal swarm plot:

```
>>> import seaborn as sns
>>> sns.set(style="whitegrid")
>>> tips = sns.load_dataset("tips")
>>> ax = sns.swarmplot(x=tips["total_bill"])
```

Output:




```
>>> ax = sns.swarmplot(x="day", y="total_bill", data=tips)
```



6. boxplot(), violinplot() and lmplot()

seaborn.boxplot

seaborn.boxplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, dodge=True, fliersize=5, linewidth=None, whis=1.5, ax=None, **kwargs)

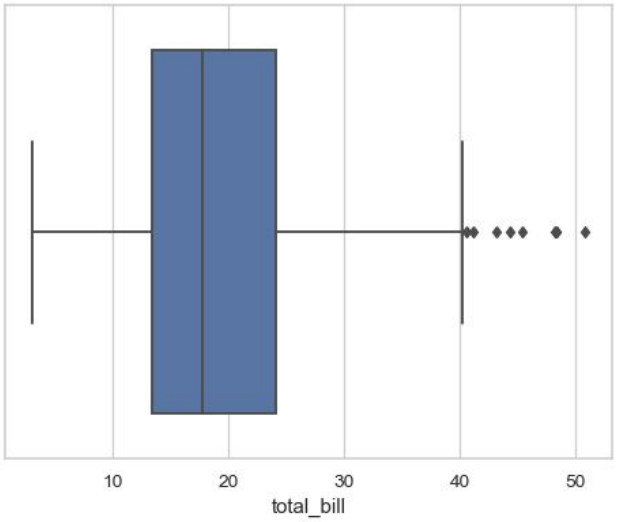
Draw a box plot to show distributions with respect to categories.

A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

Examples

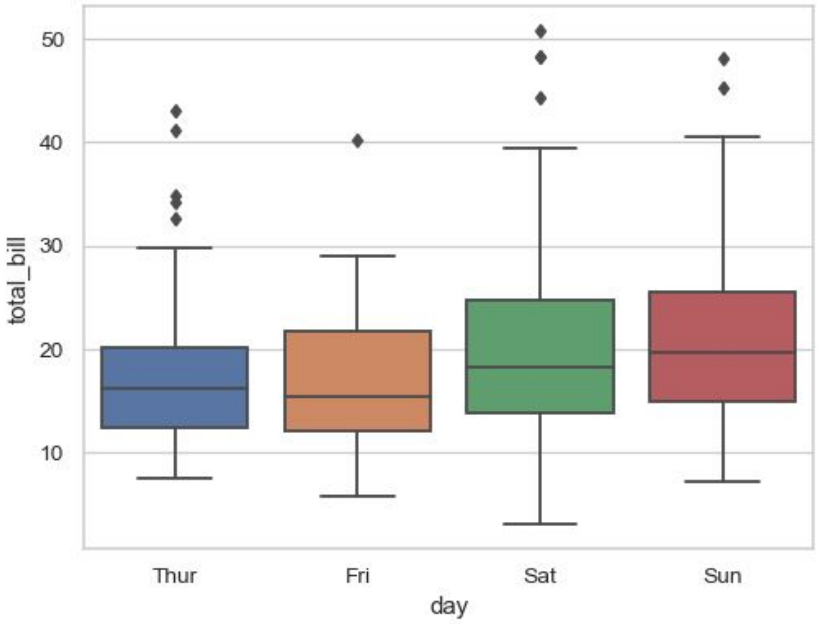
Draw a single horizontal boxplot:

```
>>> import seaborn as sns
>>> sns.set(style="whitegrid")
>>> tips = sns.load_dataset("tips")
>>> ax = sns.boxplot(x=tips["total_bill"])
```



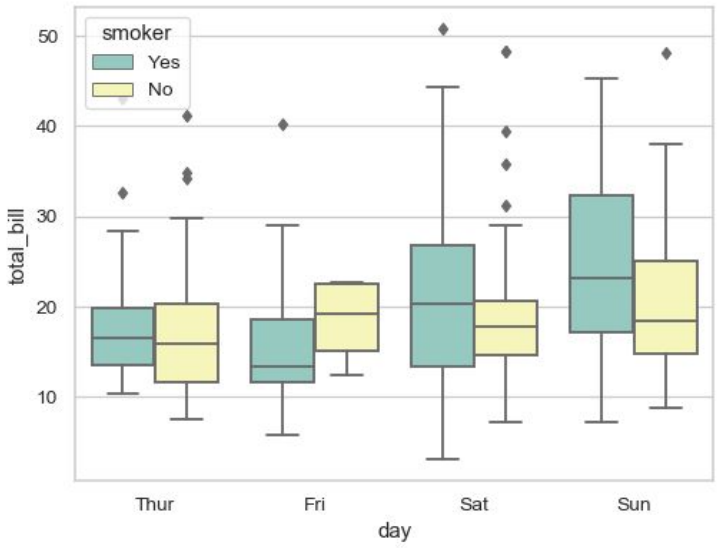
Draw a vertical boxplot grouped by a categorical variable:

```
>>> ax = sns.boxplot(x="day", y="total_bill", data=tips)
```



Draw a boxplot with nested grouping by two categorical variables:

```
>>> ax = sns.boxplot(x="day", y="total_bill", hue="smoker",
...                  data=tips, palette="Set3")
```



seaborn.violinplot

seaborn.violinplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, bw='scott', cut=2, scale='area', scale_hue=True, gridsize=100, width=0.8, inner='box', split=False, dodge=True, orient=None, linewidth=None, color=None, palette=None, saturation=0.75, ax=None, **kwargs)

Draw a combination of boxplot and kernel density estimate.

A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those

distributions can be compared. Unlike a box plot, in which all of the plot components correspond to actual datapoints, the violin plot features a kernel density estimation of the underlying distribution.

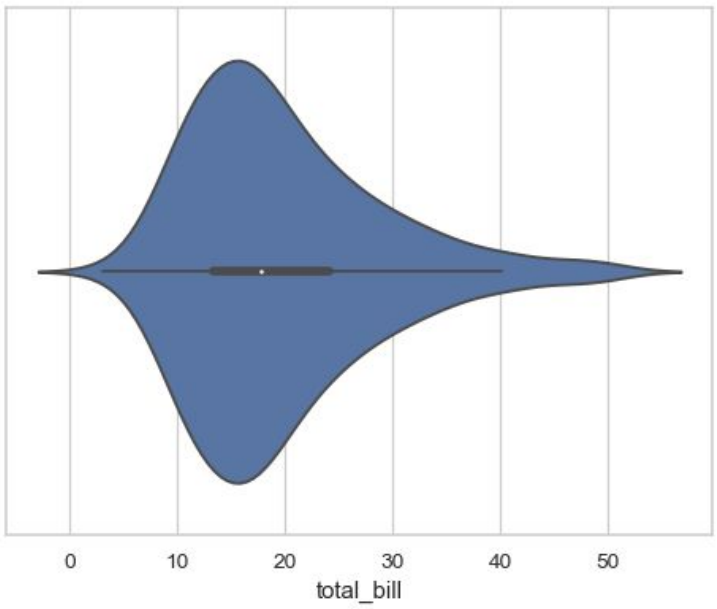
This can be an effective and attractive way to show multiple distributions of data at once, but keep in mind that the estimation procedure is influenced by the sample size, and violins for relatively small samples might look misleadingly smooth.

Examples

Draw a single horizontal violinplot:

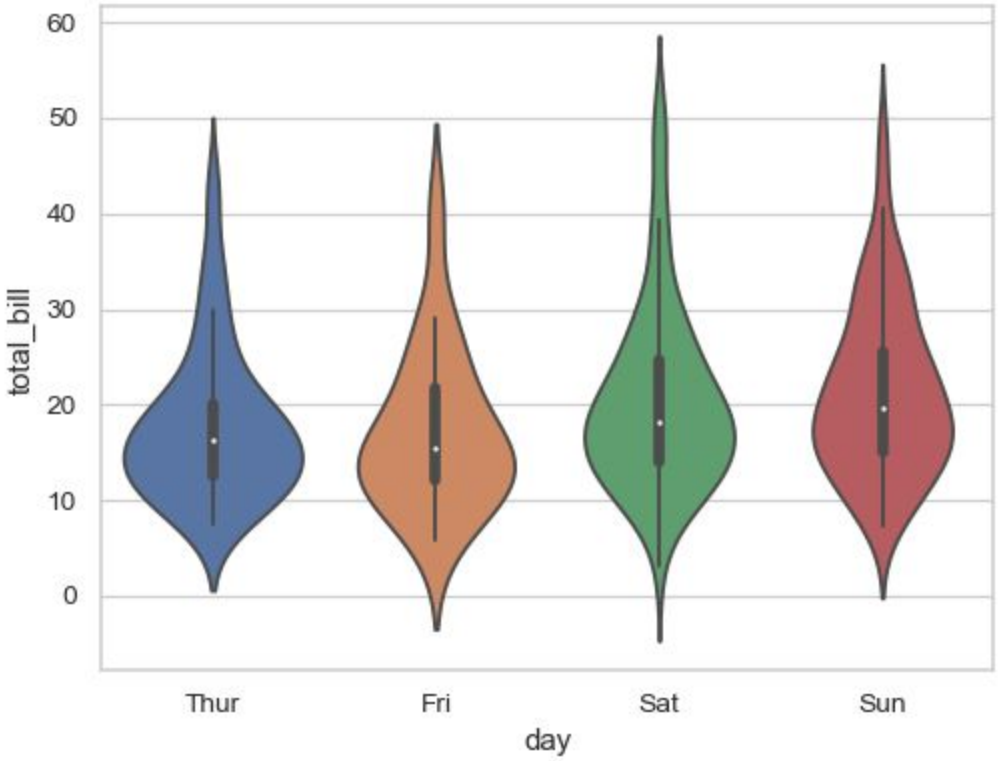
```
>>> import seaborn as sns
>>> sns.set(style="whitegrid")
>>> tips = sns.load_dataset("tips")
>>> ax = sns.violinplot(x=tips["total_bill"])
```

Output:



Draw a vertical violinplot grouped by a categorical variable:

```
>>> ax = sns.violinplot(x="day", y="total_bill", data=tips)
```



seaborn.lmplot

seaborn.lmplot(x, y, data, hue=None, col=None, row=None, palette=None, col_wrap=None, height=5, aspect=1, markers='o', sharex=True, sharey=True, hue_order=None, col_order=None, row_order=None, legend=True, legend_out=True, x_estimator=None, x_bins=None, x_ci='ci', scatter=True, fit_reg=True, ci=95, n_boot=1000, units=None, seed=None, order=1, logistic=False, lowess=False, robust=False, logx=False, x_partial=None, y_partial=None, truncate=True, x_jitter=None, y_jitter=None, scatter_kws=None, line_kws=None, size=None)

Plot data and regression model fits across FacetGrid.

This function combines regplot() and FacetGrid. It is intended as a convenient interface to fit regression models across conditional subsets of a dataset.

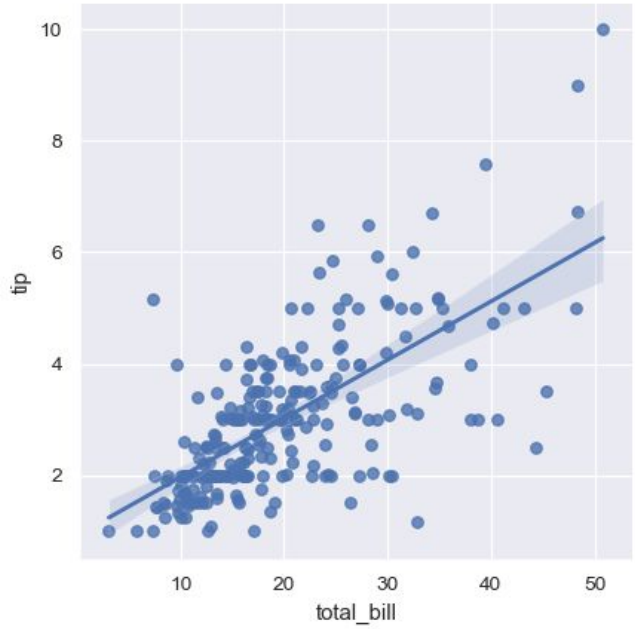
When thinking about how to assign variables to different facets, a general rule is that it makes sense to use hue for the most important comparison, followed by col and row.

Examples

These examples focus on basic regression model plots to exhibit the various faceting options; see the **regplot**() docs for demonstrations of the other options for plotting the data and models. There are also other examples for how to manipulate plot using the returned object on the FacetGrid docs.

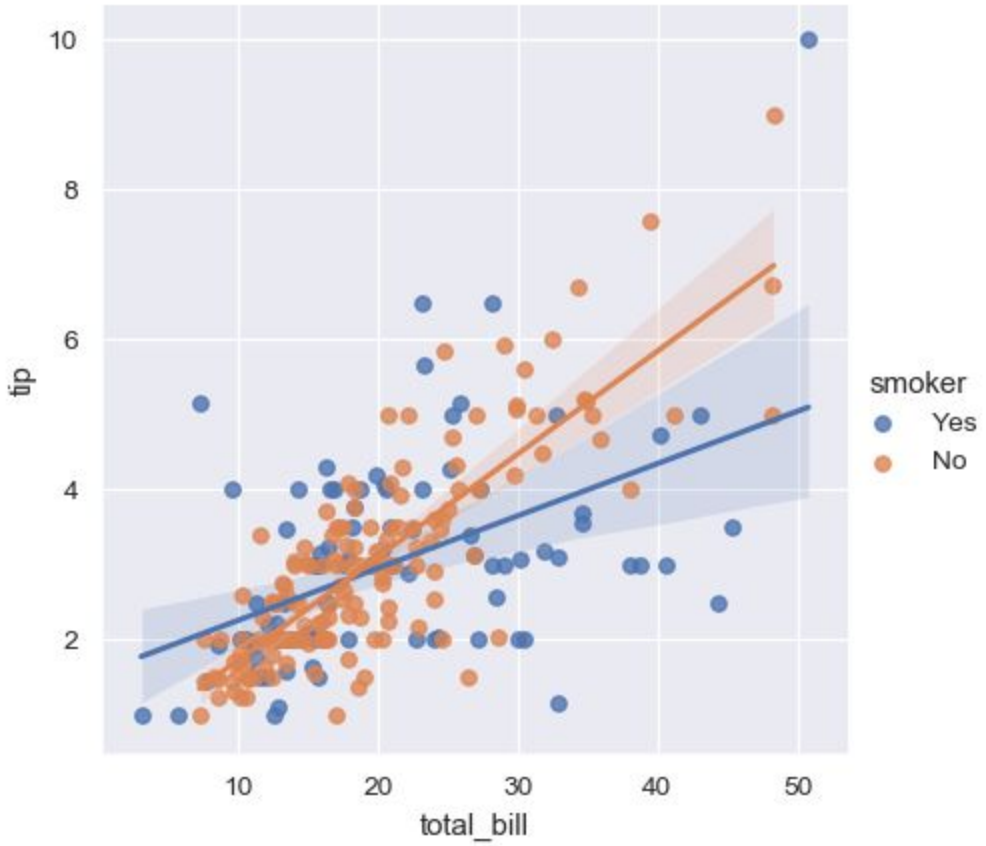
Plot a simple linear relationship between two variables:

```
>>> import seaborn as sns; sns.set(color_codes=True)
>>> tips = sns.load_dataset("tips")
>>> g = sns.lmplot(x="total_bill", y="tip", data=tips)
```



Condition on a third variable and plot the levels in different colors:

```
>>> g = sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips)
```



7. barplot(), pointplot() and countplot()

seaborn.barplot

seaborn.barplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, estimator=<function mean at 0x105c7d9e0>, ci=95, n_boot=1000, units=None, seed=None, orient=None, color=None, palette=None, saturation=0.75, errcolor='.26', errwidth=None, capsize=None, dodge=True, ax=None, **kwargs)

Show point estimates and confidence intervals as rectangular bars.

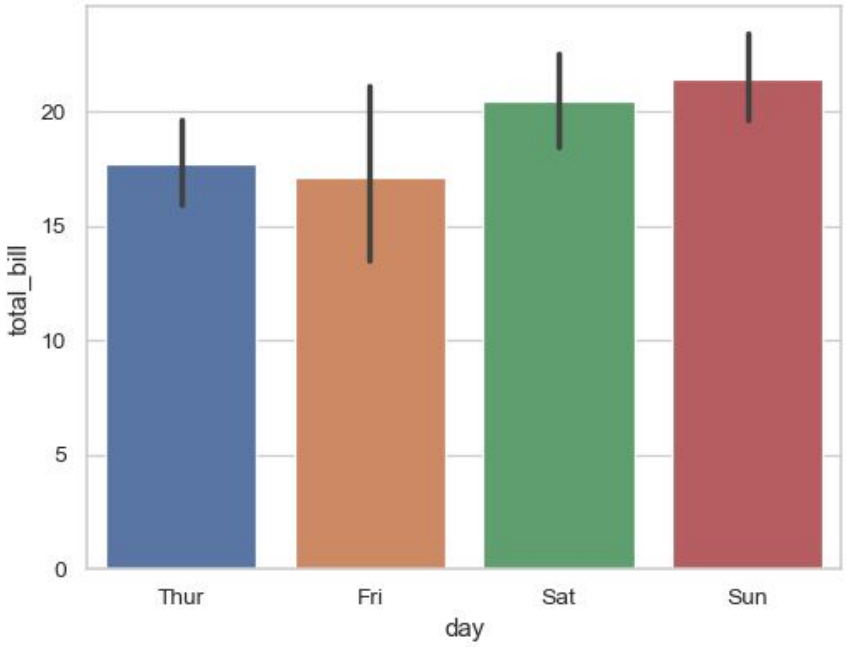
A bar plot represents an estimate of central tendency for a numeric variable with the height of each rectangle and provides some indication of the uncertainty around that estimate using error bars. Bar plots include 0 in the quantitative axis range, and they are a good choice when 0 is a meaningful value for the quantitative variable, and you want to make comparisons against it.

For datasets where 0 is not a meaningful value, a point plot will allow you to focus on differences between levels of one or more categorical variables.
It is also important to keep in mind that a bar plot shows only the mean (or other estimator) value, but in many cases it may be more informative to show the distribution of values at each level of the categorical variables. In that case, other approaches such as a box or violin plot may be more appropriate.

Examples

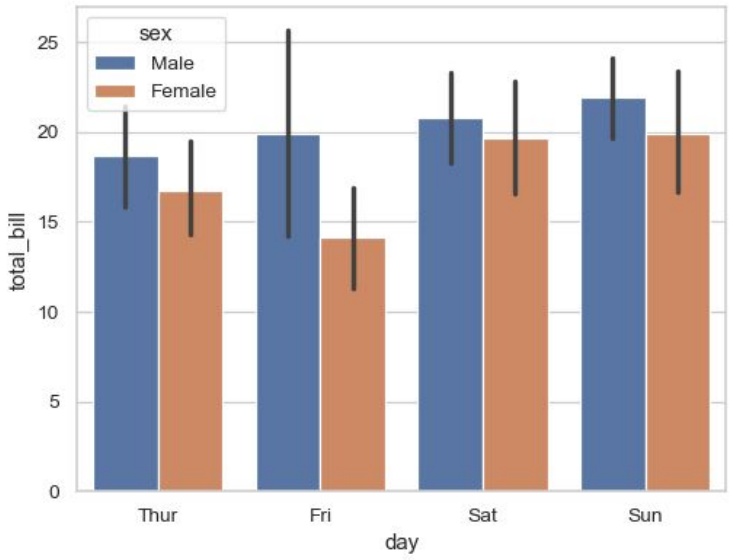
Draw a set of vertical bar plots grouped by a categorical variable:

```
>>> import seaborn as sns
>>> sns.set(style="whitegrid")
>>> tips = sns.load_dataset("tips")
>>> ax = sns.barplot(x="day", y="total_bill", data=tips)
```



Draw a set of vertical bars with nested grouping by a two variables:

```
>>> ax = sns.barplot(x="day", y="total_bill", hue="sex", data=tips)
```



seaborn.pointplot

seaborn.pointplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, estimator=<function mean at 0x105c7d9e0>, ci=95, n_boot=1000, units=None, seed=None, markers='o', linestyles='-', dodge=False, join=True, scale=1, orient=None, color=None, palette=None, errwidth=None, capsize=None, ax=None, **kwargs)

Show point estimates and confidence intervals using scatter plot glyphs.

A point plot represents an estimate of central tendency for a numeric variable by the position of scatter plot points and provides some indication of the uncertainty around that estimate using error bars.

Point plots can be more useful than bar plots for focusing comparisons between different levels of one or more categorical variables. They are particularly adept at showing interactions: how the relationship between levels of one categorical variable changes across levels of a second categorical variable. The lines that join each point from the same hue level allow interactions to be judged by differences in slope, which is easier for the eyes than comparing the heights of several groups of points or bars.

Examples

Draw a set of vertical point plots grouped by a categorical variable:

```
>>> import seaborn as sns
>>> sns.set(style="darkgrid")
>>> tips = sns.load_dataset("tips")
>>> ax = sns.pointplot(x="time", y="total_bill", data=tips)
```

Draw a set of vertical points with nested grouping by a two variables:

```
>>> ax = sns.pointplot(x="time", y="total_bill", hue="smoker",
...                    data=tips)
```

seaborn.countplot

seaborn.countplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, dodge=True, ax=None, **kwargs)

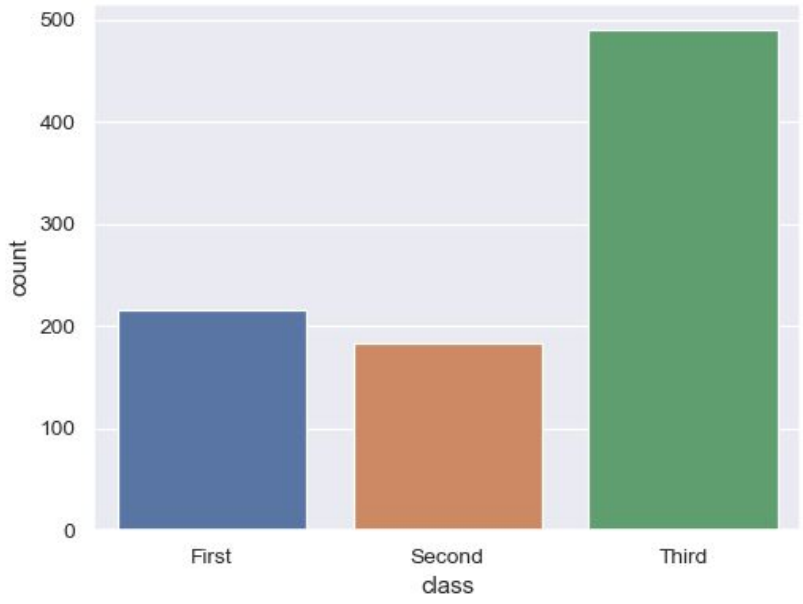
Show the counts of observations in each categorical bin using bars.

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot(), so you can compare counts across nested variables.

Examples

Show value counts for a single categorical variable:

```
>>> import seaborn as sns
>>> sns.set(style="darkgrid")
>>> titanic = sns.load_dataset("titanic")
>>> ax = sns.countplot(x="class", data=titanic)
```



8. Regression Plot

seaborn.regplot(x, y, data=None, x_estimator=None, x_bins=None, x_ci='ci', scatter=True, fit_reg=True, ci=95, n_boot=1000, units=None, seed=None, order=1, logistic=False, lowess=False, robust=False, logx=False, x_partial=None, y_partial=None, truncate=True, dropna=True, x_jitter=None, y_jitter=None, label=None, color=None, marker='o', scatter_kws=None, line_kws=None, ax=None)

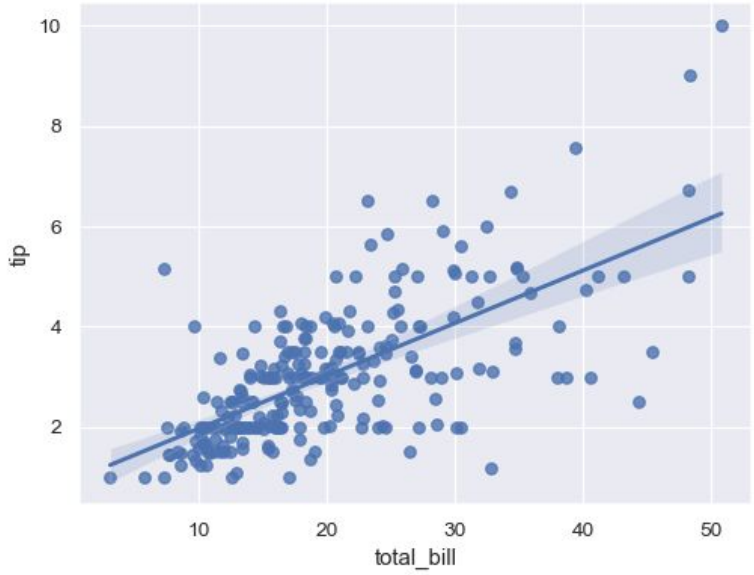
Plot data and a linear regression model fit.

There are a number of mutually exclusive options for estimating the regression model.

Examples

Plot the relationship between two variables in a DataFrame:

```
>>> import seaborn as sns; sns.set(color_codes=True)
>>> tips = sns.load_dataset("tips")
>>> ax = sns.regplot(x="total_bill", y="tip", data=tips)
```



9. seaborn.heatmap

seaborn.heatmap(data, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt='.2g', annot_kws=None, linewidths=0, linecolor='white', cbar=True, cbar_kws=None, cbar_ax=None, square=False, xticklabels='auto', yticklabels='auto', mask=None, ax=None, **kwargs)

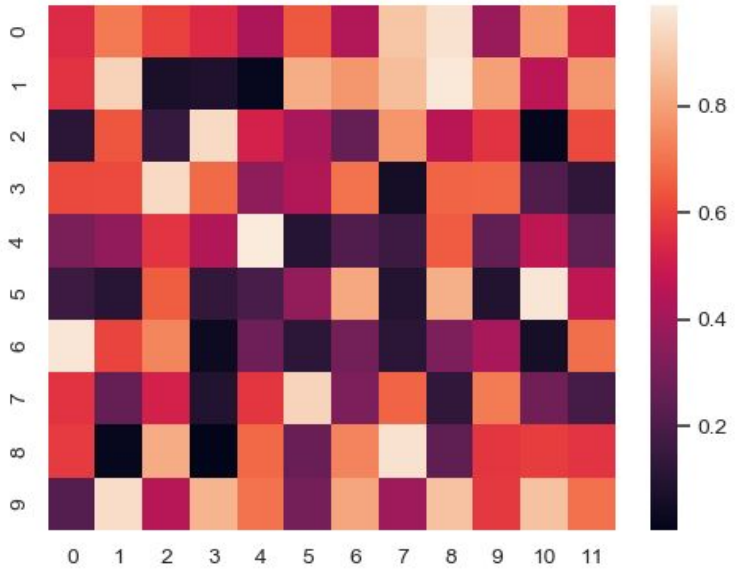
Plot rectangular data as a color-encoded matrix.

This is an Axes-level function and will draw the heatmap into the currently-active Axes if none is provided to the ax argument. Part of this Axes space will be taken and used to plot a colormap, unless cbar is False or a separate Axes is provided to cbar_ax.

Examples

Plot a heatmap for a numpy array:

```
>>> import numpy as np; np.random.seed(0)
>>> import seaborn as sns; sns.set()
>>> uniform_data = np.random.rand(10, 12)
>>> ax = sns.heatmap(uniform_data)
```



Plot a heatmap for data centered on 0 with a diverging colormap:

```
>>> normal_data = np.random.randn(10, 12)
>>> ax = sns.heatmap(normal_data, center=0)
```