Visualization

November 22, 2020

1 Data visualization

There are multiple libraries for data visualization in *python*.

In this lecture, we will focus on the most famous and used, *matplotlib*, plus *seaborn* that is built on-top of it.

While these libraries are general enough, others like *plotly*, *geopandas* and *Bokeh* may come handy for specific tasks.

1.1 *matplotlib*

matplotlib is a **plotting library** that produces figures in a variety of hardcopy formats and interactive environments.

It has many kind of **heavily customizable** plots: **line plots, bar plots**, **stacked bar plots**, **scatter plots**, **histograms** and more.

matplotlib can handle categorical data, timestamps and other data types.

1.1.1 Import convention

Core plot functions are in the *.pyplot* subpackage that is conventionally imported as *plt*.

[1]:
```
import matplotlib.pyplot as plt
```

Jupyter magic *IPython* (*Interactive Python*, the shell that powers Jupyter kernels and offers support for interactive data visualization) provides *magic* commands that can be triggered with %.

From the docs:

```
With the following backend, the output of plotting commands is displayed inline within frontend
The resulting plots will then also be stored in the notebook document.
```

[2]:
```
%matplotlib inline
```

1.2 Anatomy of a figure

Let's begin by inspecting the anatomy of a figure to better undestand the names of each element and what we are doing.

Each of these elements is called an ***Artist***.

There are *Artist*s for the axes, for the labels, for the plots, etc.

A ***Figure*** represents the figure as whole.

Axes is the region of the image with the data space. An *Axes* contains two (or three) *Axis*.

Axis objects are the axis of the figure.

[3]:
```
plt.rcParams['figure.figsize'] = [6, 4]
plt.rcParams['figure.dpi'] = 150
```

1.3 Create a new figure

We can create a new figure with the *.figure()* method.

This step is not mandatory and, if you don't instantiate a new figure, one will be created with the default parameters.

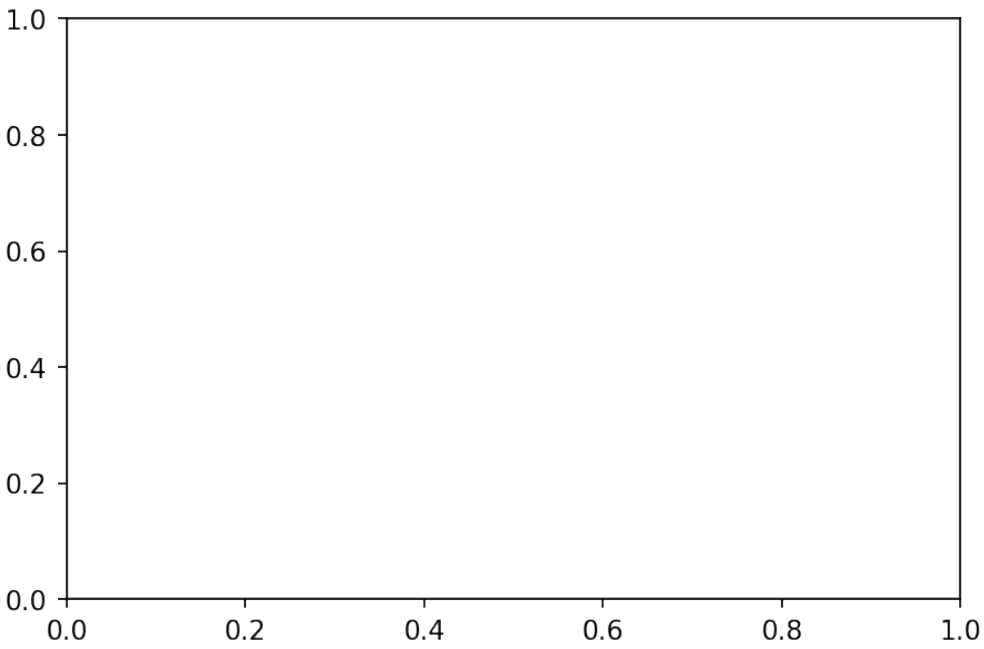
[4]:
```
# an empty figure with no Axes
fig = plt.figure(figsize=(10,10), dpi=300)
```

```
<Figure size 3000x3000 with 0 Axes>
```

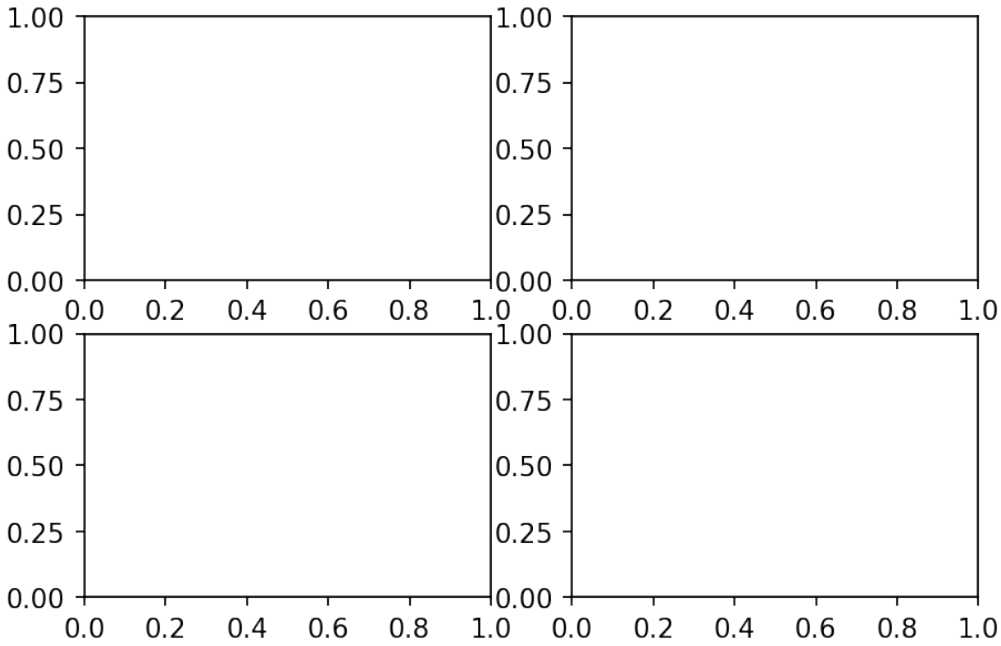
Figure with a single Axes The *.subplots()* function creates, in a single call, a figure and a set of subplots.

You can provide the number of rows and columns in the plot.

[5]:
```
# a figure with a single Axes
fig, ax = plt.subplots()
```



[6]:
```
# a figure with a 2x2 grid of Axes
fig, axs = plt.subplots(2, 2)
```

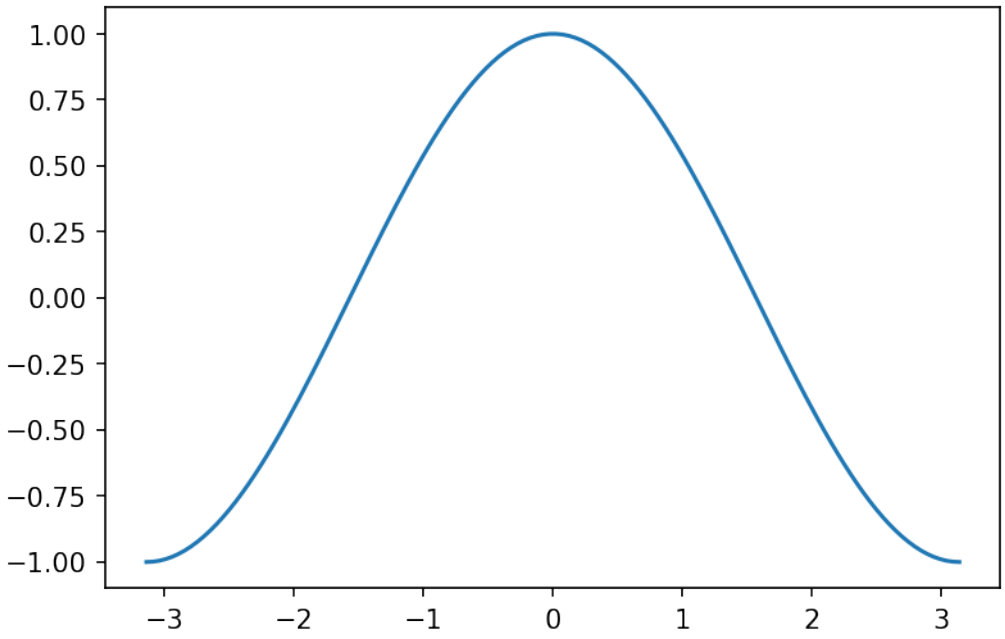


1.3.1 Plotting examples

After getting familiar with the names and with figure creation, let's move to the actual plotting.

[7]:
```
import numpy as np

X = np.linspace(-np.pi, np.pi, 128)
C = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, C)
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f2c45da6c18>]
```

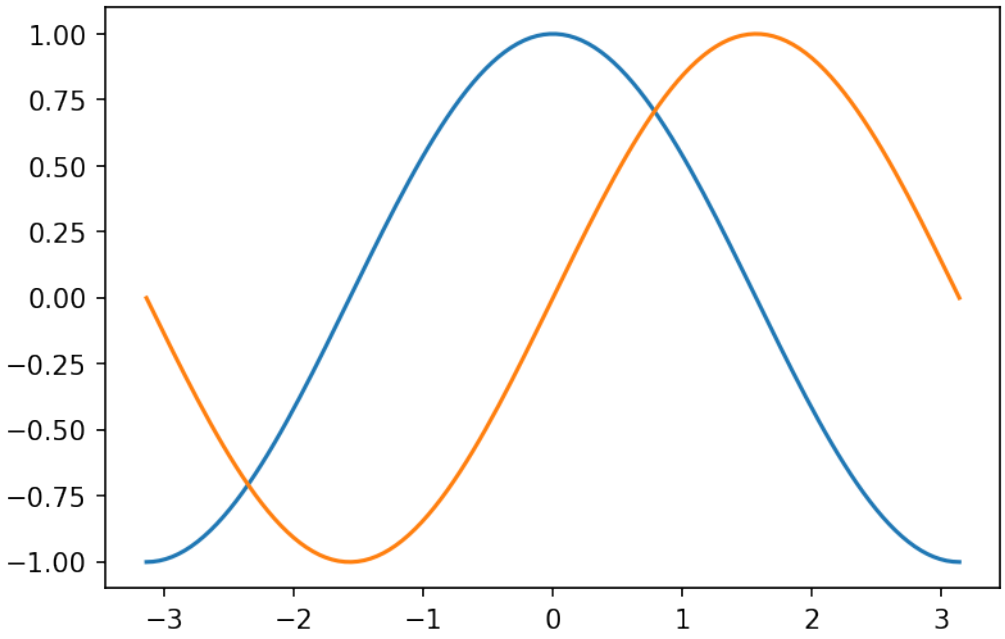


1.3.2 Multiple plots on the same *Axes*

How can we add a second plot with the *sin* function?

We just plot on the same *Axes* multiple times.

[8]:
```
S = np.sin(X)

ax.plot(X, S)

display(fig)
```



1.3.3 Setting the *ticks*

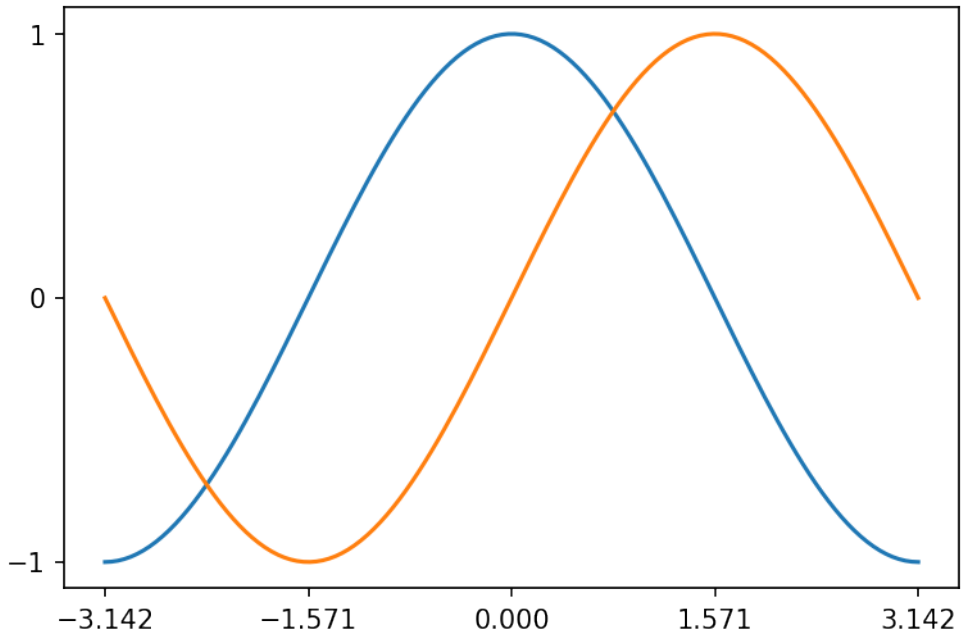
Ticks are placed automatically and this *automagical* placement usually works very well.

However, if you want to setup *ticks*, you can:

- Setup them manually (i.e., providing values where to place ticks)
- Setup a ***Locator***
  - ***Locators*** define the placement of ticks according to some rule.
  - This placement is performed by the ***AutoLocator()*** *Locator*.

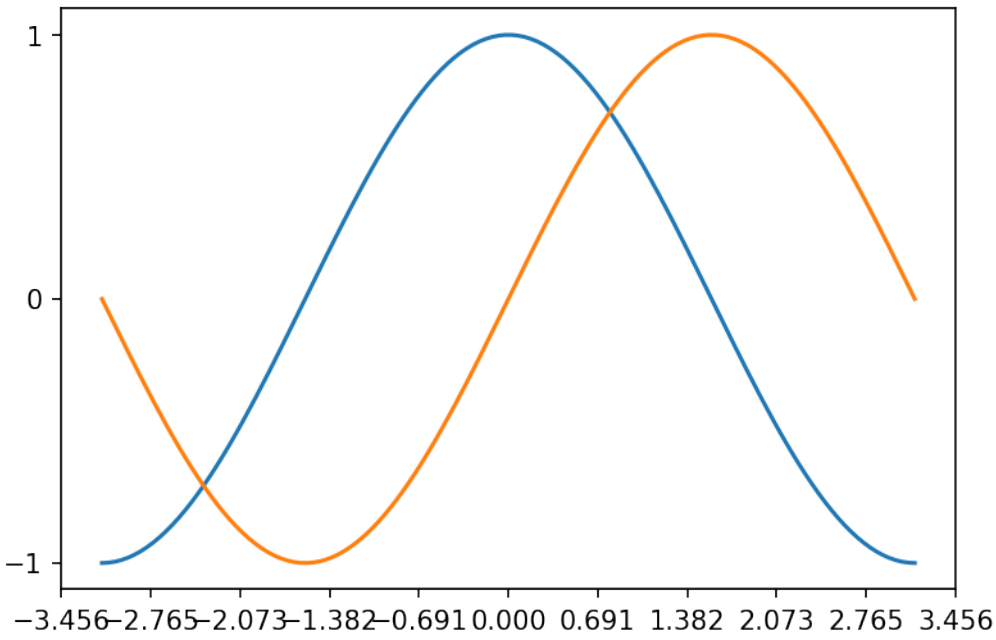
Setting *ticks* manually

[9]:
```
# Setting up on the axis
ax.xaxis.set_ticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
ax.yaxis.set_ticks([-1, 0, +1])

# # Setting up on the plot
# plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
# plt.yticks([-1, 0, +1])

display(fig)
```



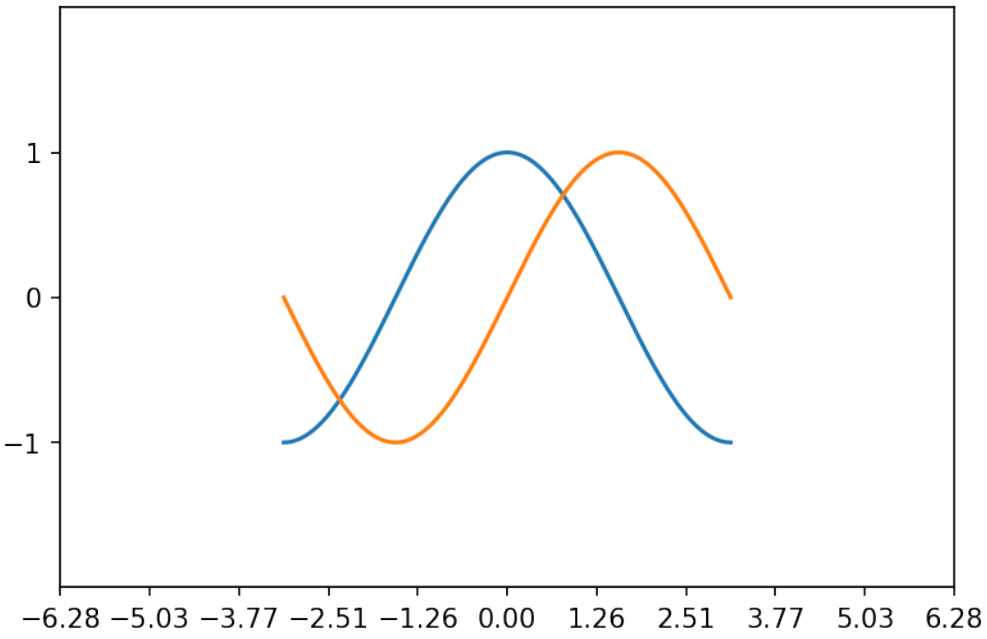
Setting a Locator instead

[10]:
```
from matplotlib.ticker import LinearLocator

ax.xaxis.set_major_locator(LinearLocator())

display(fig)
```



Setting up axis limits You may need to setup the limits of the *Axis*.

[11]:
```
# plt.xlim(X.min() * 1.1, X.max() * 1.1)
# plt.ylim(C.min() * 1.1, C.max() * 1.1)

ax.set_xlim(X.min() * 2, X.max() * 2)
ax.set_ylim(C.min() * 2, C.max() * 2)

display(fig)
```

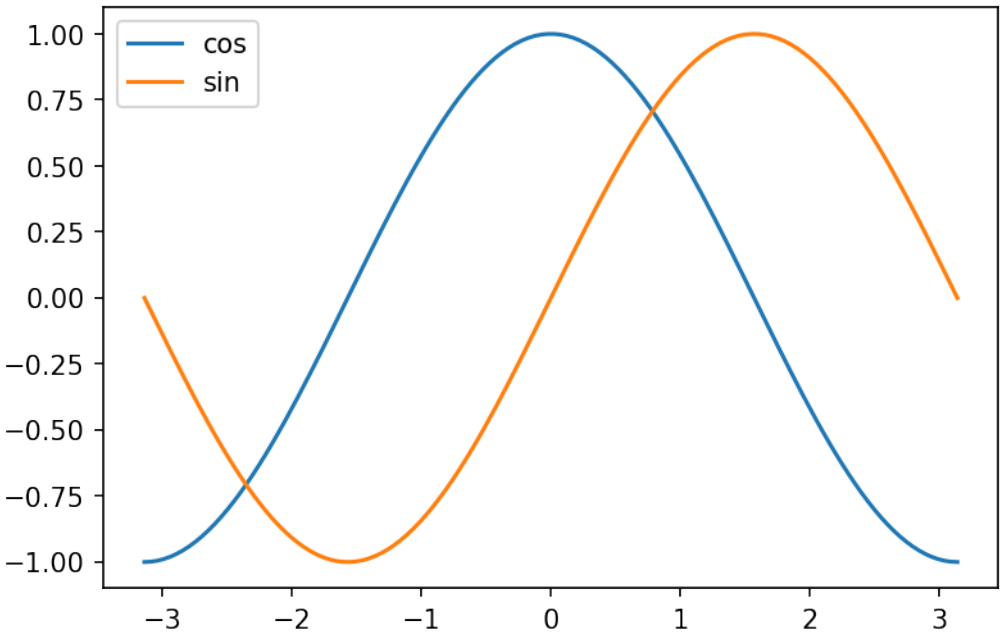


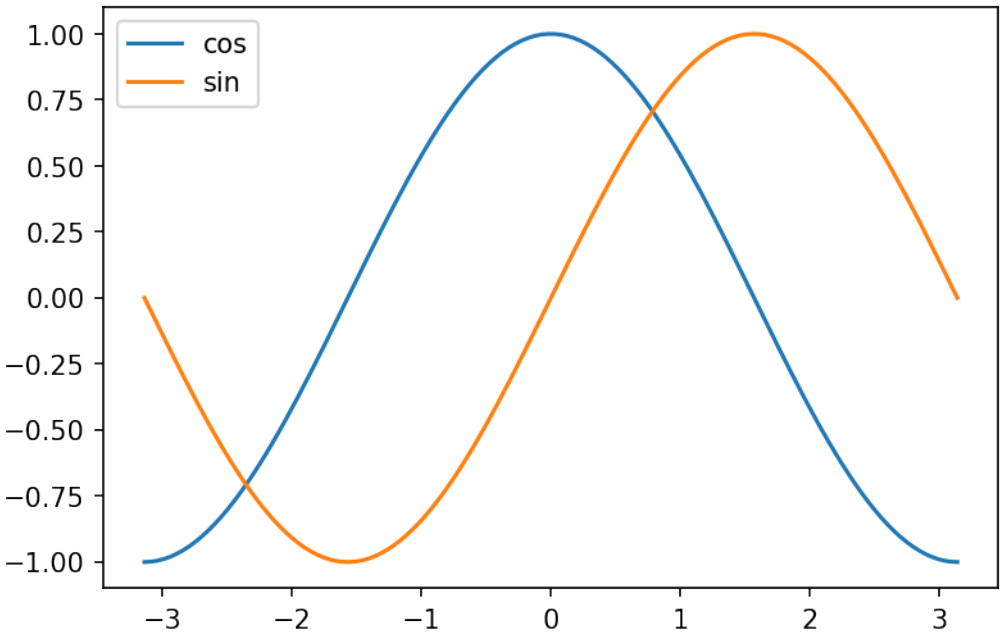
1.3.4 Adding a legend

You can easily add a legend to your plot by setting a label for each sub-plot and calling the *.legend()* method.

[12]:
```
fig, ax = plt.subplots()

ax.plot(X, C, label="cos")
ax.plot(X, S, label="sin")

ax.legend(loc='best')
# fig.legend()

# plt.plot(X, C, label="cos")
```

```
# plt.plot(X, S, label="sin")
# plt.legend()

# ax.legend()

display(fig)
```

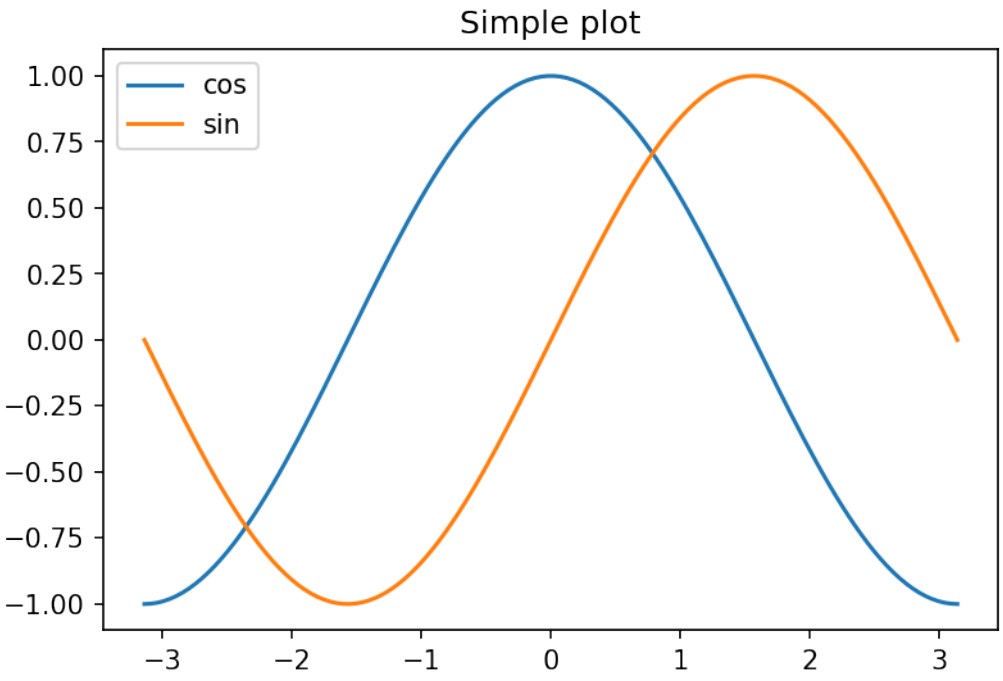




1.3.5 Setting the labels

We can set the *Axis* labels and Figure title as follows.

[13]:
```
# plt.xlabel('x label')
# plt.ylabel('y label')
# plt.title("Simple Plot")

ax.xaxis.set_label('x label')
ax.yaxis.set_label('y label')
ax.set_title("Simple plot")
display(fig)
```



1.3.6 Plot types

Line plots

`Line plot is a type of chart that displays a series of data points called "markers" connected l`

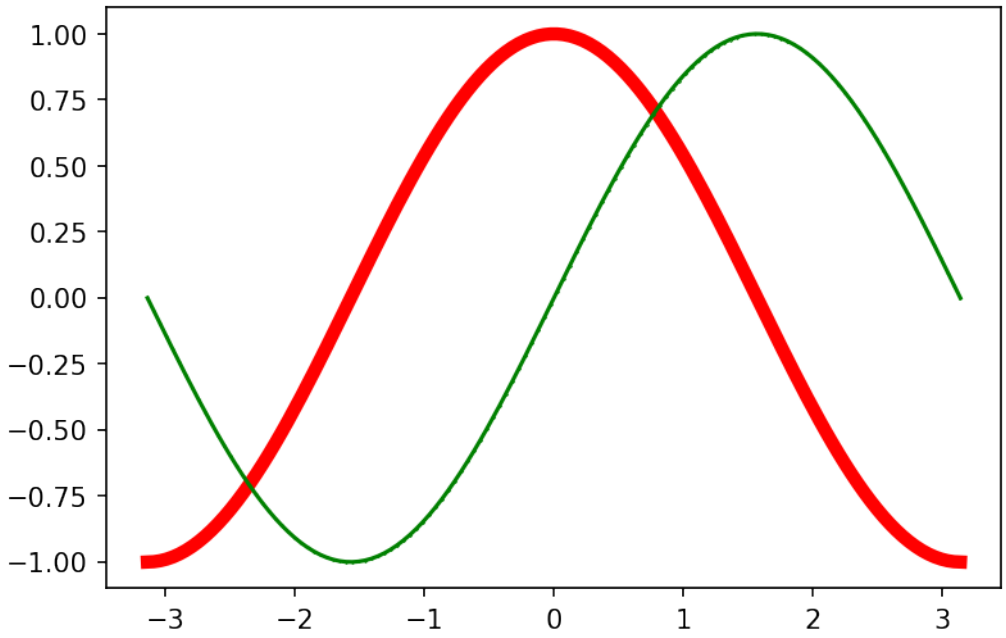
The associated method is *.plot* and it is highly customizable.

For an extensive list of properties, markers and styles, visit the documentation!

As an example, you can configure line color, width and markers.

[14]:
```
plt.plot(X, C, linewidth=5, color="red")
plt.plot(X, S, marker="D", markersize=0.1, color="green")
```

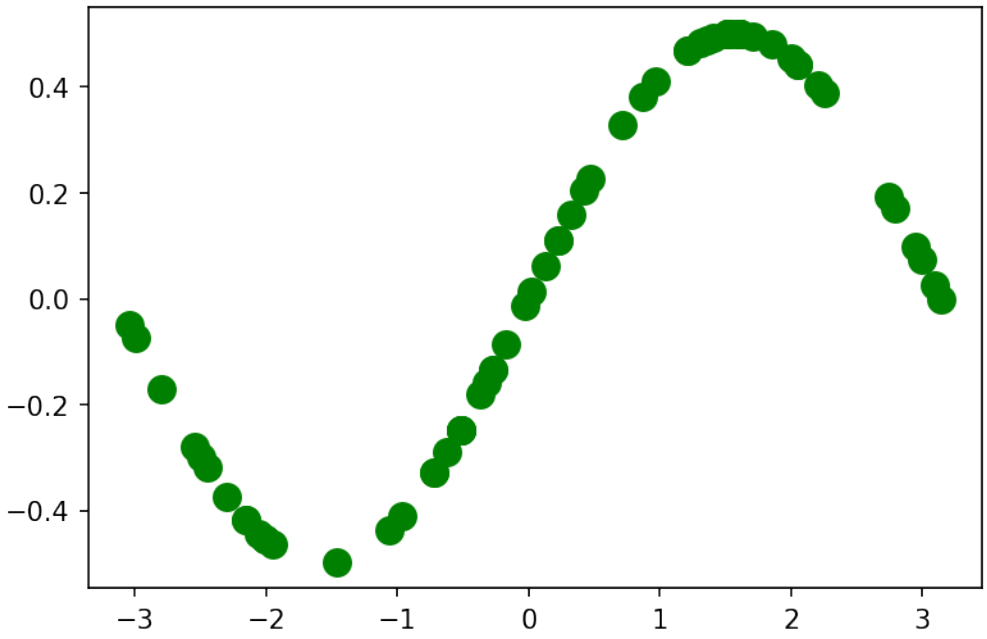
[14]:
```
[<matplotlib.lines.Line2D at 0x7f2c45ac2c50>]
```



You can also plot only the markers.

[15]:
```
indices = np.random.choice(list(range(X.shape[0])), size=64)
plt.plot(X[indices], (S/2)[indices], marker="o", linewidth=0, color="green",␣
↪markersize=10)
```

[15]:
```
[<matplotlib.lines.Line2D at 0x7f2c459f66a0>]
```



Scatter plots A scatter plot is a type of plot that displays values as markers.

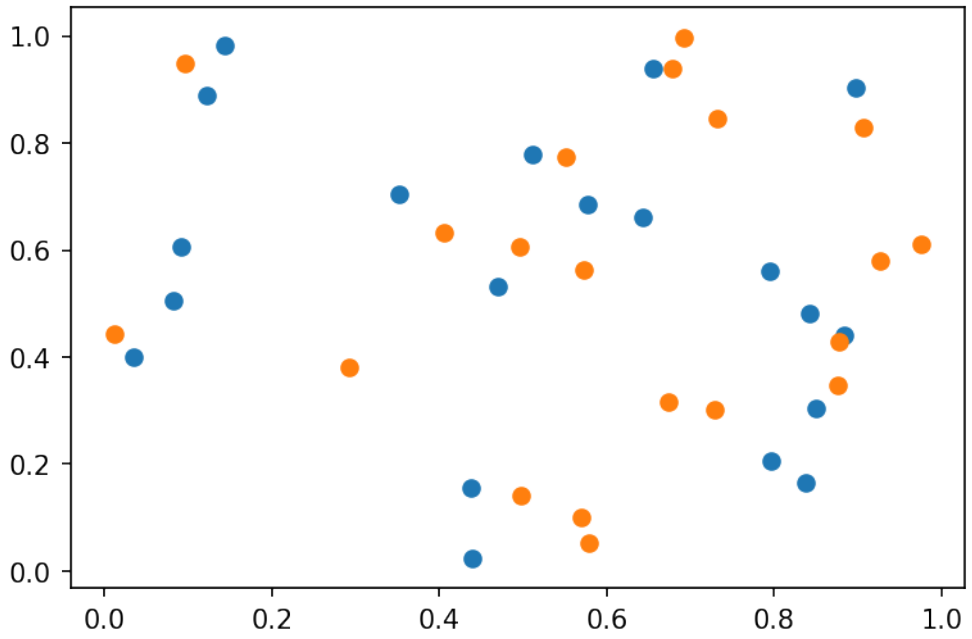
Scatter plots are often used to display two (or more) variables encoded as x and y coordinates, but also as color and size of the markers.

This kind of plot is used to visually inspect the data and find, for instance, relations between the variables.

You can create scatter plots in matplotlib with the *.scatter* method.

[16]:
```
plt.scatter(np.random.rand(1, 20), np.random.rand(1, 20))
plt.scatter(np.random.rand(1, 20), np.random.rand(1, 20))
```

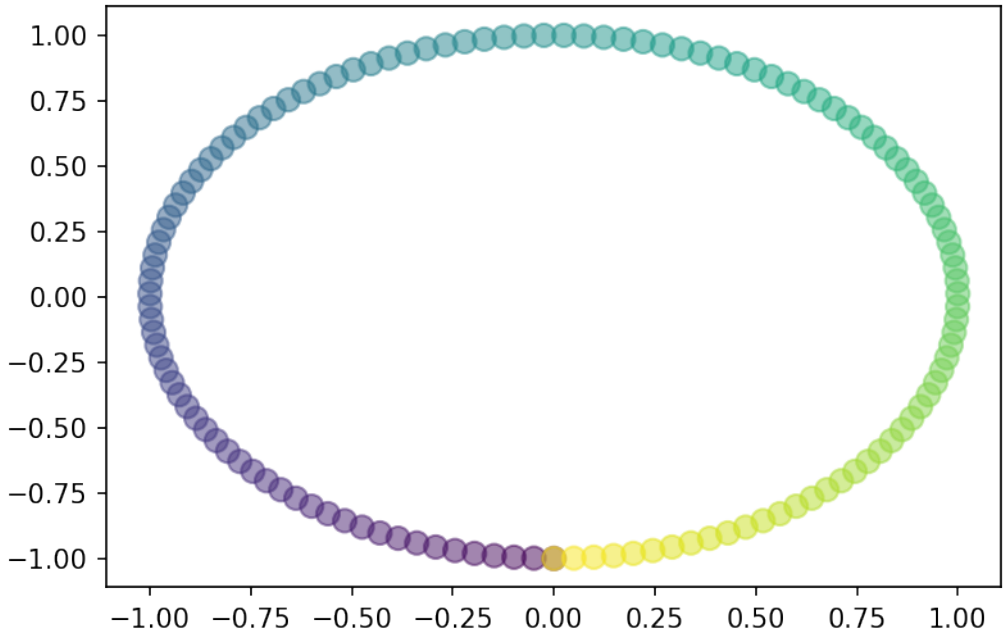
[16]:
```
<matplotlib.collections.PathCollection at 0x7f2c459b8550>
```



This kind of plot is highly customizable.

[17]:
```
plt.scatter(S, C, s=75, c=X, alpha=.5)
```

[17]:
```
<matplotlib.collections.PathCollection at 0x7f2c45f27518>
```



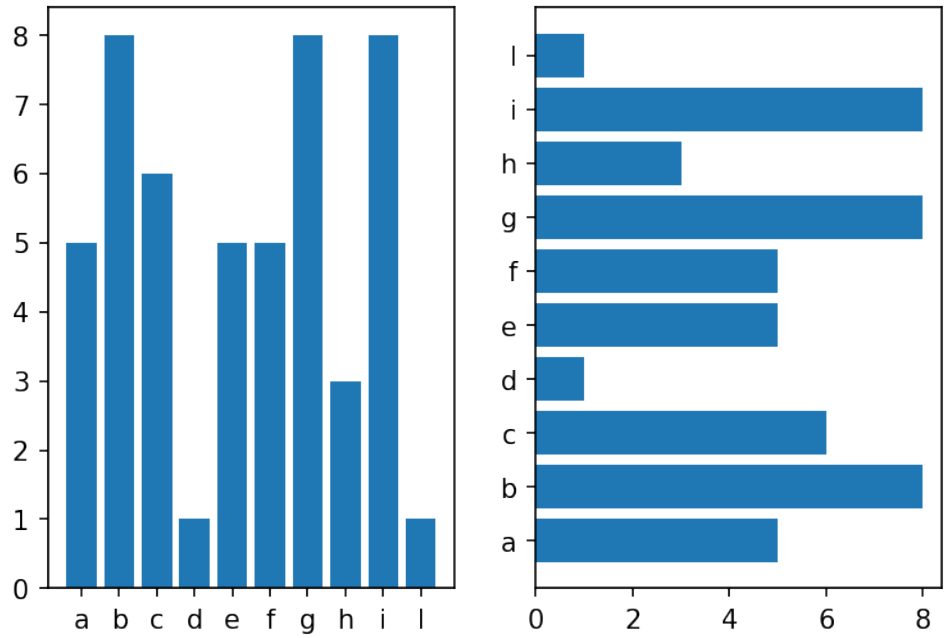
Barplots *Barplots* represent categorical data with rectangular bars.

They can be plotted vertically or horizontally and the height (or length) of each bar depends on the values.

Barplots and *horizontal barplots* can be created with the *.bar* and *.barh* methods respectively.

[18]:
```
teams = list("abcdefghil")
match1 = np.random.randint(1, 10, size=10)

fig, ax = plt.subplots(1, 2)

ax[0].bar(teams, match1)
ax[1].barh(teams, match1)
```

[18]:
```
<BarContainer object of 10 artists>
```



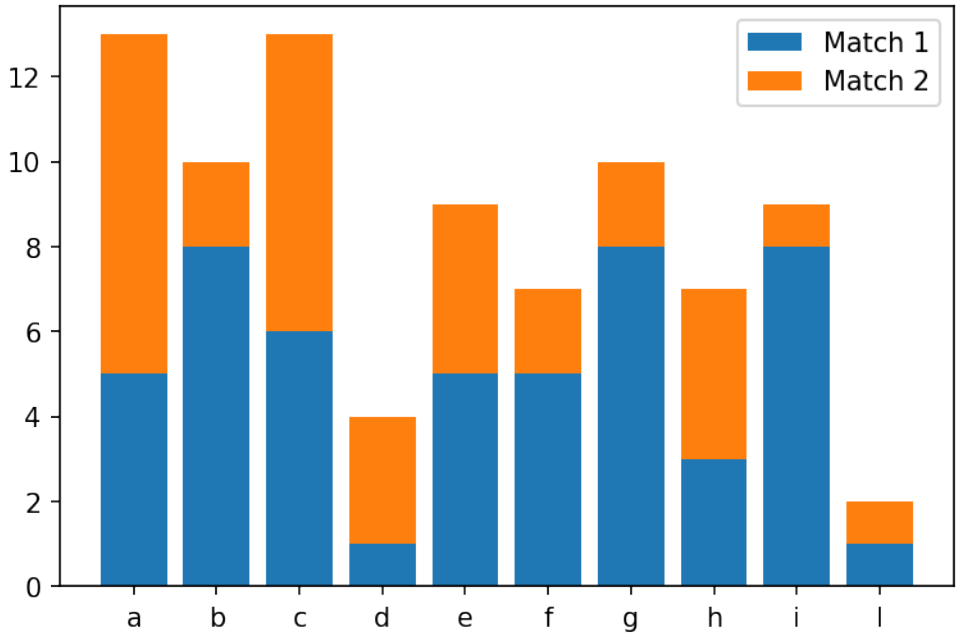
Stacked barplots Stacked barplots are barplots that stack multiple values of the same category together.

The height (or length!) of the resulting bar shows the combined result.

Vertical stacked barplots You just use the *.bar* method and provide the sum of the previous groups as the *bottom* (offset) parameter.

[19]:
```
match2 = np.random.randint(1, 10, size=10)

p1 = plt.bar(teams, match1, label="Match 1")
p2 = plt.bar(teams, match2, label="Match 2", bottom=match1)

plt.legend()
```

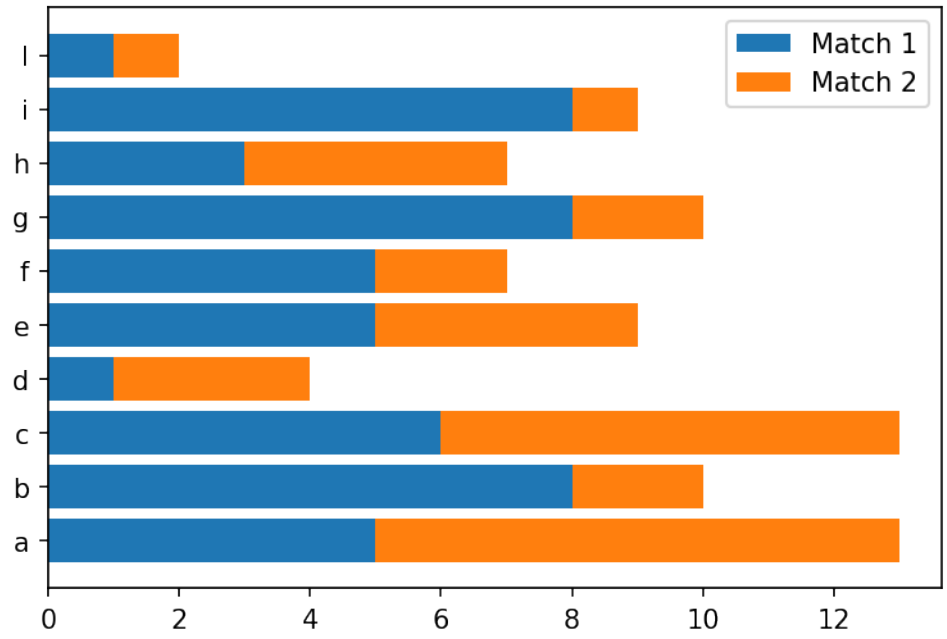
[19]:
```
<matplotlib.legend.Legend at 0x7f2c45ba7c50>
```



Horizontal stacked barplots You just use the *.barh* method and provide the sum of the previous groups as the *left* (offset) parameter.

[20]:
```
p1 = plt.barh(teams, match1, label="Match 1")
p2 = plt.barh(teams, match2, label="Match 2", left=match1)

plt.legend()
```

[20]:
```
<matplotlib.legend.Legend at 0x7f2c440eddd8>
```



1.4 *Seaborn*

Seaborn is a library for making **statistical graphics** in Python.

It has built-in functions to show relationships between variables and to visualize univariate and bivariate distributions and it also provides estimators and linear regression models.

It has advanced support for categorical data.

Seaborn comes nice built-in themes to improve your plots and with better default colors that are studied to improve readability from users. It also has advanced functions to simplify the construction of the plot (e.g., grids, legends).

Seaborn is built on top of *matplotlib* and is closely integrated with *pandas*.

1.4.1 Import convention

Seaborn is conventionally imported as *sns*.

[21]:
```
import seaborn as sns
```

1.4.2 Themes

Seaborn comes with nice themes that affect even your *matplotlib* plots. The default can be set with

.set_theme(context='notebook', style='darkgrid', palette='deep', font='sans-serif', font_scale=1, ...)

The *context* parameter affects the scale elements of the figure and is meant to switch to different contexts (paper, poster, etc) easily.

The *style* parameter affects some aesthetic elements like colors of the axes and of the grid.

The *palette* parameter affects the color palette.

The other parameters are self-explanatory.

You can also set the parameters above individually.

[22]:
```
sns.set_theme()
# sns.reset_orig()
```

[23]:
```
plt.plot(X, C, label="cos")
```

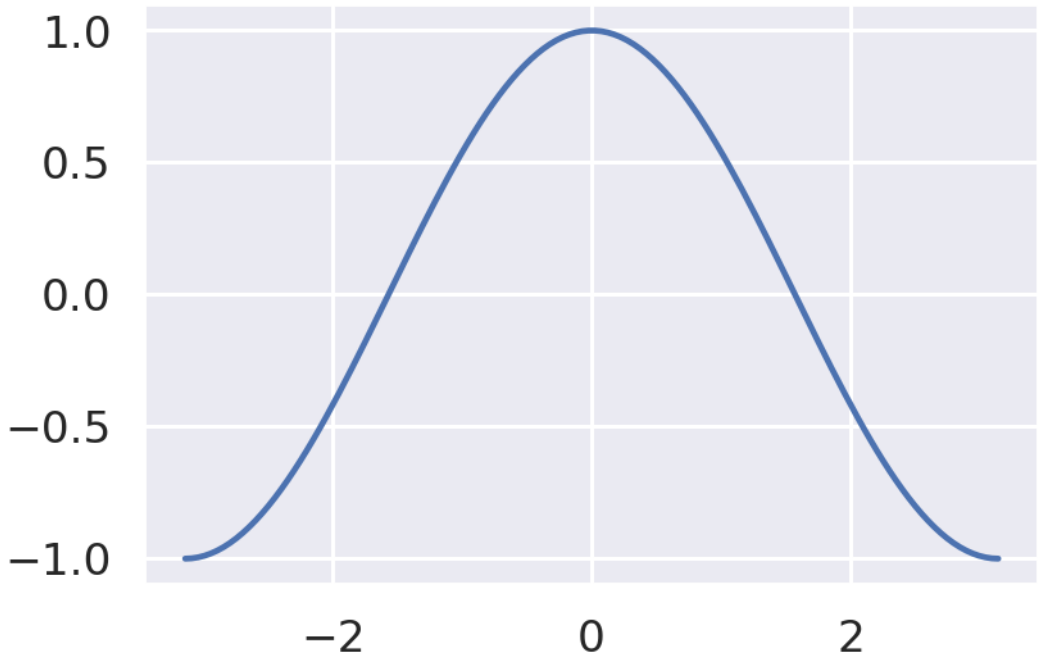
[23]:
```
[<matplotlib.lines.Line2D at 0x7f2c3c411eb8>]
```

Context You can set the style using the *.set_context()* function.

[24]:
```
sns.set_context("poster")
sns.set_context("paper")
sns.set_context("talk")
```

[25]:
```
plt.plot(X, C, label="cos")
```

[25]:
```
[<matplotlib.lines.Line2D at 0x7f2c3c378e48>]
```

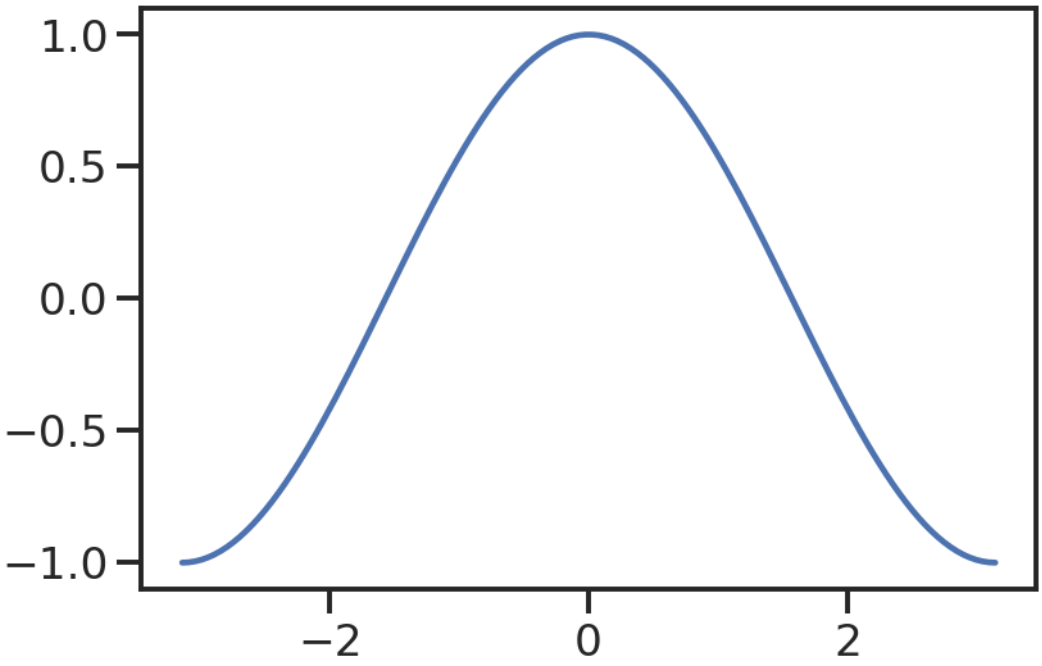


Styles You can set the style using the *.set_style()* function.

[26]:
```
# sns.set_style("white")
# sns.set_style("whitegrid")
sns.set_style("dark")
sns.set_style("darkgrid")
sns.set_style("ticks")
```

[27]:
```
plt.plot(X, C, label="cos")
```

[27]:
```
[<matplotlib.lines.Line2D at 0x7f2c3c347f28>]
```

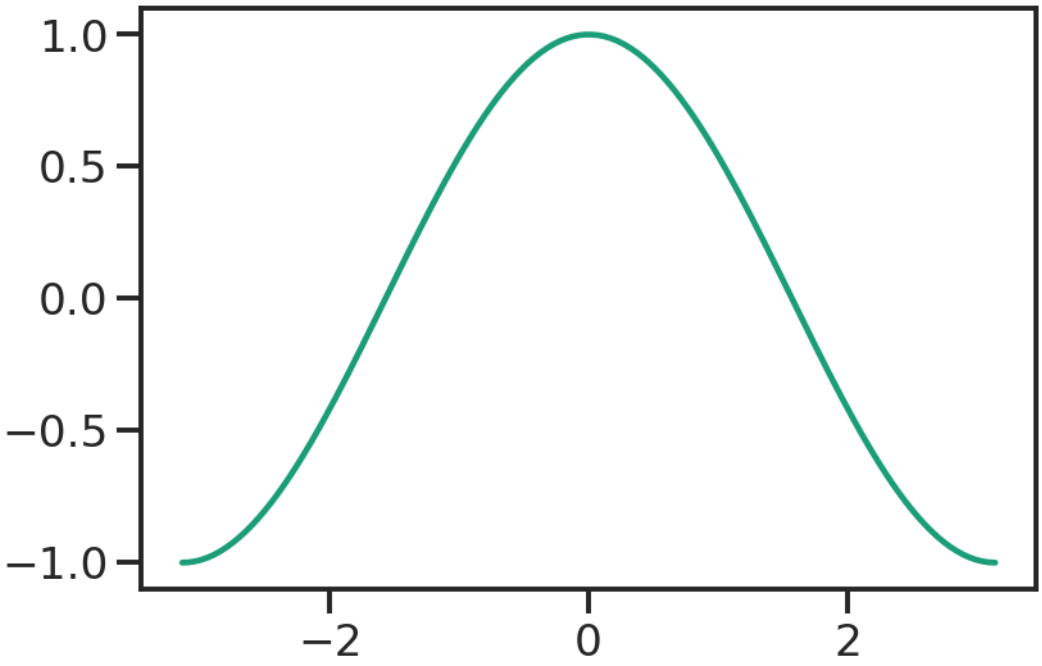


Color palette You can set the style using the *.set_palette()* function (and visualize them with *.color_palette()*.

[28]:
```
# sns.set_palette("flare")
# sns.set_palette("pastel")
# sns.set_palette("dark")
sns.set_palette("Dark2")
```

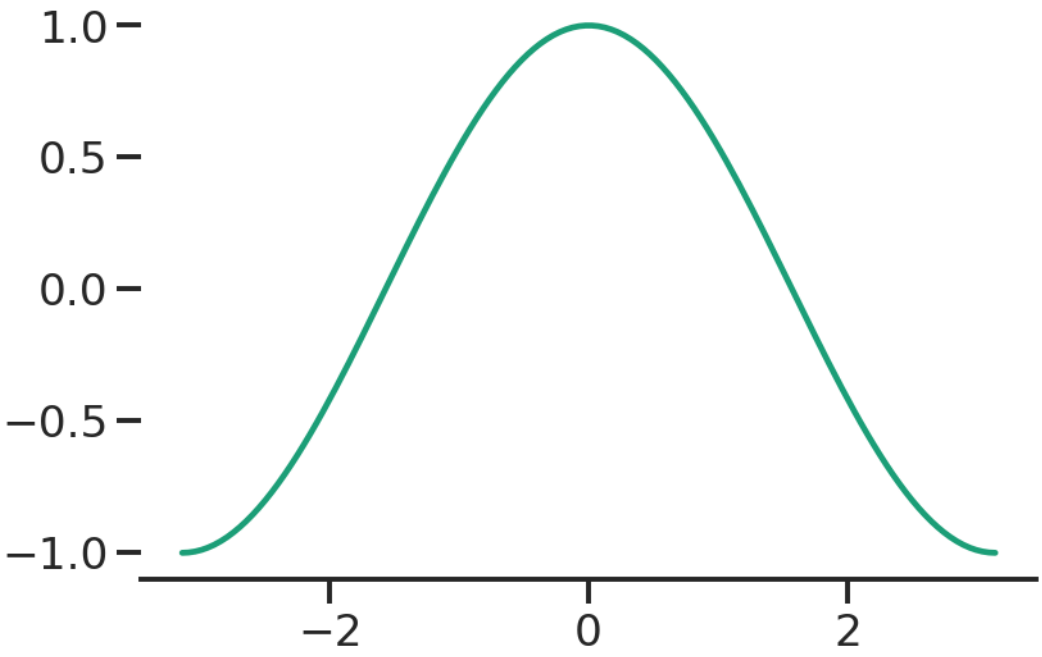
[29]:
```
plt.plot(X, C, label="cos")
```

[29]:
```
[<matplotlib.lines.Line2D at 0x7f2c3c325518>]
```



Removing the spines You can also remove the spines (axis) using *.despine()*.

[30]:
```
plt.plot(X, C, label="cos")

sns.despine(left=True, top=True)
```



[31]:
```
plt.rcParams['figure.figsize'] = [6, 4]
plt.rcParams['figure.dpi'] = 150
```

1.4.3 Structured multi-plot grids: *FacetGrid*

When visualizing data, you may need to plot multiple instances of the same plot on different subsets of your dataset.

For this purpose, *seaborn* provides ***FacetGrid***s, which are basically grids of *Axes*.

FacetGrid can have up to three dimensions: *row*, *col* and *hue*.

We will not discuss how to create *FacetGrid*s manually as many functions automatically create them.

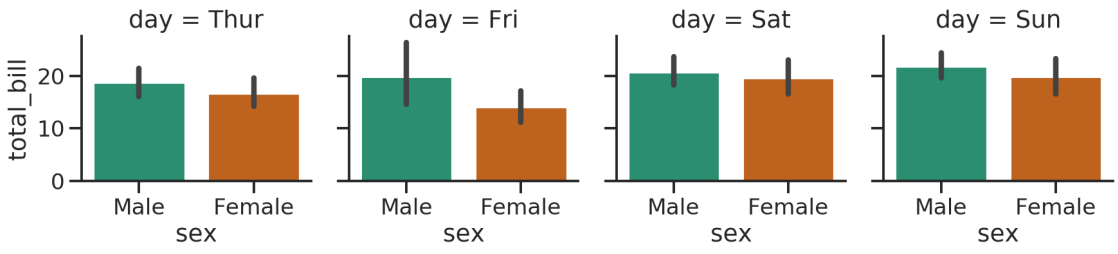
[32]:
```
# Datasets: _seaborn_ has some built in datasets for testing
tips = sns.load_dataset("tips")

tips.head()
```

[32]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

[33]:
```
# Example of manual creation
g = sns.FacetGrid(tips, col="day", hue="sex")
g.map(sns.barplot, "sex", "total_bill", order=["Male", "Female"])
```

[33]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2c3c26b198>
```



1.4.4 Plotting with *seaborn*

Seaborn has a number of very versatile plotting functions.

We will only focus on a few that work as a "wrapper" for the basic ones.

Another nice thing is that it can create multiple *Axes* automagically through *FacetGrid*, depending on the *rows* and *columns* parameters that correspond to the columns of your *DataFrame*.

Relations How can we plot relations with *seaborn*?

```
The relplot() function provides access to several different axes-level functions that show the
```

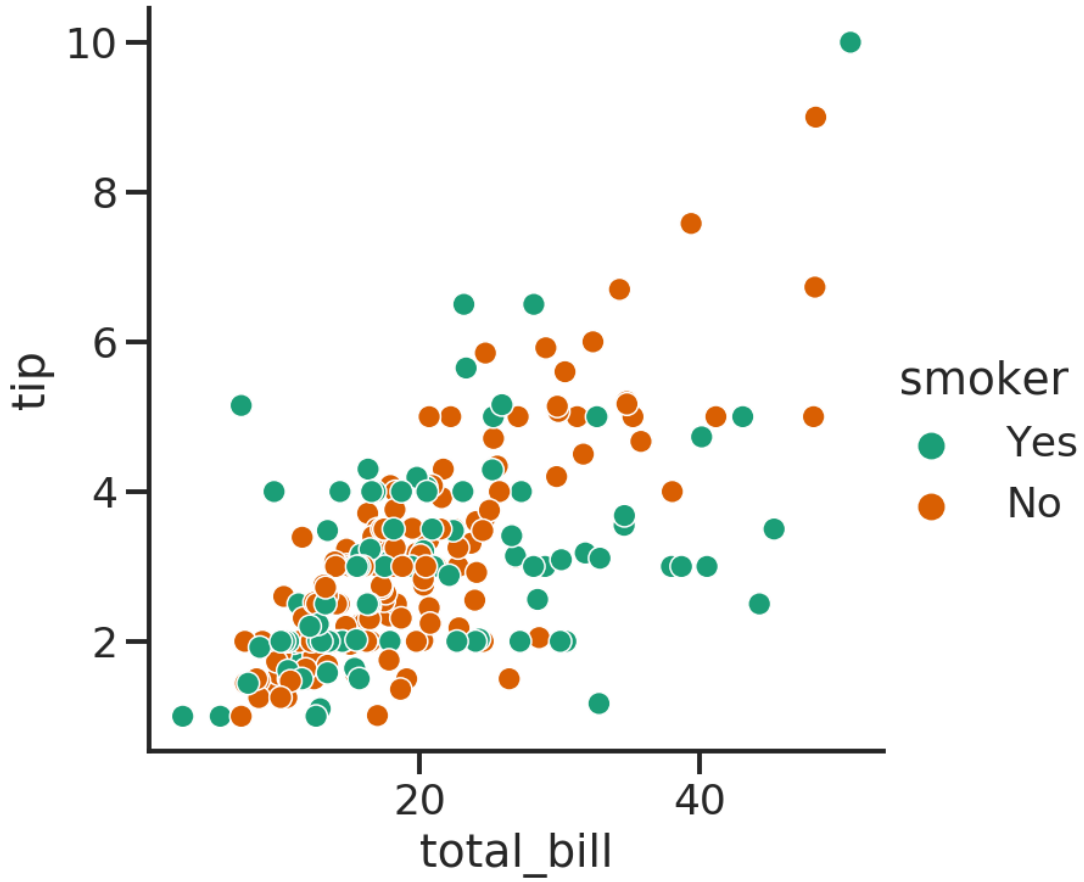
> seaborn.relplot(data=None, x=None, y=None, hue=None, size=None, style=None, row=None, col=None, palette=None, sizes=None, markers=None, dashes=None, legend='auto', kind='scatter', …)

The *kind* parameter selects the underlying axes-level function to use:

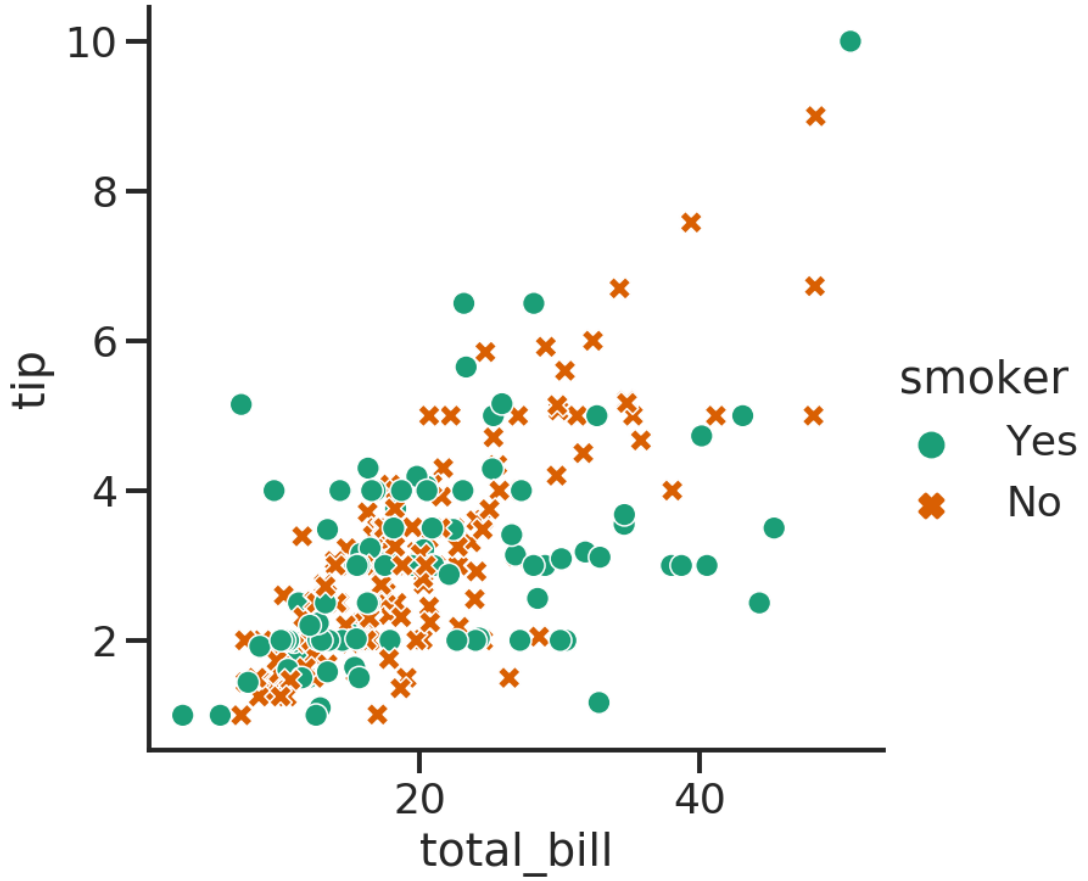
- scatterplot() (with kind="scatter"; the default)
- lineplot() (with kind="line")

[34]:
```
sns.relplot(x="total_bill", y="tip", hue="smoker", data=tips)
# hue_order=["No", "Yes"],
```

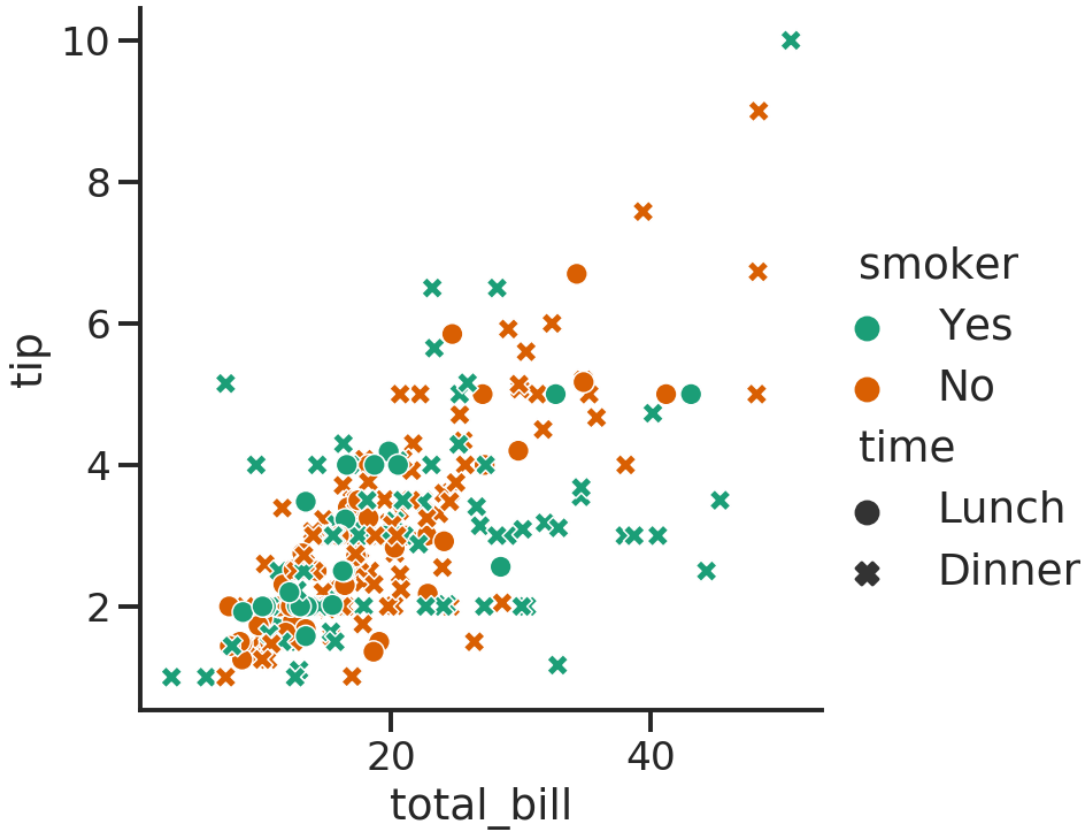
[34]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2c3bd62128>
```



[35]:
```
sns.relplot(x="total_bill", y="tip", hue="smoker", style="smoker", data=tips);
```



[36]:
```
sns.relplot(x="total_bill", y="tip", hue="smoker", style="time", data=tips);
```



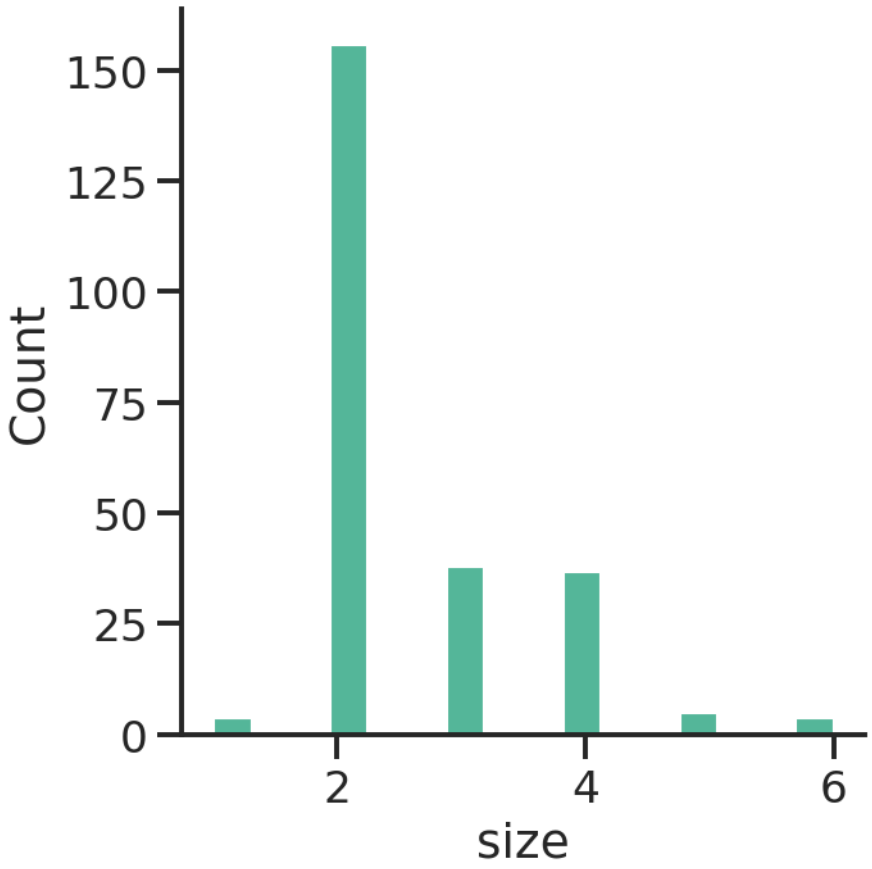
Distributions Distributions can be plotted with *.distplot()*

> *seaborn.displot(data=None, x=None, y=None, hue=None, row=None, col=None, weights=None, kind='hist', rug=False, log_scale=None, legend=True, palette=None, color=None, ...)*

It can plot histograms, kernel density estimates (KDE) or empirical (cumulative) distribution function (ECDF). The KDE and rug plot (showing the individual observations) can also be added to the plot.

[37]:
```
sns.displot(tips, x="size")
```

[37]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2c3ba83c50>
```

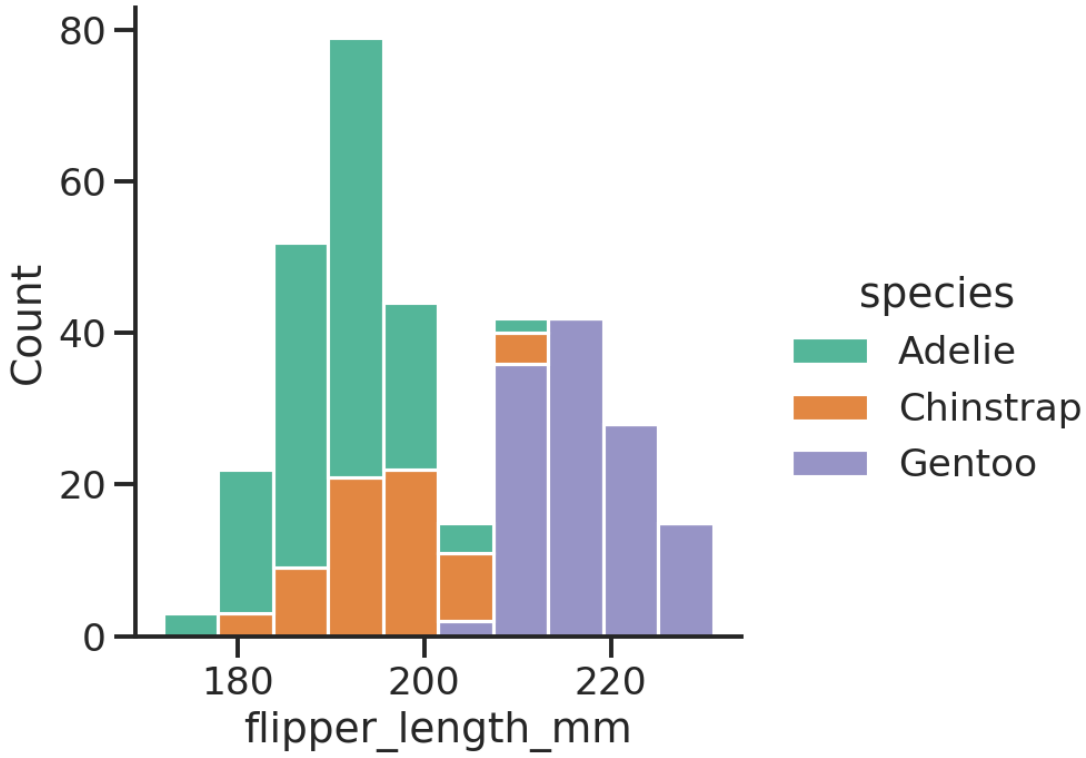


[38]:
```
penguins = sns.load_dataset("penguins")

display(penguins.head())
```

```
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
3  Adelie  Torgersen             NaN            NaN                NaN
4  Adelie  Torgersen            36.7           19.3              193.0

   body_mass_g     sex
0       3750.0    Male
1       3800.0  Female
2       3250.0  Female
3          NaN     NaN
4       3450.0  Female
```

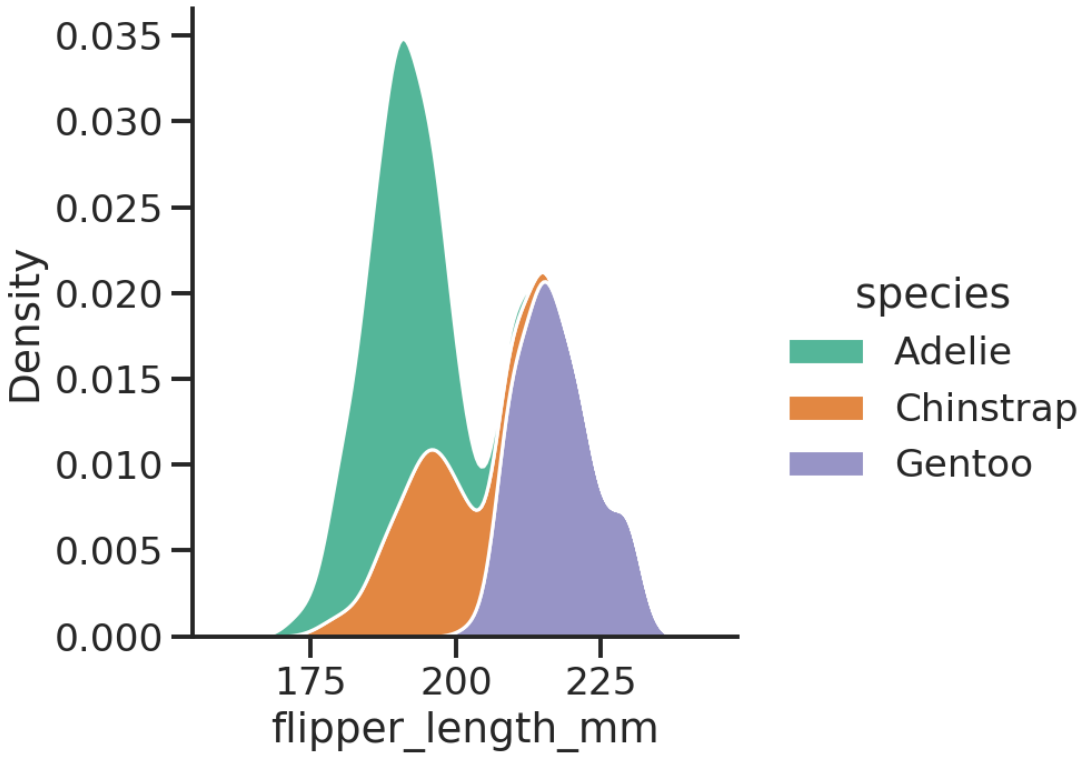
[39]:
```
sns.displot(data=penguins, x="flipper_length_mm", hue="species",␣
↪multiple="stack")
```

[39]: <seaborn.axisgrid.FacetGrid at 0x7f2c3a005630>



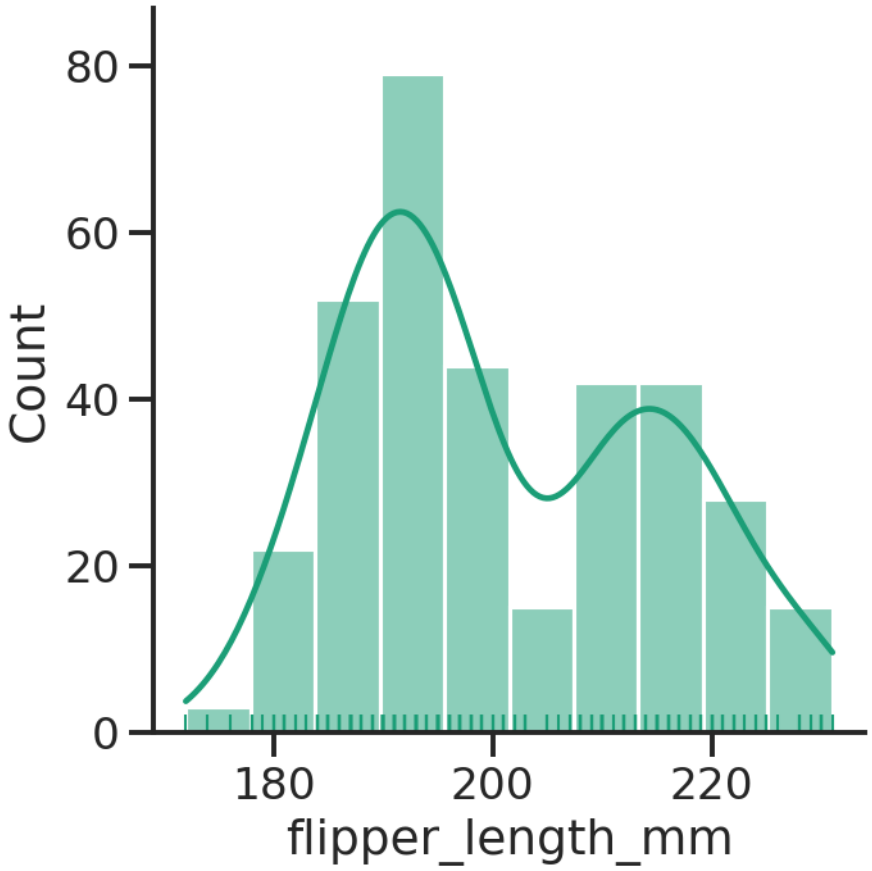
[40]:
```
sns.displot(data=penguins, x="flipper_length_mm", hue="species",␣
↪multiple="stack", kind="kde")
```

[40]: <seaborn.axisgrid.FacetGrid at 0x7f2c3a01cba8>



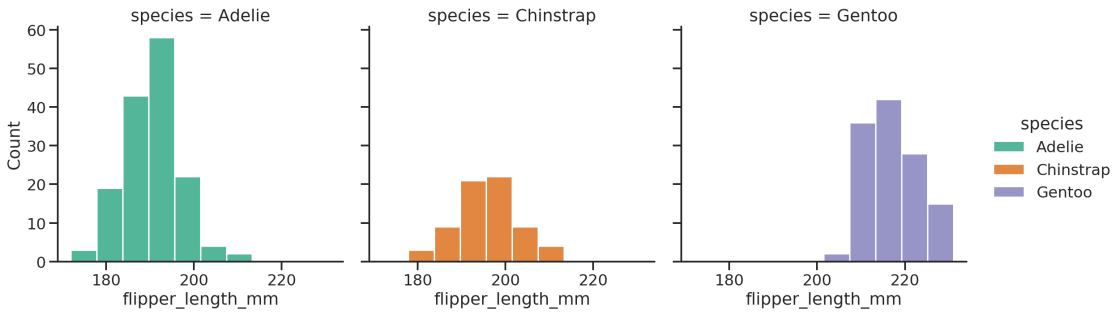
[41]:
```
sns.displot(data=penguins, kind='hist', x="flipper_length_mm", kde=True,␣
↪rug=True)
# sns.displot(data=penguins, kind='kde', x="flipper_length_mm", rug=True)
```

[41]: <seaborn.axisgrid.FacetGrid at 0x7f2c394cf8d0>



[42]:
```
sns.displot(data=penguins, x="flipper_length_mm", hue="species", col="species")
```

[42]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2c38c8dc50>
```



Catplot The *catplot* function provides several functions that show the relationship between a numerical and one or more categorical variables.

seaborn.catplot(data=None, x=None, y=None, hue=None, data=None, row=None,

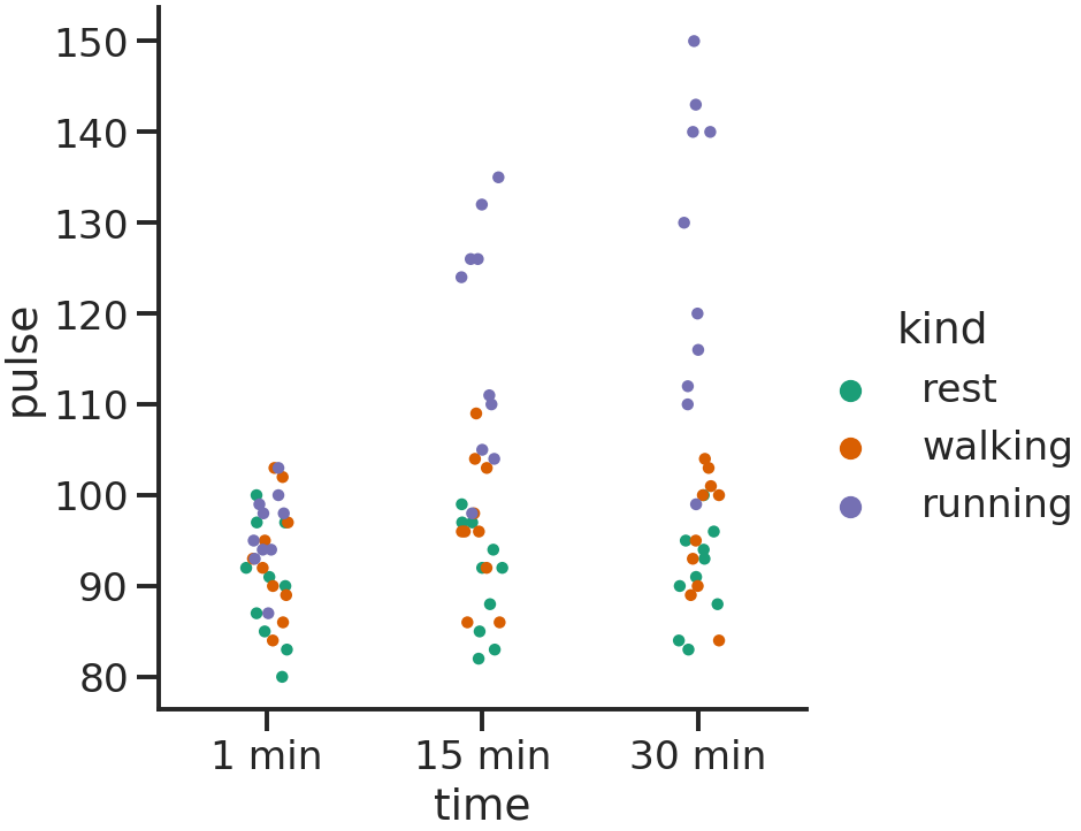
col=None, col_wrap=None, estimator=<function mean at 0x7fa4c4f67940>, ci=95, n_boot=1000, units=None, kind='strip', ...)

The *kind* parameter selects the underlying axes-level function to use. There are categorical:

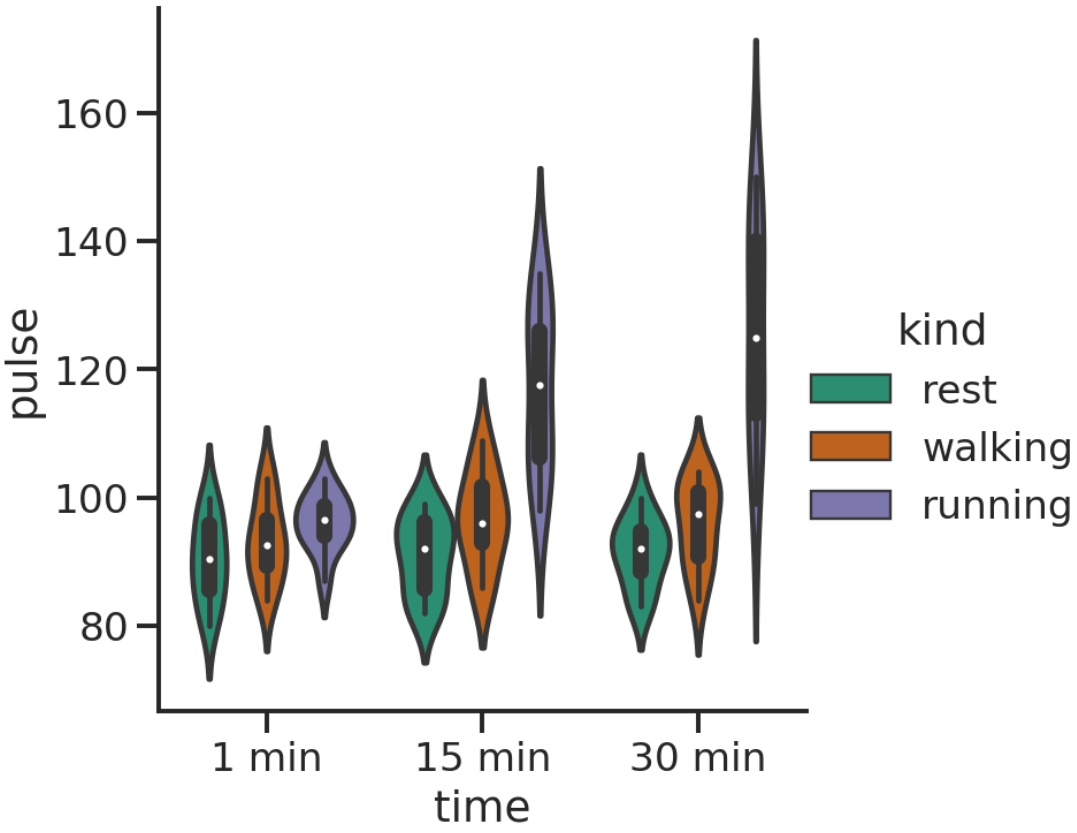
- scatterplots (*stripplot* with *kind*="strip", *swarmplot* with kind="swarm")
- distribution plots (*boxplot* with *kind*="box", *violinplot* with kind="violin", *boxenplot* with kind="boxen")
- estimate plots (*pointplot* with *kind*="point", *barplot* with kind="bar", *countplot* with kind="count")

[43]:
```
exercise = sns.load_dataset("exercise")
```

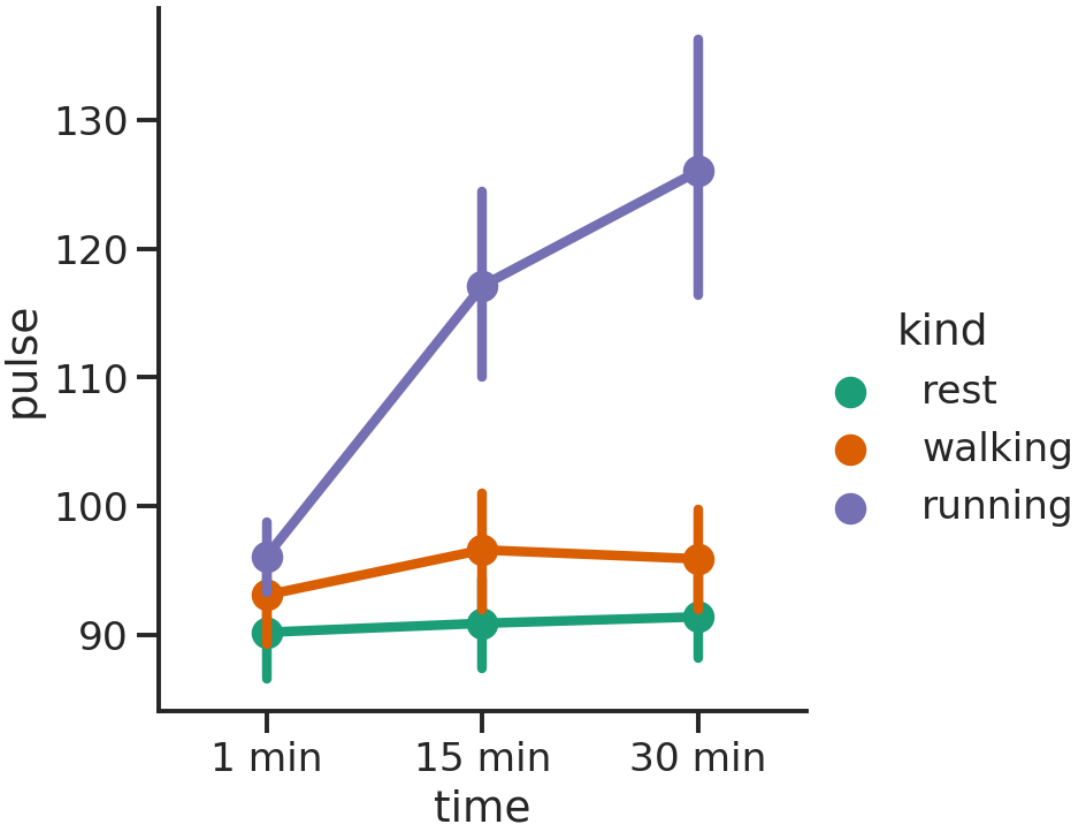
[44]:
```
g = sns.catplot(x="time", y="pulse", hue="kind", data=exercise)
```



[45]:
```
g = sns.catplot(x="time", y="pulse", hue="kind", data=exercise, kind="violin")
```



[46]:
```
g = sns.catplot(x="time", y="pulse", hue="kind", data=exercise, kind="point")
```

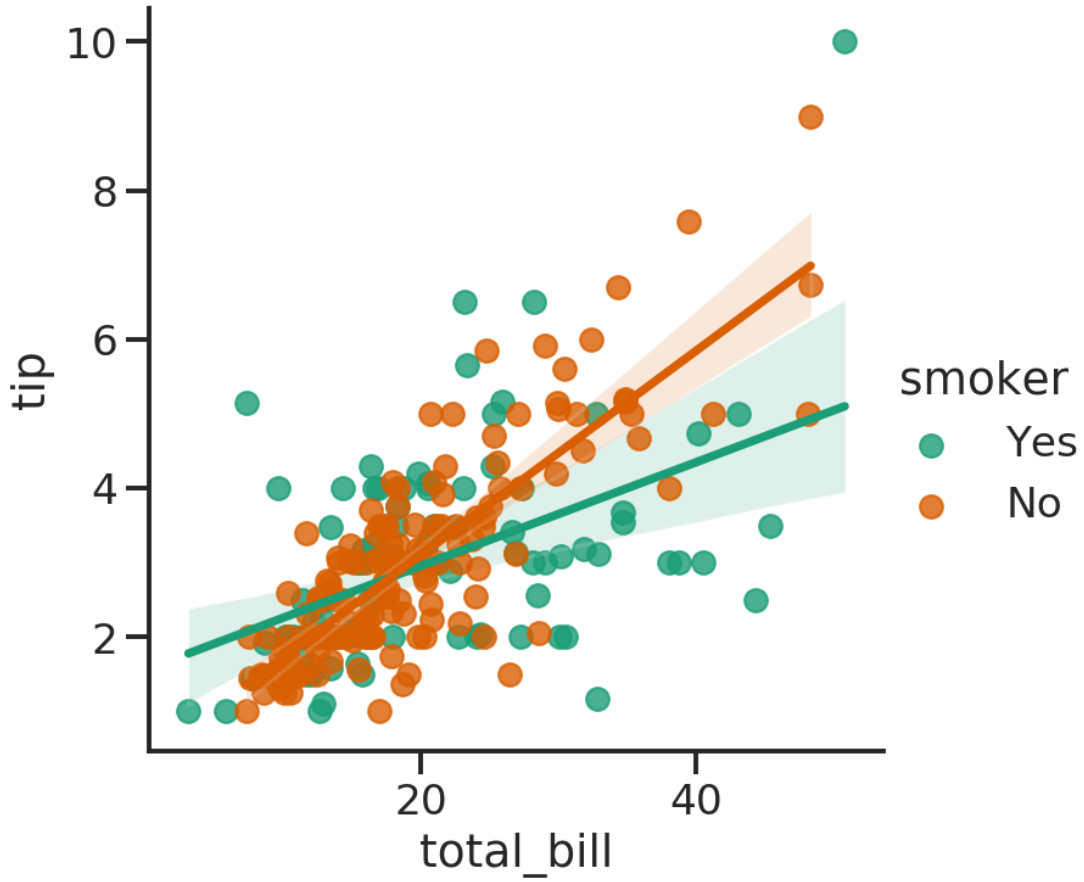


Regressions *lmplot* provides an easy way to fit regression models and plot the.

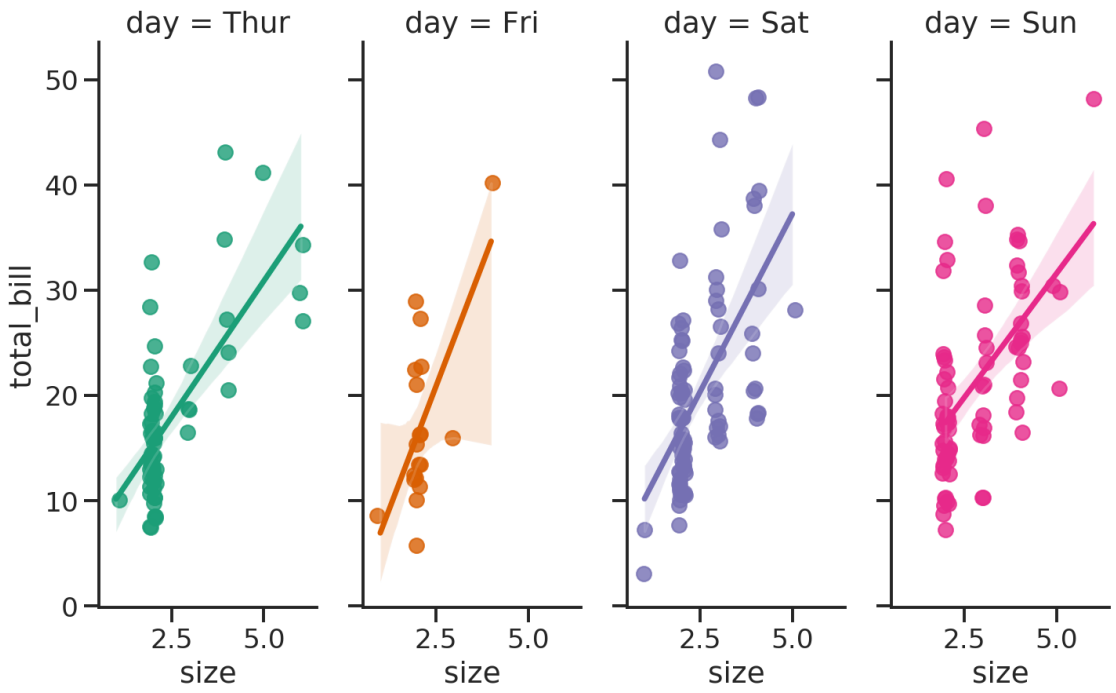
It is intended as a convenient interface to fit regression models across conditional subsets of a dataset.

> seaborn.lmplot(*, x=None, y=None, data=None, hue=None, col=None, row=None, palette=None, col_wrap=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, x_jitter=None, y_jitter=None, scatter_kws=None, line_kws=None, size=None)

[47]:
```
g = sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips)
```



[48]:
```
g = sns.lmplot(x="size", y="total_bill", hue="day", col="day", data=tips,␣
↪height=6, aspect=.4, x_jitter=.1)
```



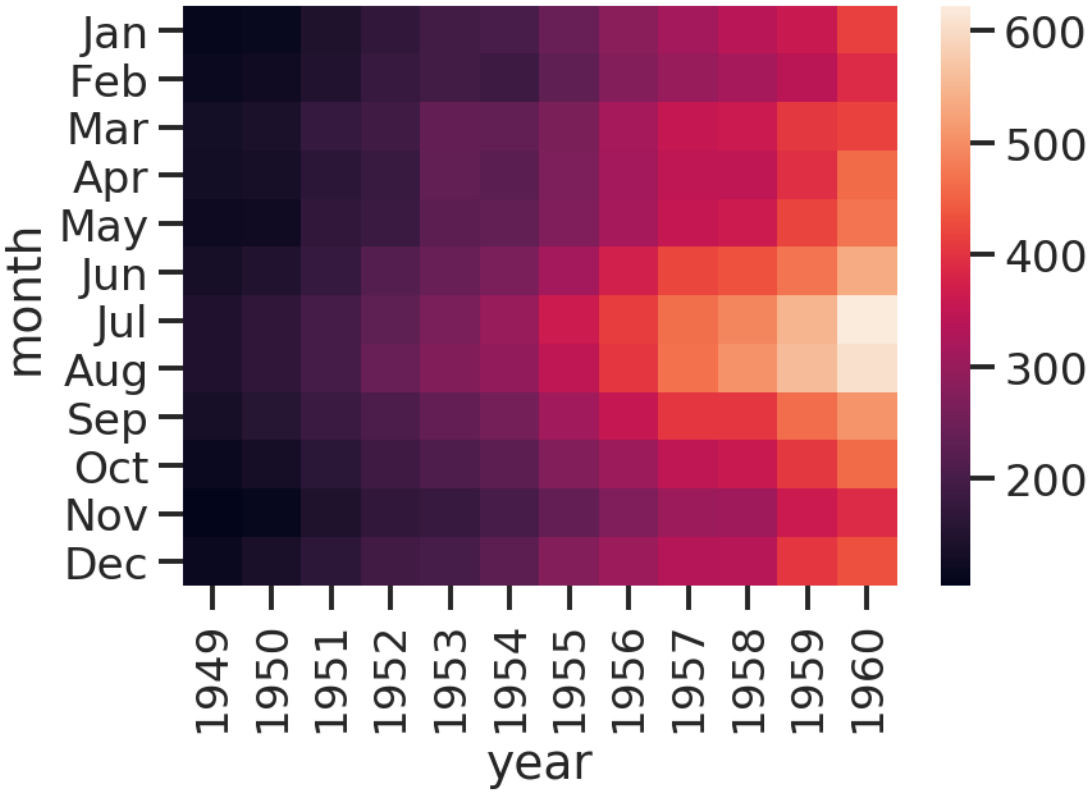
Heatmap and *clustermap*

> *Heatmap* is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. [Wiki]

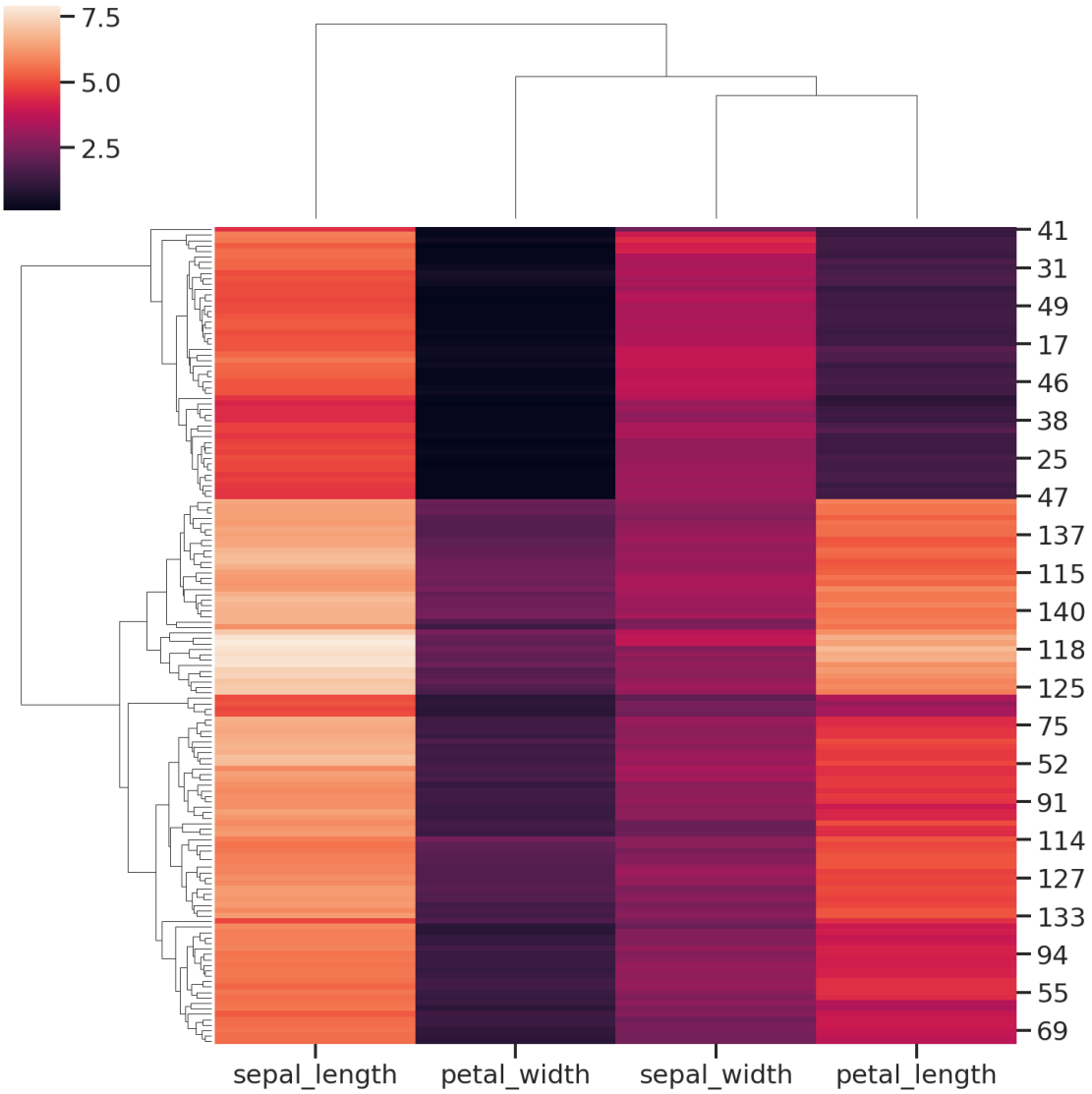
The color may vary in hue or intensity and, if the rows and columns may be reordered to find clusters in the data, the plot is called *clustered heatmap*.

[49]:
```
flights = sns.load_dataset("flights")
flights = flights.pivot("month", "year", "passengers")

sns.heatmap(flights)
```

[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2c3afc1f60>
```



[50]:
```
iris = sns.load_dataset("iris")

species = iris.pop("species")

g = sns.clustermap(iris)
```



1.5 Exercise

Given the CSV file from the previous lecture, load it and perform some statistical analysis.

Load the CSV file.

[51]:
```
import pandas as pd

file = "albumlist.csv"

df = pd.read_csv(file,
                 encoding="ISO-8859-15",
                 index_col="Number"
                )
```

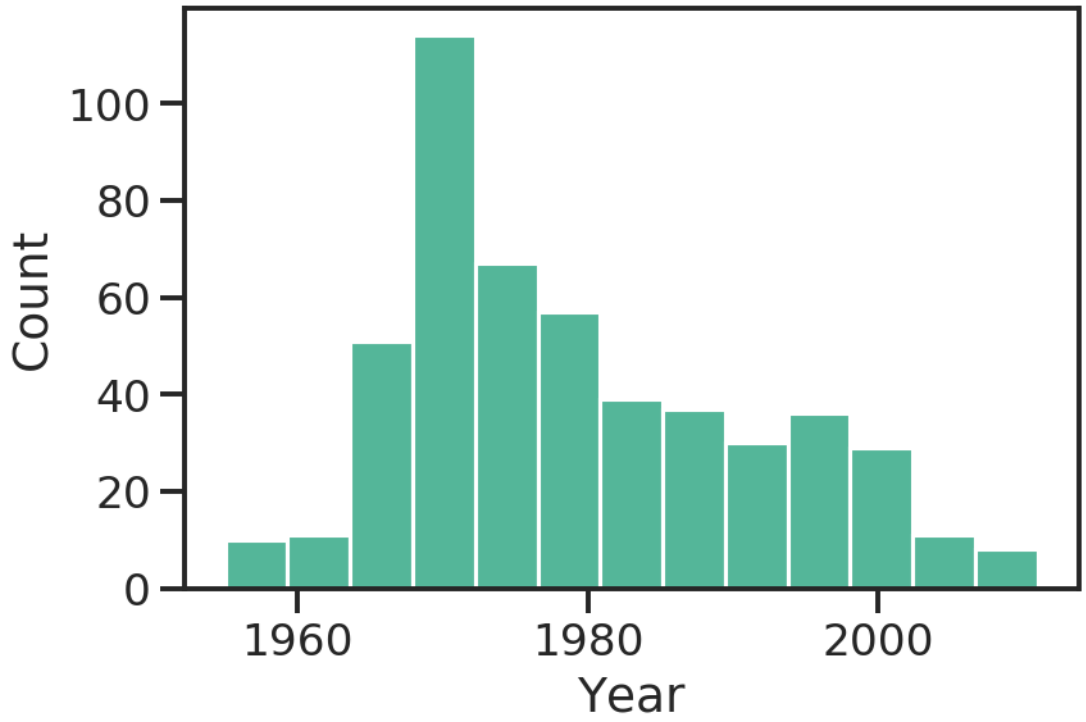
```
display(df.head())
```

```
        Year                                    Album          Artist  \
Number
1       1967  Sgt. Pepper's Lonely Hearts Club Band     The Beatles
2       1966                              Pet Sounds  The Beach Boys
3       1966                                Revolver     The Beatles
4       1965                  Highway 61 Revisited       Bob Dylan
5       1965                             Rubber Soul     The Beatles

            Genre                       Subgenre
Number
1            Rock  Rock & Roll, Psychedelic Rock
2            Rock     Pop Rock, Psychedelic Rock
3            Rock     Psychedelic Rock, Pop Rock
4            Rock          Folk Rock, Blues Rock
5       Rock, Pop                       Pop Rock
```

Plot the histogram of the number of albums in the cart for each year.

[52]:
```
sns.histplot(data=df, x="Year")
```

[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2c3ad55780>
```



Show the unique values of the *Genre* column.

[53]:
```
df["Genre"].unique()
```

[53]:
```
array(['Rock', 'Rock, Pop', 'Funk / Soul', 'Rock, Blues', 'Jazz',
       'Jazz, Rock, Blues, Folk, World, & Country', 'Funk / Soul, Pop',
       'Blues', 'Pop', 'Rock, Folk, World, & Country',
       'Folk, World, & Country', 'Classical, Stage & Screen', 'Reggae',
       'Hip Hop', 'Jazz, Funk / Soul', 'Rock, Funk / Soul, Pop',
       'Electronic, Rock',
       'Jazz, Rock, Funk / Soul, Folk, World, & Country',
       'Jazz, Rock, Funk / Soul, Pop, Folk, World, & Country',
       'Funk / Soul, Stage & Screen',
       'Electronic, Rock, Funk / Soul, Stage & Screen',
       'Rock, Funk / Soul', 'Rock, Reggae', 'Jazz, Pop',
       'Funk / Soul, Folk, World, & Country', 'Latin, Funk / Soul',
       'Funk / Soul, Blues',
       'Reggae,ÊPop,ÊFolk, World, & Country,ÊStage & Screen',
       'Electronic,ÊStage & Screen', 'Jazz, Rock, Funk / Soul, Blues',
       'Jazz, Rock', 'Rock, Latin, Funk / Soul', 'Electronic, Rock, Pop',
       'Hip Hop, Rock, Funk / Soul', 'Electronic, Pop',
       'Rock, Blues, Pop', 'Electronic, Rock, Funk / Soul, Pop',
       'Rock, Funk / Soul, Folk, World, & Country', 'Rock,ÊBlues',
       'Rock, Pop, Folk, World, & Country', 'Rock, Latin',
       'Rock, Stage & Screen', 'Rock, Blues, Folk, World, & Country',
       'Electronic', 'Electronic, Funk / Soul, Pop',
       'Pop, Folk, World, & Country', 'Electronic, Hip Hop, Pop',
       'Blues, Folk, World, & Country',
       'Electronic, Hip Hop, Funk / Soul, Pop',
       'Rock, Funk / Soul, Blues, Pop, Folk, World, & Country',
       'Jazz, Pop, Folk, World, & Country', 'Jazz, Rock, Pop',
       'Hip Hop, Funk / Soul', 'Hip Hop, Rock',
       'Electronic, Hip Hop, Funk / Soul',
       'Funk / Soul,ÊFolk, World, & Country',
       'Electronic, Hip Hop, Reggae, Pop', 'Electronic, Reggae',
       'Electronic, Funk / Soul', 'Rock, Funk / Soul, Blues', 'Rock,ÊPop',
       'Electronic, Rock, Funk / Soul, Blues, Pop', 'Rock, Reggae, Latin'],
      dtype=object)
```

Since there are a bit too many sub-genres for each row, keep just the first one.

[54]:
```
df["MainGenre"] = df["Genre"].apply(lambda x: x.strip().split(",")[0].strip())
```

[55]:
```
df["MainGenre"].unique()
```

[55]:
```
array(['Rock', 'Funk / Soul', 'Jazz', 'Blues', 'Pop', 'Folk', 'Classical',
       'Reggae', 'Hip Hop', 'Electronic', 'Latin'], dtype=object)
```

Set the *Genre* as categorical variable.

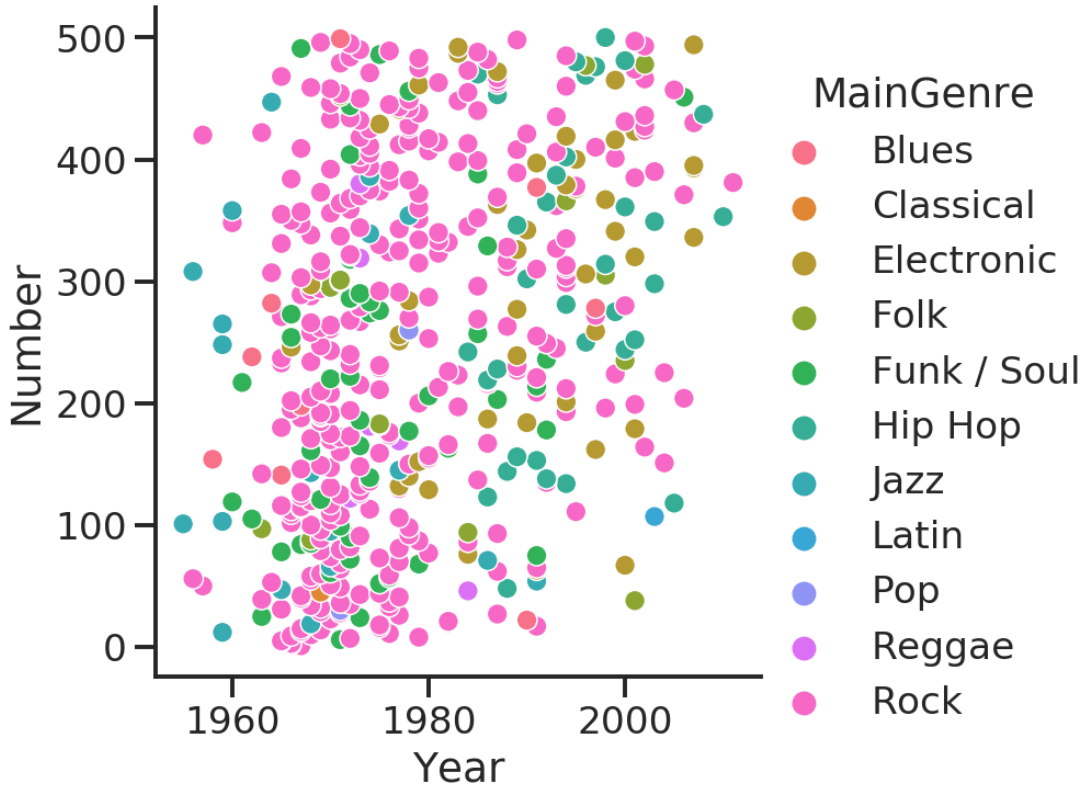
[56]:
```
df["Genre"] = df["Genre"].astype("category")
df["MainGenre"] = df["MainGenre"].astype("category")

display(df["MainGenre"].unique())
```

```
[Rock, Funk / Soul, Jazz, Blues, Pop, …, Classical, Reggae, Hip Hop, Electronic,␣
↪Latin]
Length: 11
Categories (11, object): [Rock, Funk / Soul, Jazz, Blues, …, Reggae, Hip Hop,␣
↪Electronic, Latin]
```

Now let's plot a *scatterplot* of the position (*Number*, used as index) as function of the *Year*. As a third variable, we are also interested in the main genre of the album.

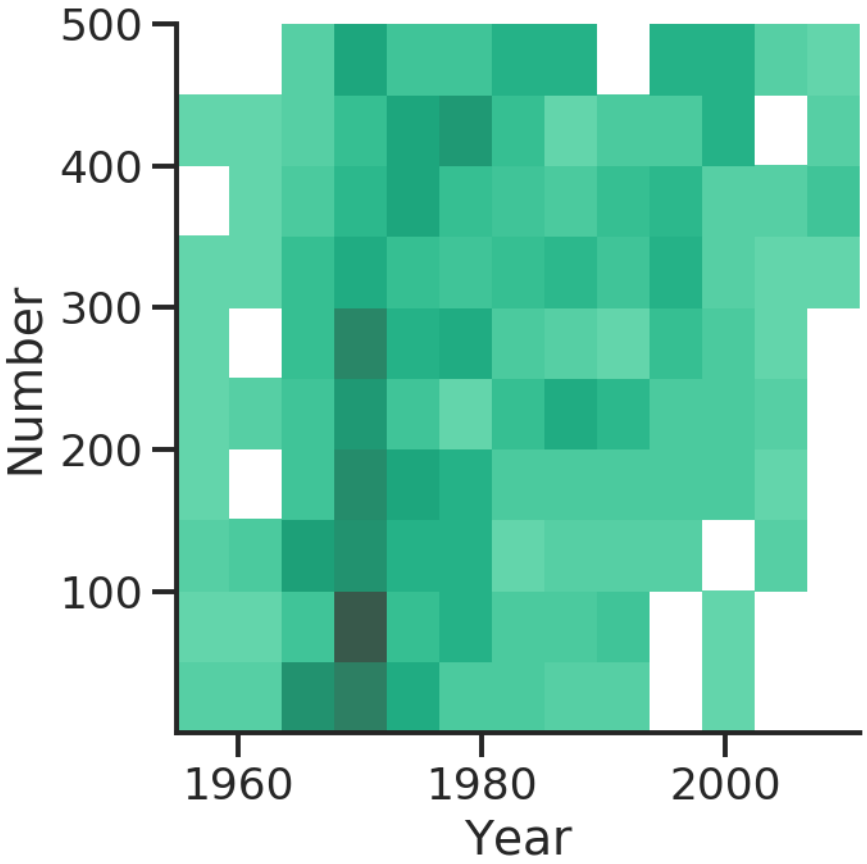
[57]:
```
sns.relplot(kind="scatter", data=df, x="Year", y=df["Year"].index,␣
↪hue="MainGenre")
```

[57]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2c3acadcc0>
```



[58]:
```
sns.displot(df, x="Year", y="Number", kind="hist")
```

[58]: <seaborn.axisgrid.FacetGrid at 0x7f2c3ac300b8>



Reset the index to restore the *Number* column.

[59]:
```
df.reset_index(drop=False, inplace=True)
```

And compute the correlation between the *Number* and *Year* columns.

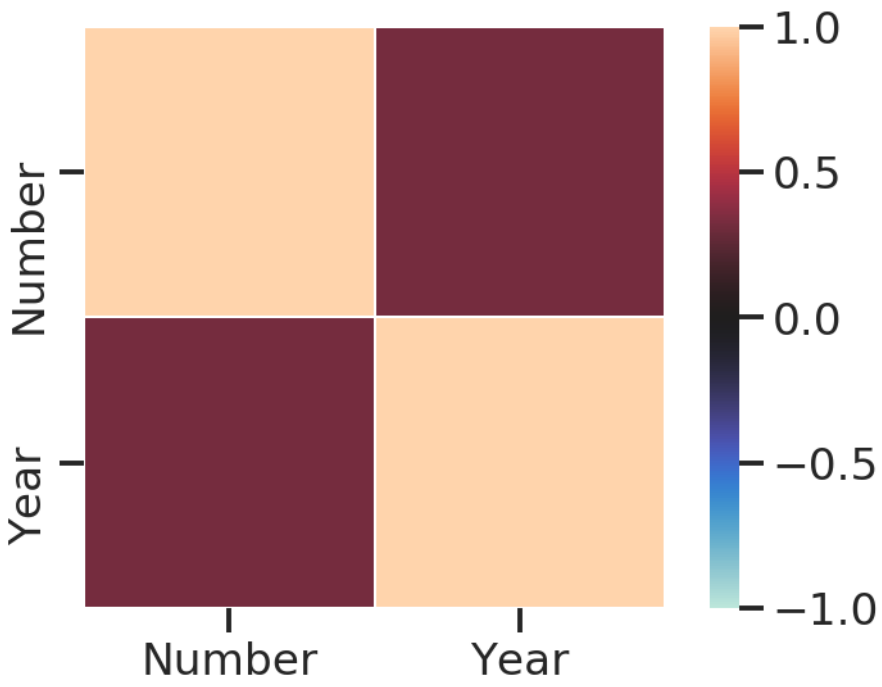
[60]:
```
corr = df[["Number", "Year"]].corr()

display(corr)
```

```
          Number      Year
Number  1.000000  0.325667
Year    0.325667  1.000000
```

What about showing the correlation via a plot?

[61]:
```
sns.heatmap(corr, center=0, vmin=-1, vmax=1, square=True, linewidths=.5)
```

[61]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c3abcd048>



The heatmap above shows the correlation between the two variables.

Yet, we don't like the colors and we know the matrix is symmetrical, so we would like to show only the lower triangle.

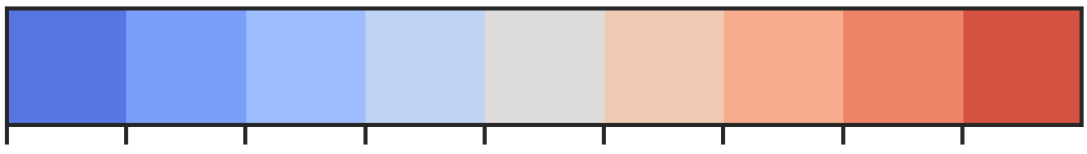
Let's try to improve the plot.

What about the color palette?

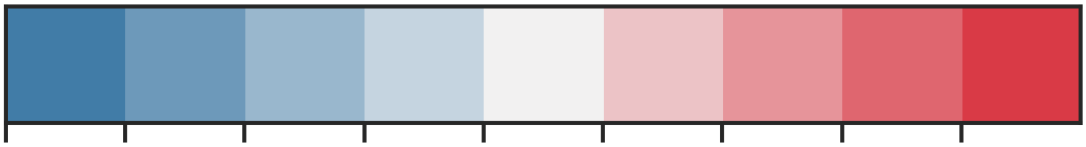
While *matplotlib* has many palettes, not all of them are actually good for visualization.

Seaborn tries to overcome this limitation by providing a nice interface for the *HSLuv* (formerly known as *HUSL*) color system, which works better with human vision as it minimizes the variation of intensity of colors.

[62]:
```
# Built in in matplotlib
sns.palplot(sns.color_palette("coolwarm", n_colors=9))
```



[63]:
```
# HUSL system palette
sns.palplot(sns.diverging_palette(240, 10, n=9))
```



[64]:
```
# Generate a custom diverging colormap
cmap = sns.diverging_palette(240, 10, as_cmap=True)
# cmap = sns.color_palette("coolwarm", as_cmap=True)
```

Let's generate a mask to filter out some values from the plot.

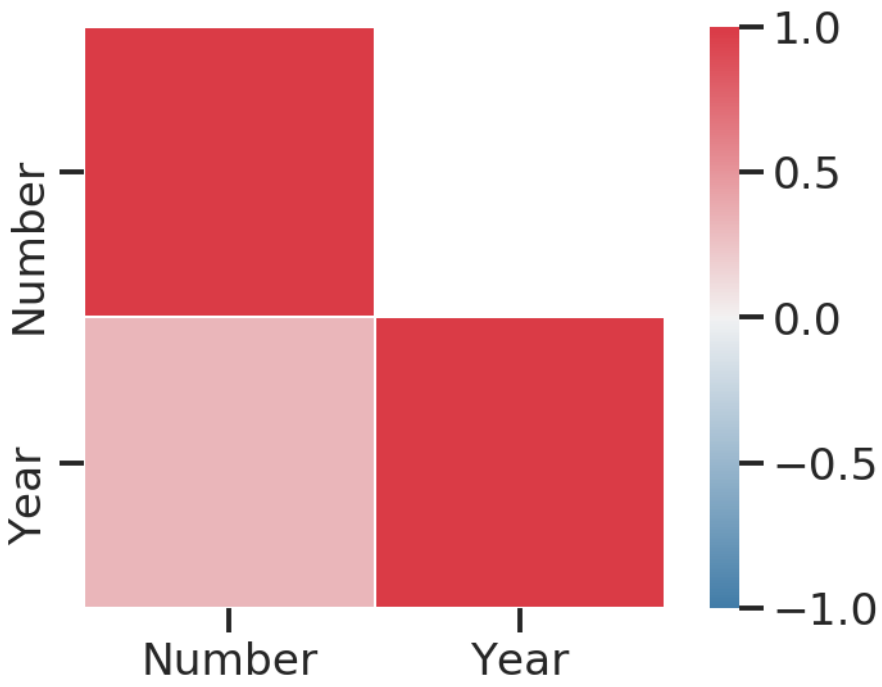
We are interested in displaying just the lower (or the upper) triangle of the matrix.

[65]:
```
# Generate a mask for the upper triangle, excluding the diagonal
mask = np.triu(
    np.ones_like(corr, dtype=bool),
    k=1
)
display(mask)
```

```
array([[False,  True],
       [False, False]])
```

[66]:
```
sns.heatmap(corr, cmap=cmap, mask=mask, center=0, vmin=-1, vmax=1, square=True,␣
↪linewidths=.5)
```

[66]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2c3b218048>
```



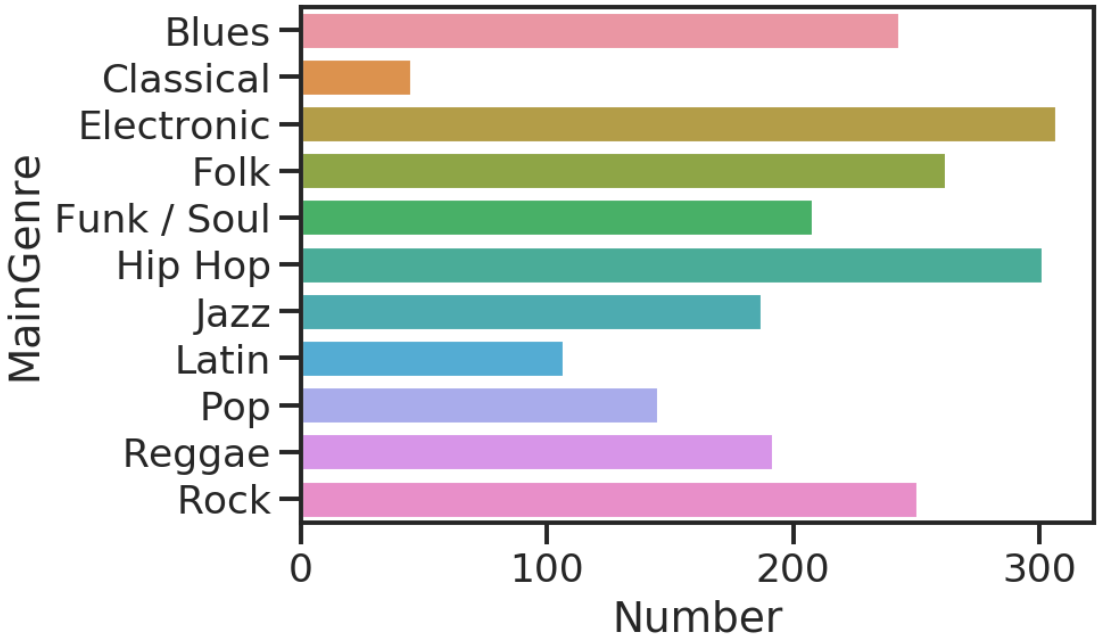
At this point we want to group the *DataFrame* by the main genre to compute the average position in the chart.

[67]:
```
grouped_by_genre = df.groupby("MainGenre")
mean_position = grouped_by_genre["Number"].mean()
mean_position = mean_position.round(2)

display(mean_position)
```

```
MainGenre
Blues          243.22
Classical       45.00
Electronic     307.13
Folk           262.23
Funk / Soul    208.08
Hip Hop        301.41
Jazz           187.42
Latin          107.00
Pop            145.00
Reggae         191.86
Rock           250.39
Name: Number, dtype: float64
```

[68]:
```
sns.barplot(x=mean_position, y=mean_position.index)
```

[68]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c440c9e10>



1.6 References

- *IPython*
- Matplotlib Usage Guide
- SciPy Lectures: Plotting
- Seaborn tutorial
- Python Graph Gallery
- Matplotlib: gallery
- Seaborn: gallery