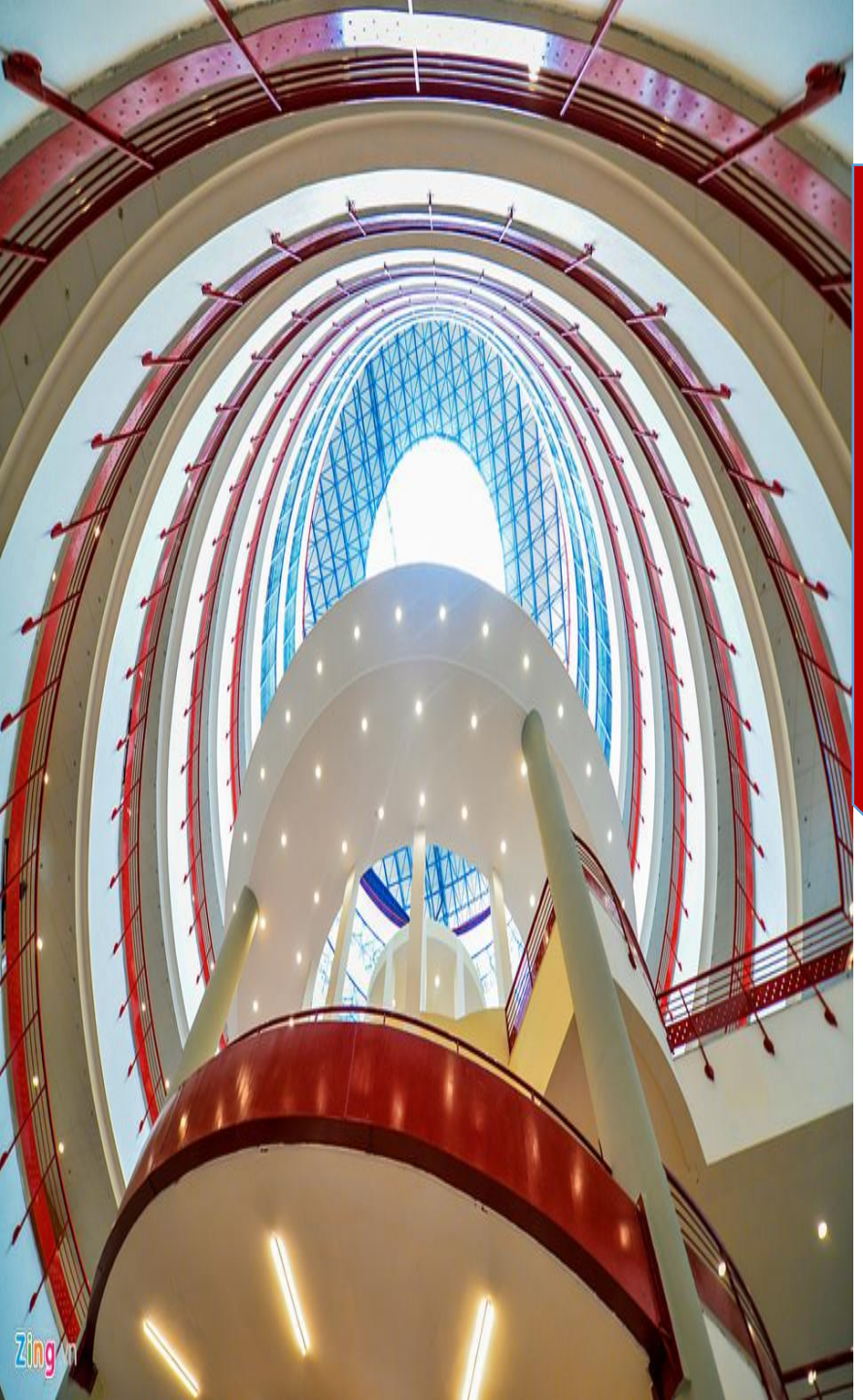
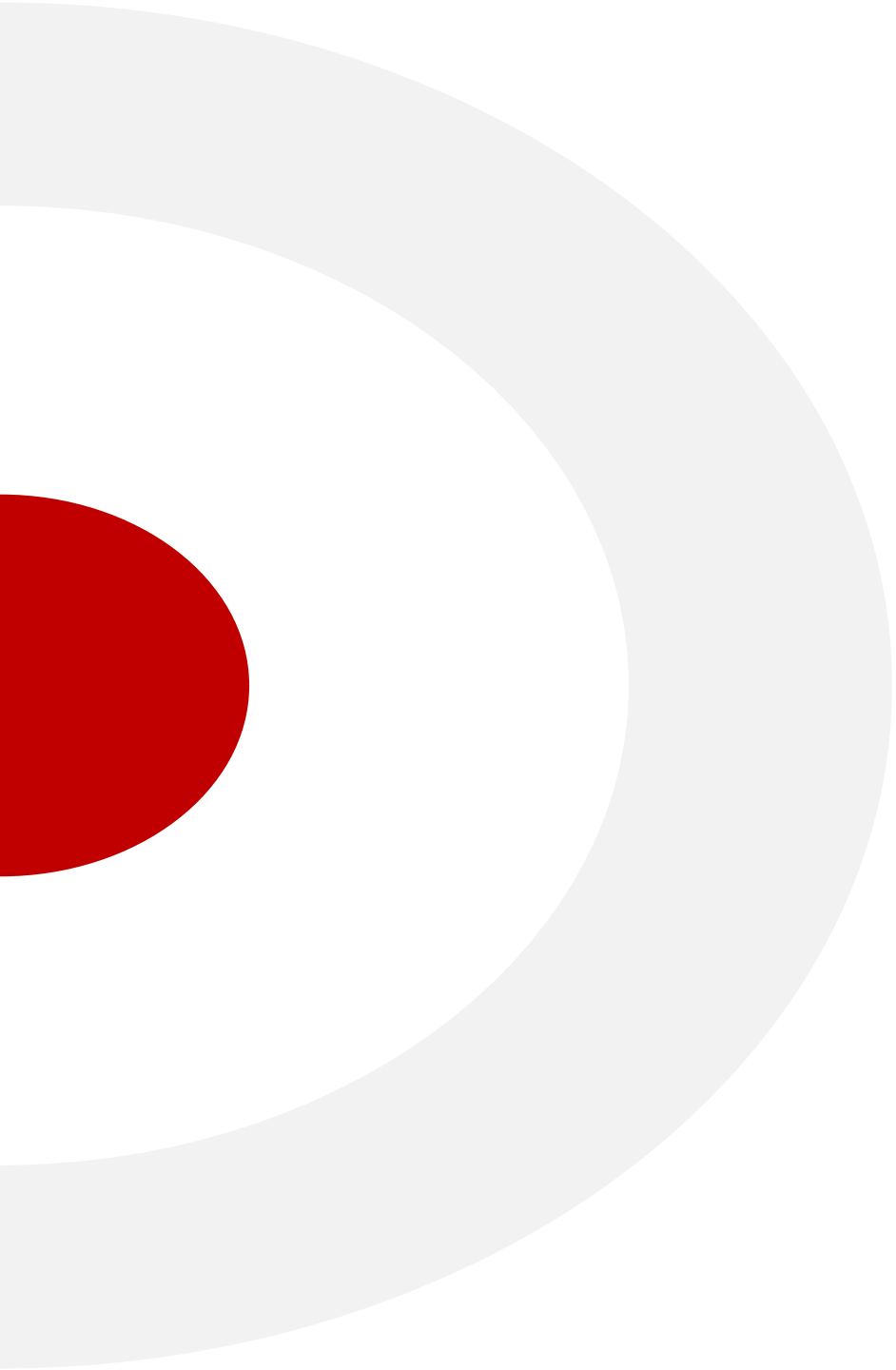


DATA PREPARATION AND VISUALIZATION

Mathematical Economics Faculty

National Economics University
https://www.neu.edu.vn/



Chapter 11: Plotting and Visualization

Outline

1. **Introduction To Matplotlib**
   - line
   - scatter
   - bar
   - hist
   - Subplots
2. **Introduction To Seaborn**
   - Distribution: Hist, KDE
   - Join Plot
   - Pair Plot
   - Bar and Box Plot Facet Plot

Introduction

- At the heart of any data science workflow is data exploration. Most commonly, we explore data by using the following:
  - Statistical methods(measuring averages, measuring variability,…)
  - Data visualization (transforming data into a visual form)
- The other central task is to help us communicate and explain the results we've found through exploring data. That being said, we have two kinds of data visualization:
  - Exploratory data visualization: we build graphs for ourselves to explore data and find patterns
  - Explanatory data visualization: we build graphs for others to communicate and explain the patterns we've found through exploring data

Introduction

Exploratory
data visualization

We build graphs for ourselves
to explore data and find patterns.

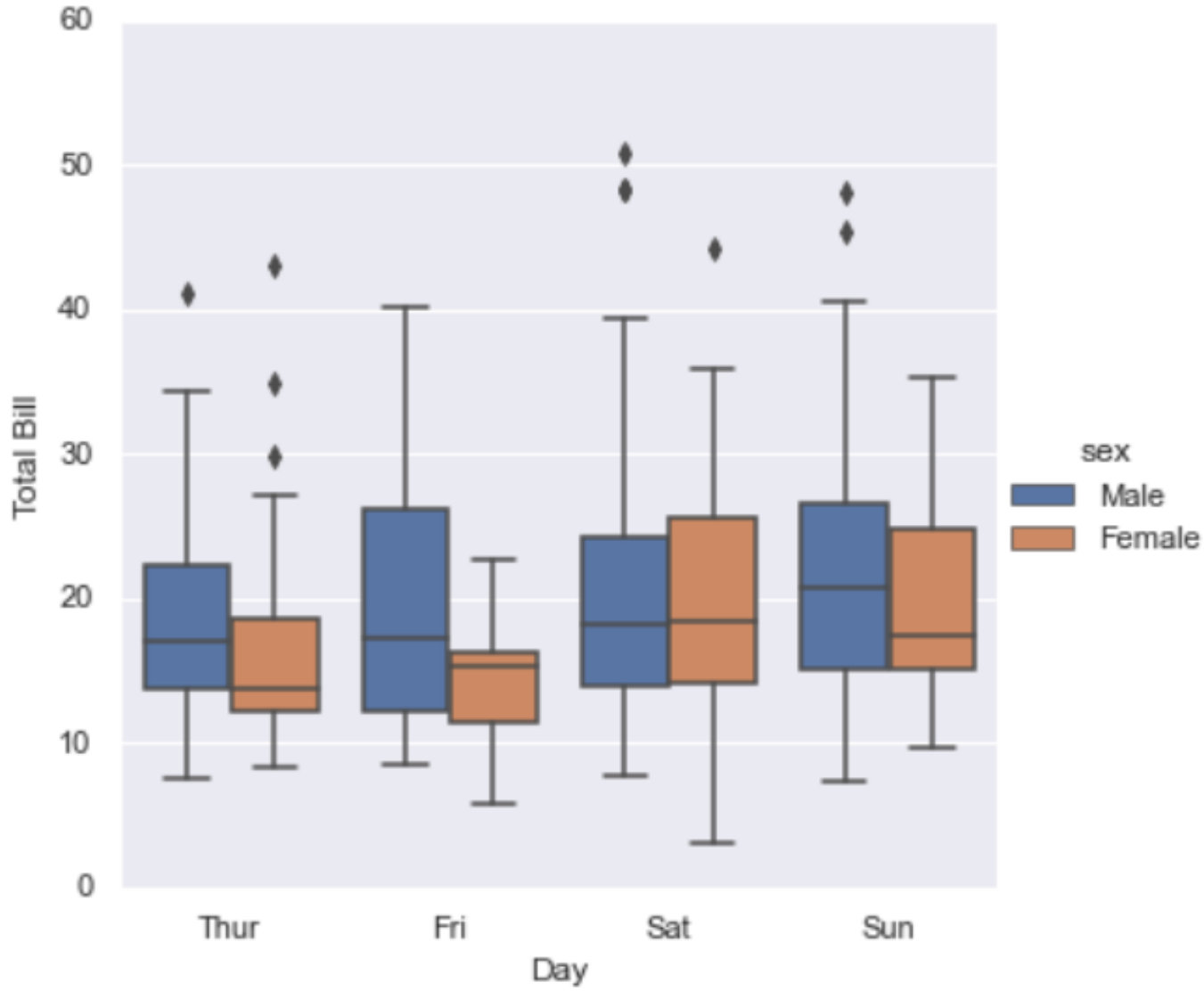
Explanatory
data visualization

We build graphs for others
to communicate and explain the patterns we've found through exploring data.

Exploratory Data Visualization

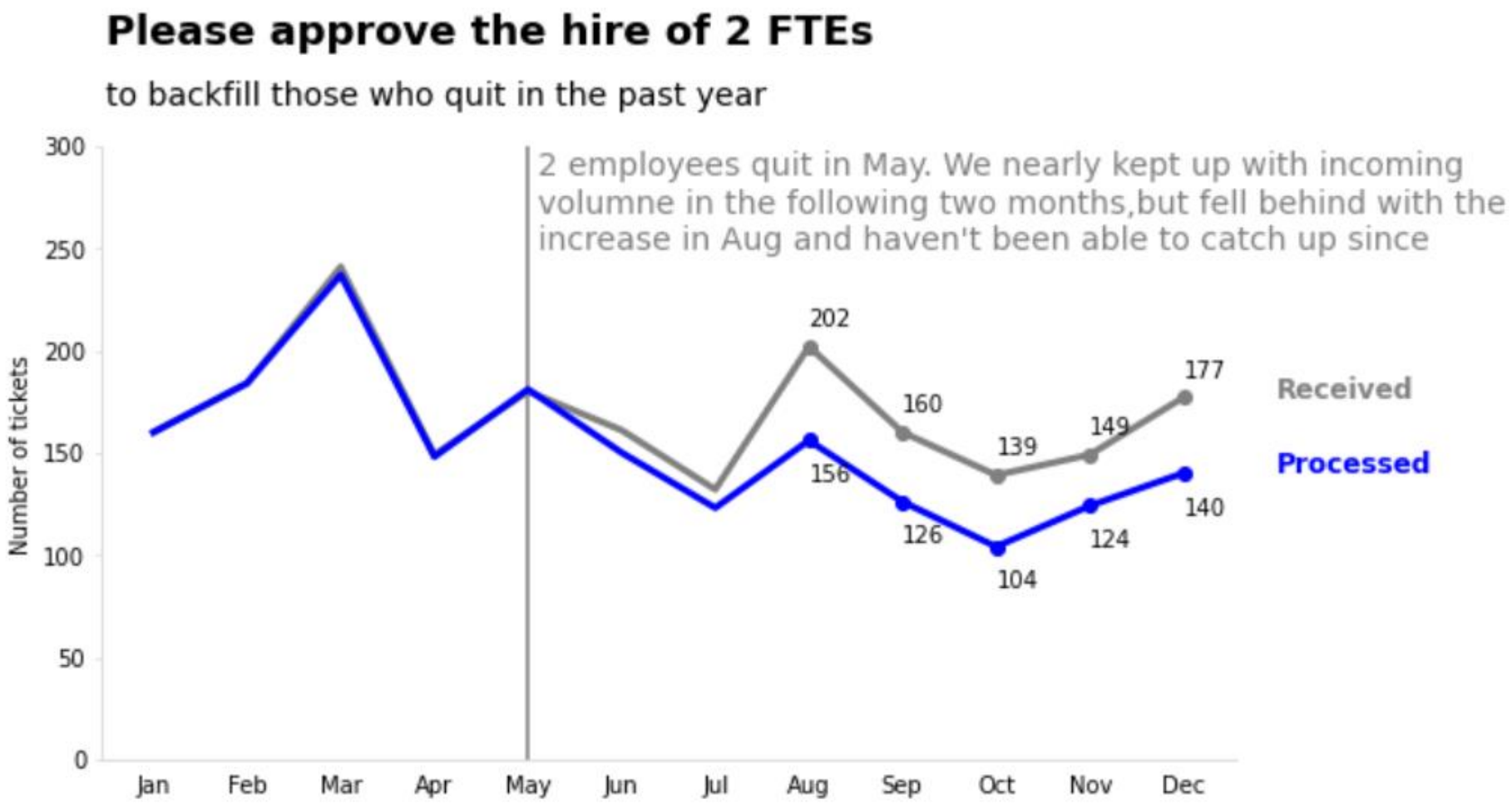
```
tips = sns.load_dataset('tips')
tips.head()
```

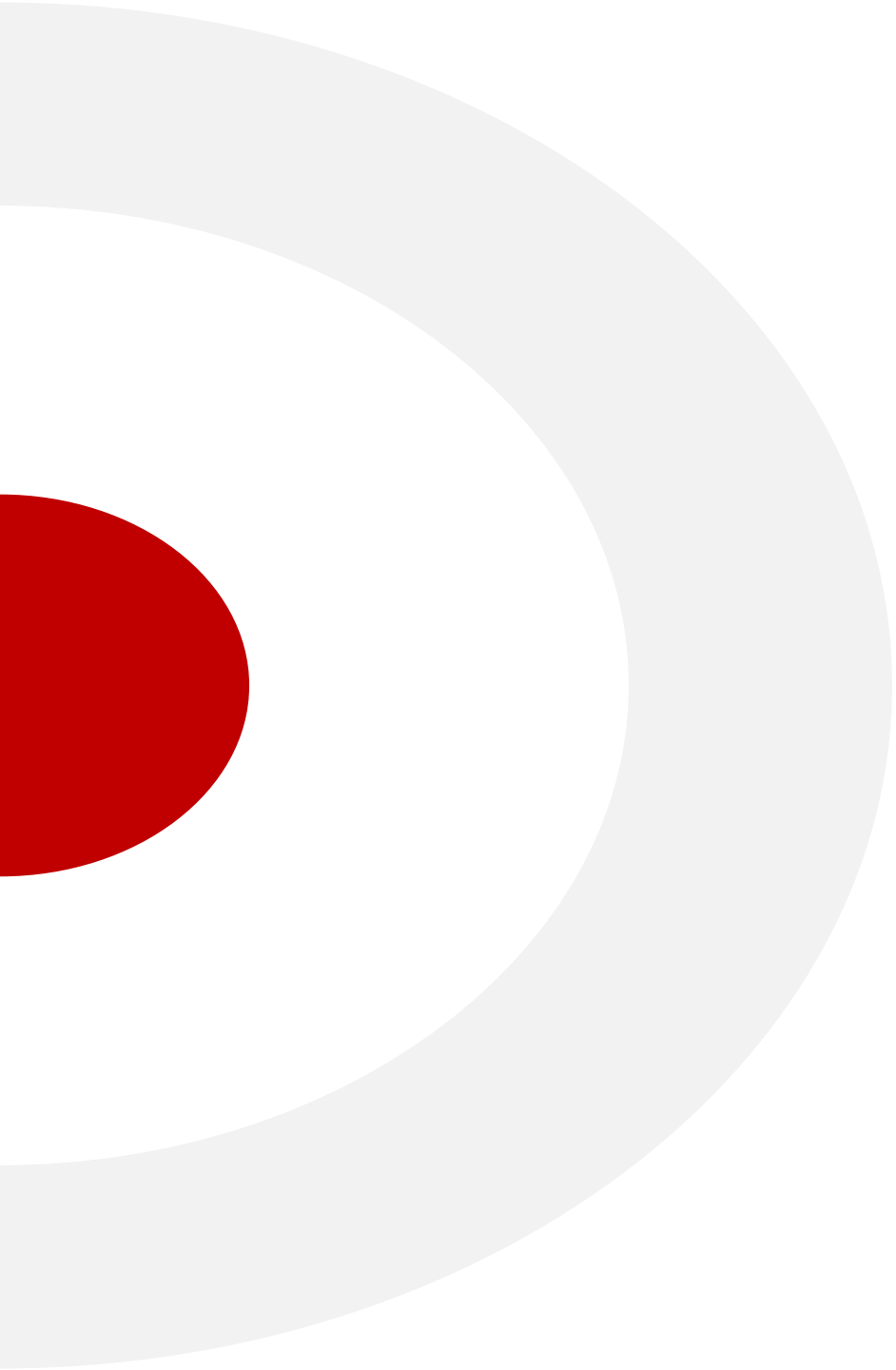
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4



Exploratory Data Visualization

Tell a story

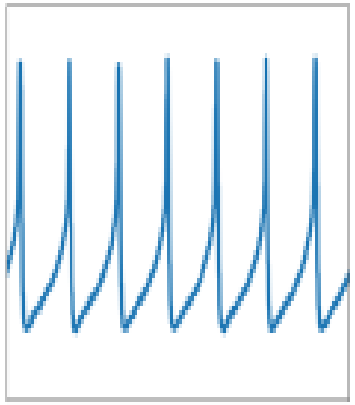
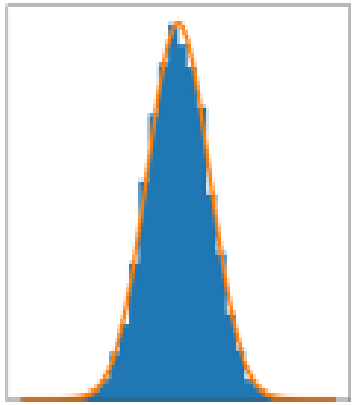
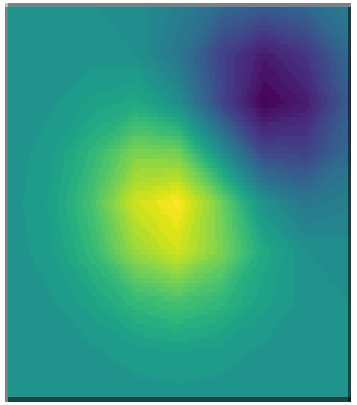
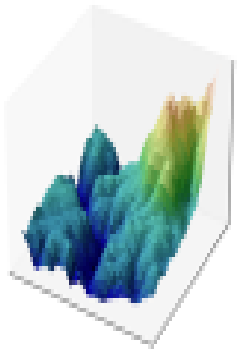




Introduction to Matplotlib

Introduction to Matplotlib

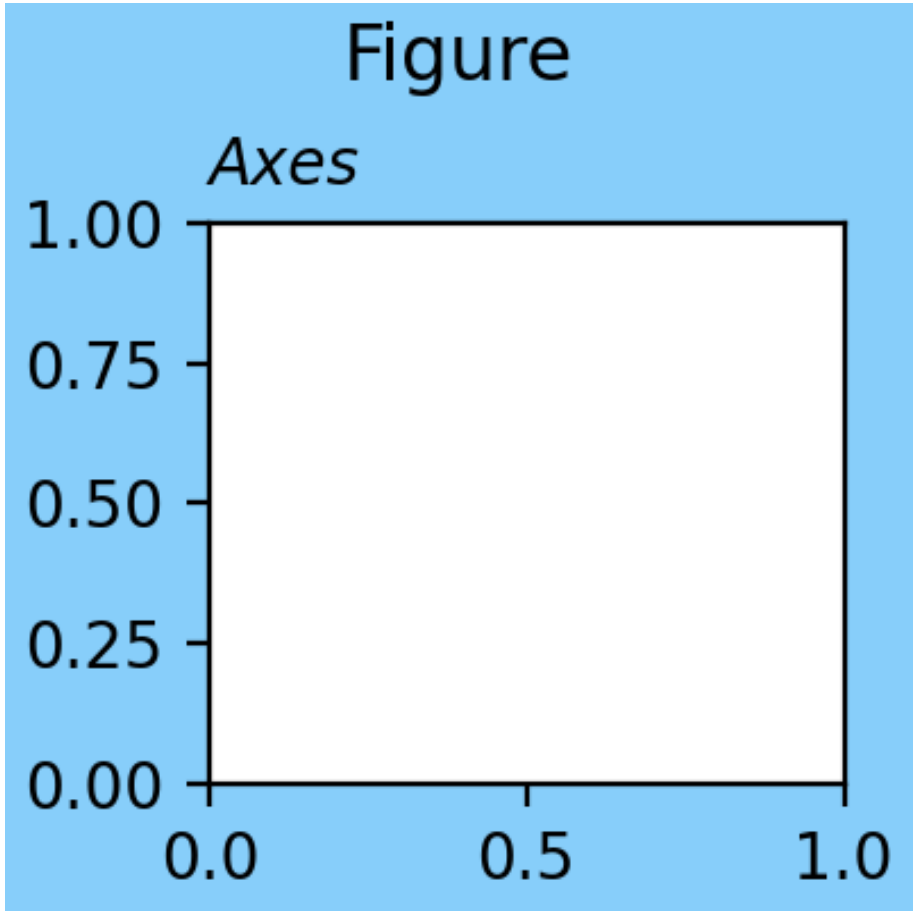
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

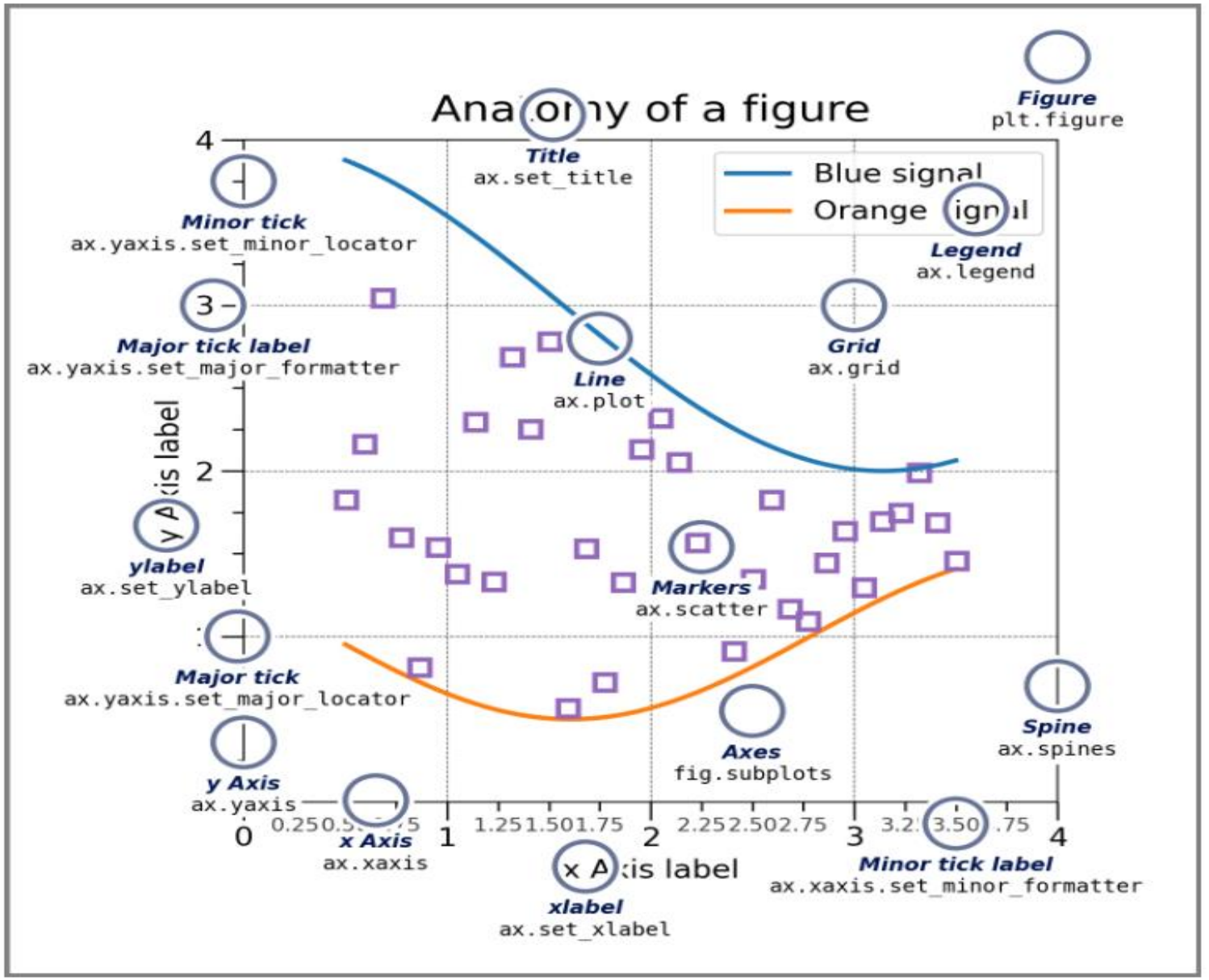
Matplotlib makes easy things easy and hard things possible.

Introduction to Figures

- Figure: Blue zone
- Axes: an Artist attached to a Figure that contains a region for plotting data.
- Axis: set the scale and limits
- Artist: everything visible on the Figure is an Artist



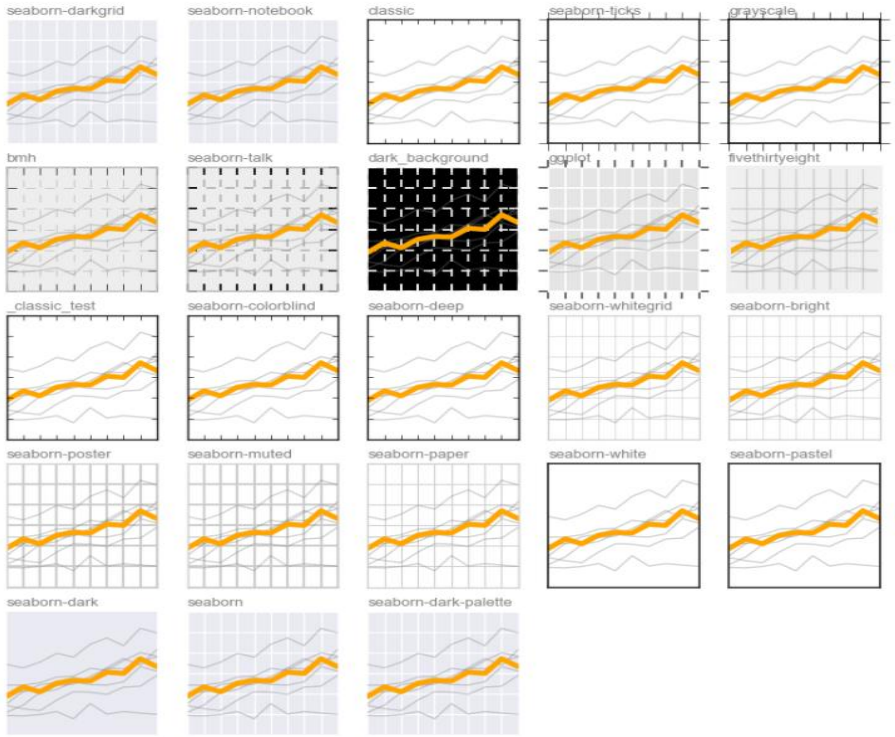
Parts of a Figure



General Matplotlib Tips

- **Importing Matplotlib**

```
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
```



- **Setting Styles**

We will use *the plt.style.use* directive to choose appropriate aesthetic styles for our figures

```
1 plt.style.available
2 plt.style.use('seaborn-whitegrid')
```

Plotting from a script

- Create a file called myplot.py containing the following

```
# ------- file: myplot.py ------
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

- Run this script from the command-line prompt:

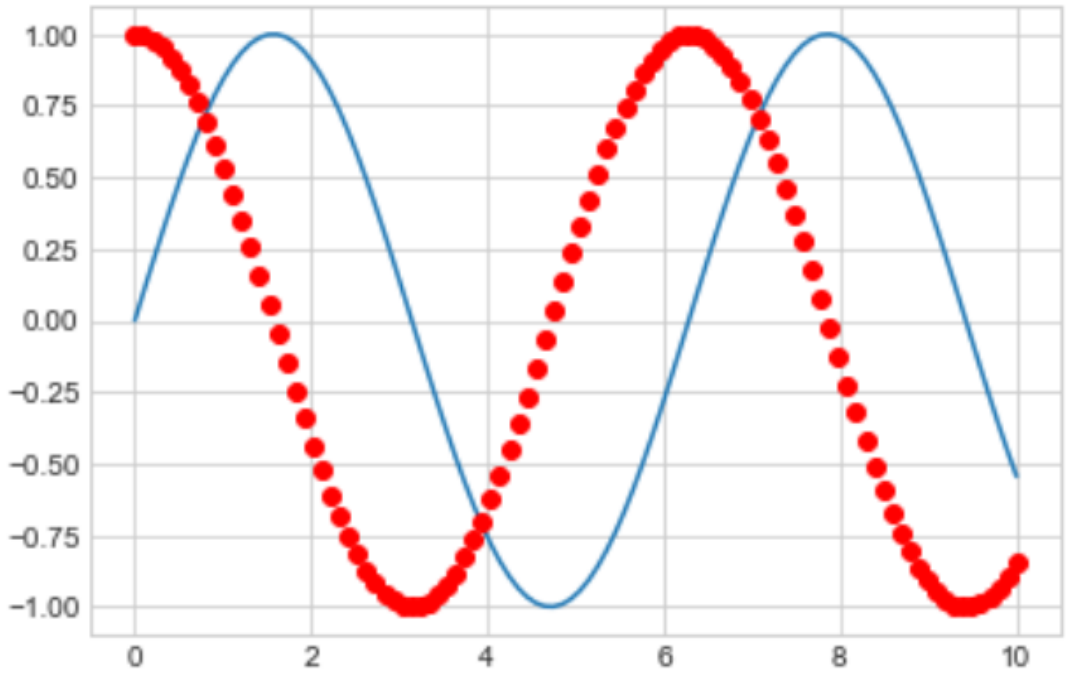
  $python myplot.py

- One thing to be aware of: the plt.show() command should be used only once per Python session, and is most often seen at the very end of the script.

Saving Figures to File

```
x = np.linspace(0,10,100)
```

```
1  fig = plt.figure()
2  plt.plot(x, np.sin(x),'-')
3  plt.plot(x,np.cos(x),'ro');
```




```
fig.savefig('my_figure.png')
```

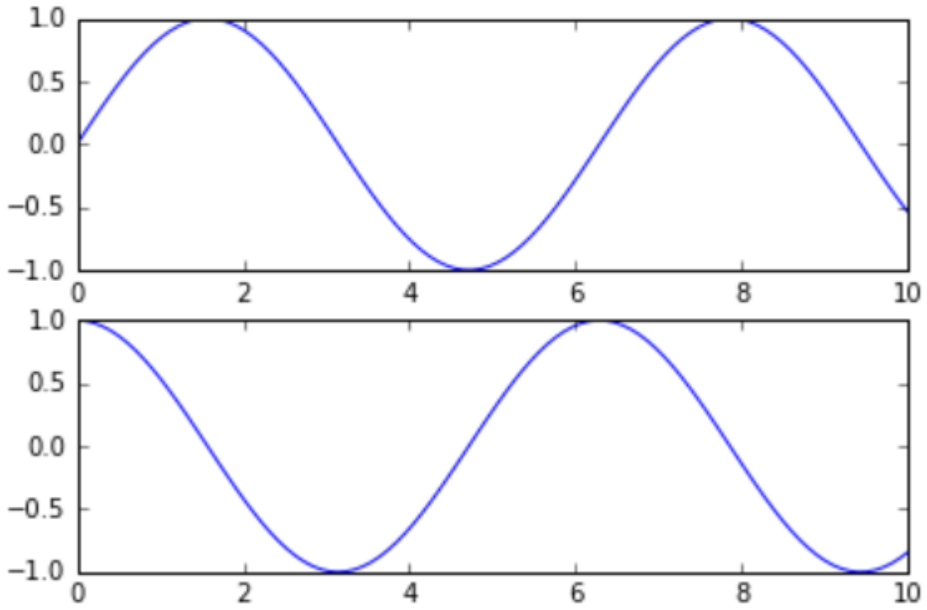
To confirm that it contains what we think it contains, let's use the IPython Image object to display the contents of this file:

```
from IPython.display import Image
Image('my_figure.png')
```

Two Interfaces for the Price of One(I)

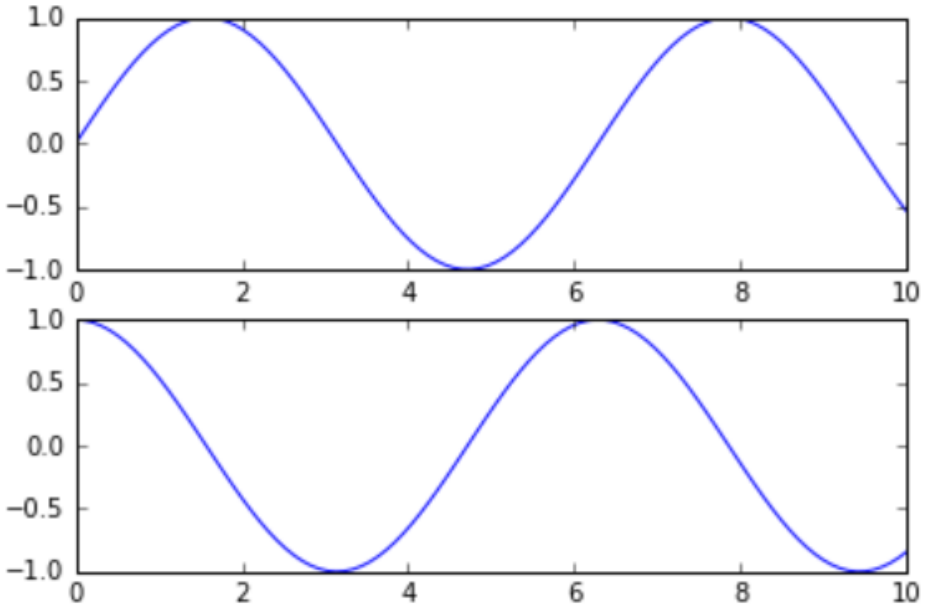
- **MATLAB-style Interface**

```
plt.figure()  # create a plot figure

# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```



- **Object-oriented interface**

```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)

# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
```



Two Interfaces for the Price of One(II)

- To create a graph using the OO interface, we use the plt.subplots() function, which generates an empty plot and returns a tuple of two objects

```
plt.subplots()

fig, ax = plt.subplots()

print(type(fig))

print(type(ax))
```

```
<class 'matplotlib.figure.figure'="">
<class 'matplotlib.axes._subplots.axessubplot'="">
</class></class>
```

Two Interfaces for the Price of One(III)

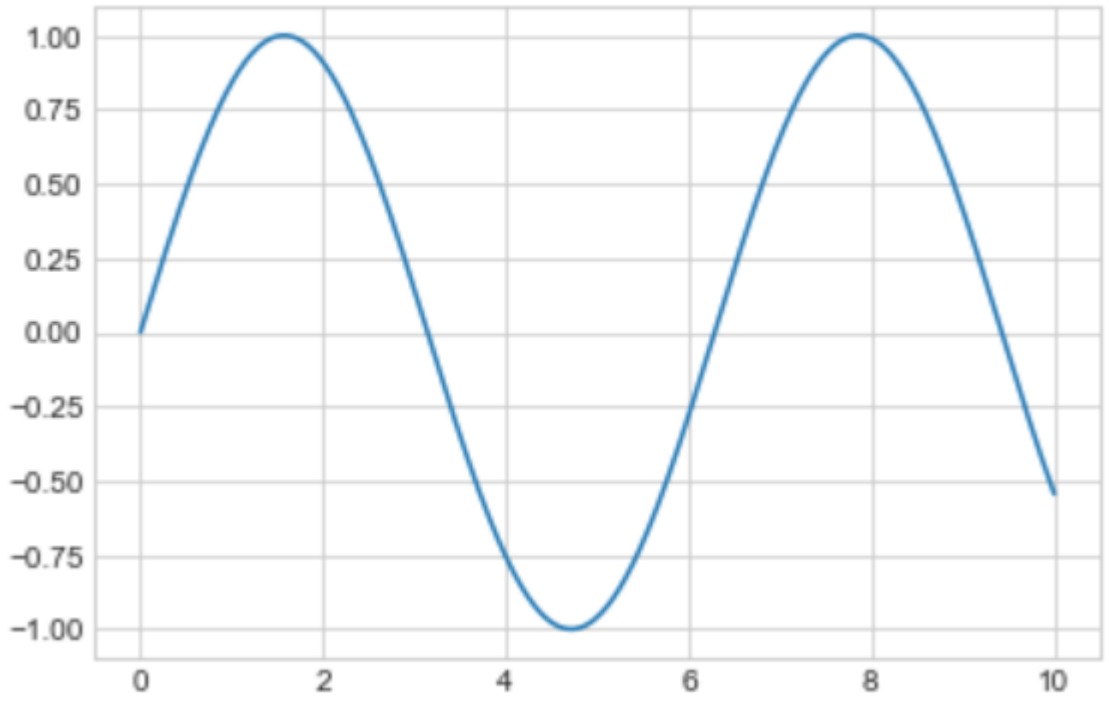
- The matplotlib.figure.Figure object acts as a canvas on which we can add one or more plots
- The matplotlib.axes._subplots.AxesSubplot object is the actual plot
- In short, we have two objects:
  - The Figure (the canvas)
  - The Axes (the plot; don't confuse with "axis", which is the x- and y-axis of a plot)
- To create a bar plot, we use the Axes.bar() method and call plt.show()
- The final code

```
fig, ax = plt.subplots()
ax.bar(['A', 'B', 'C'], [2, 4, 16])
```

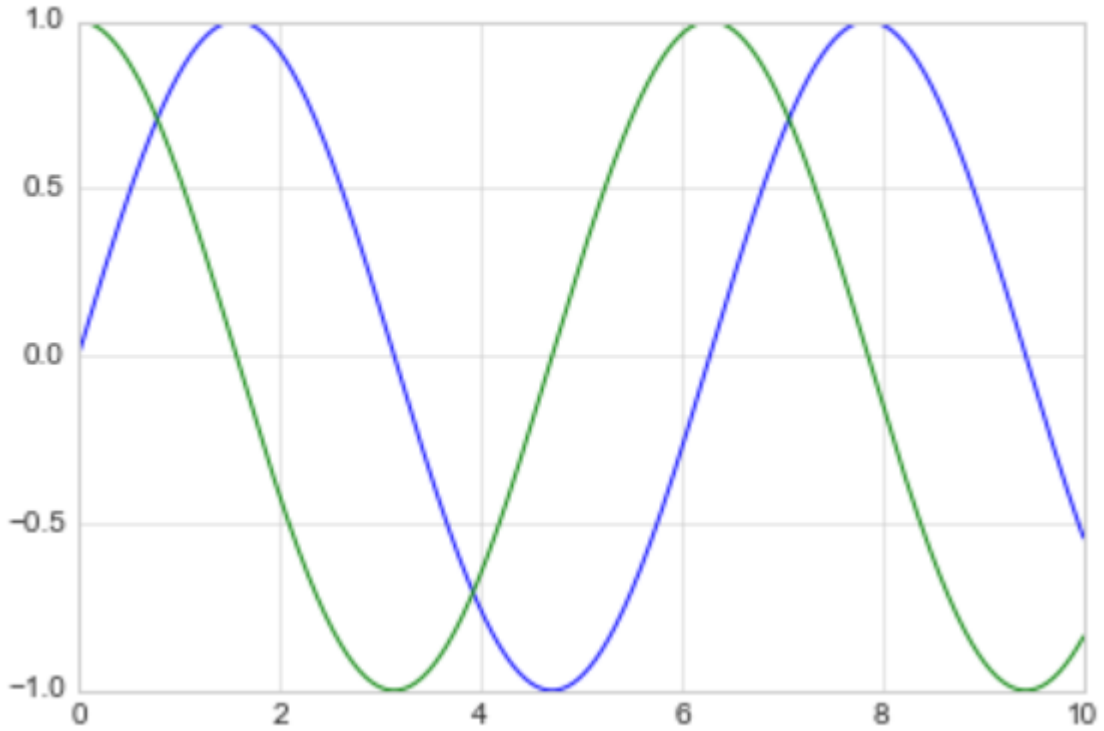
Simple Line Plots

Simple Line Plots

```
plt.plot(x, np.sin(x));
```

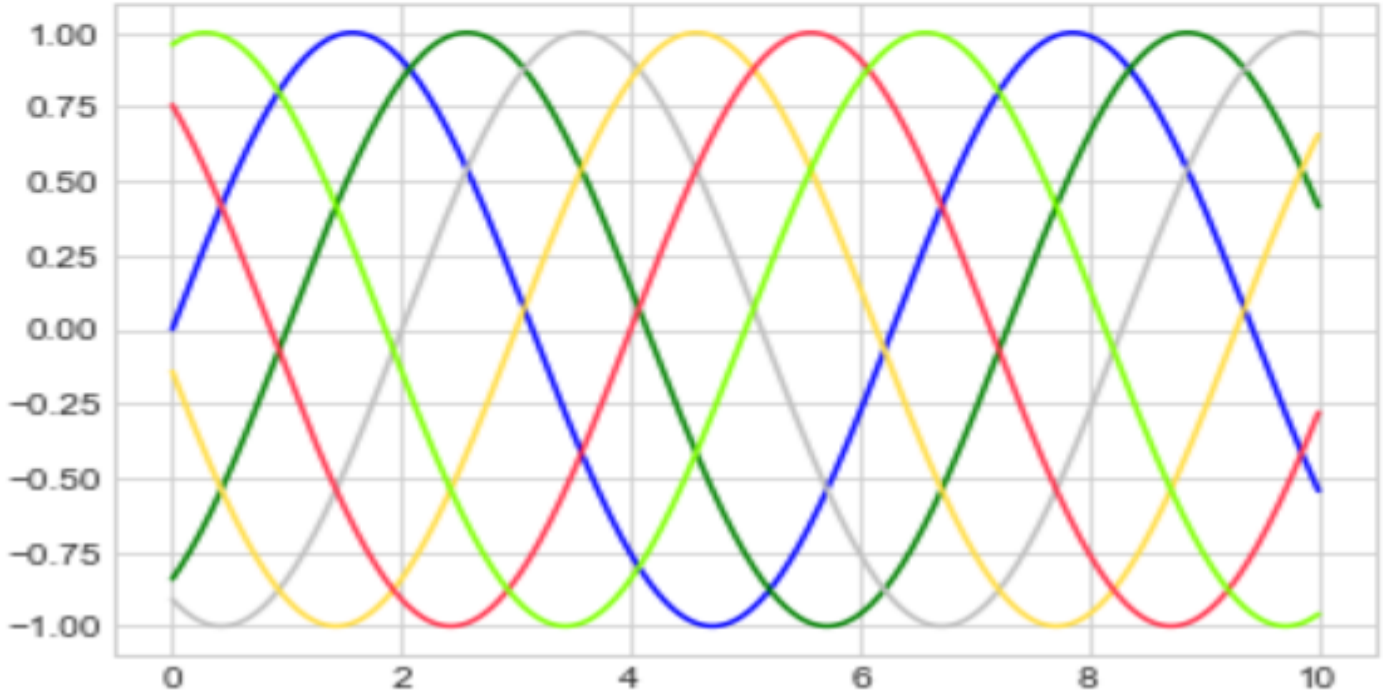



```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
```



Adjusting the Plot: Line Colors and Styles (I)

```
plt.plot(x, np.sin(x - 0), color='blue')          # specify color by name
plt.plot(x, np.sin(x - 1), color='g')             # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')          # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')       # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

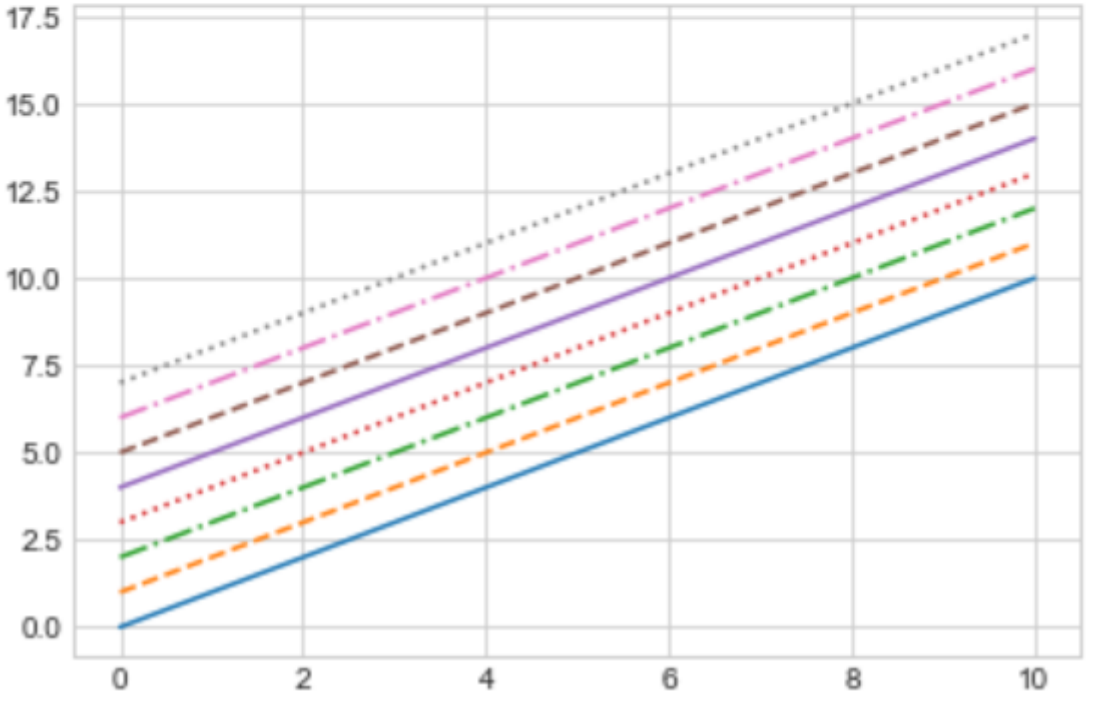


Adjusting the Plot: Line Colors and Styles (II)

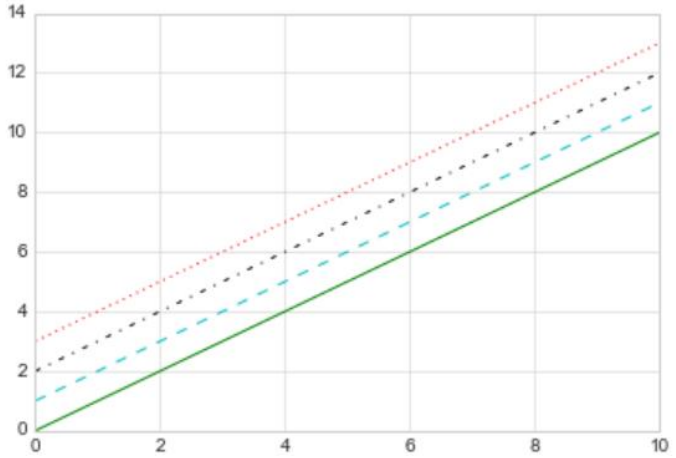
- Similarly, the line style can be adjusted using the linestyle keyword:

- If you would like to be extremely terse, these linestyle and color codes can be combined into a single non-keyword argument to the plt.plot() function:

```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```



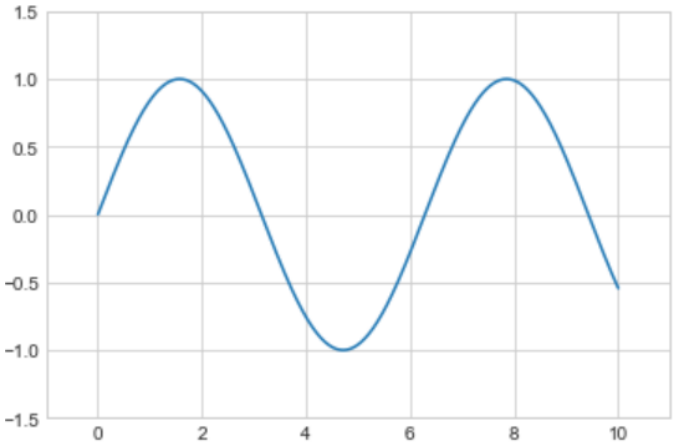

```
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r');  # dotted red
```



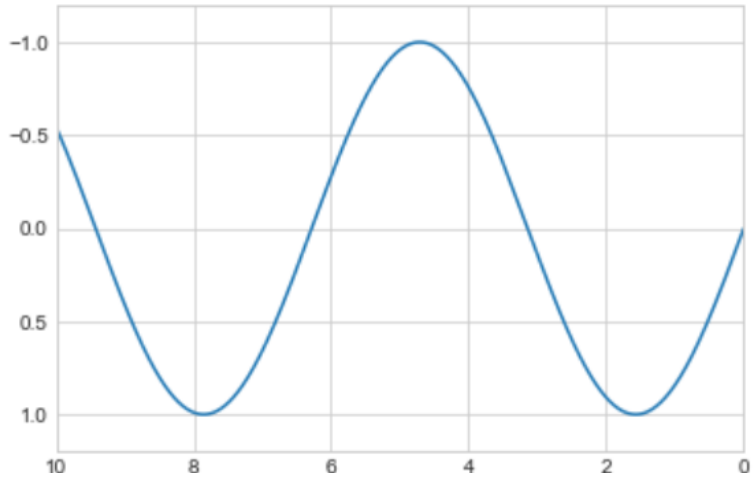
Adjusting the Plot: Axes Limits

Matplotlib does a decent job of choosing default axes limits for your plot, but sometimes it's nice to have finer control. The most basic way to adjust axis limits is to use the plt.xlim() and plt.ylim() methods:

```
plt.plot(x, np.sin(x))

plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```

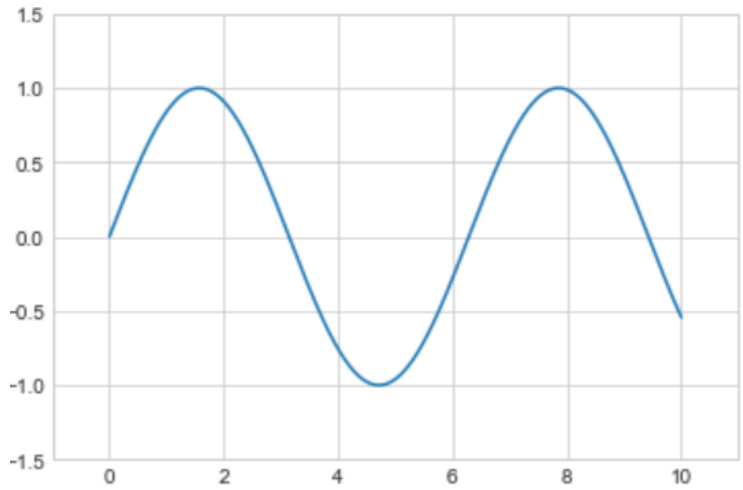



```
plt.plot(x, np.sin(x))

plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```



The plt.axis() method allows you to set the x and y limits with a single call, by passing a list which specifies [xmin, xmax, ymin, ymax]:

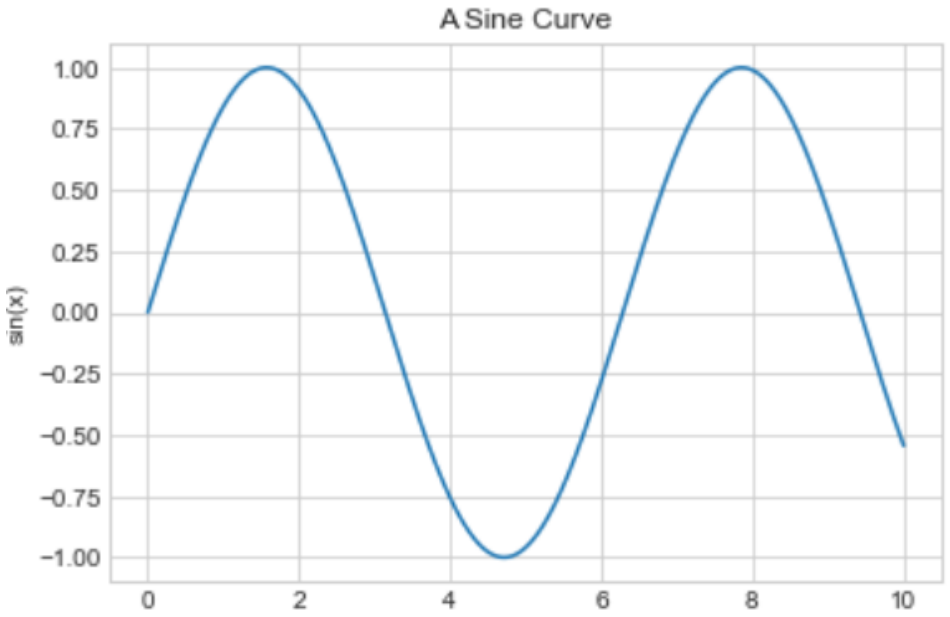
```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```



Labeling Plots

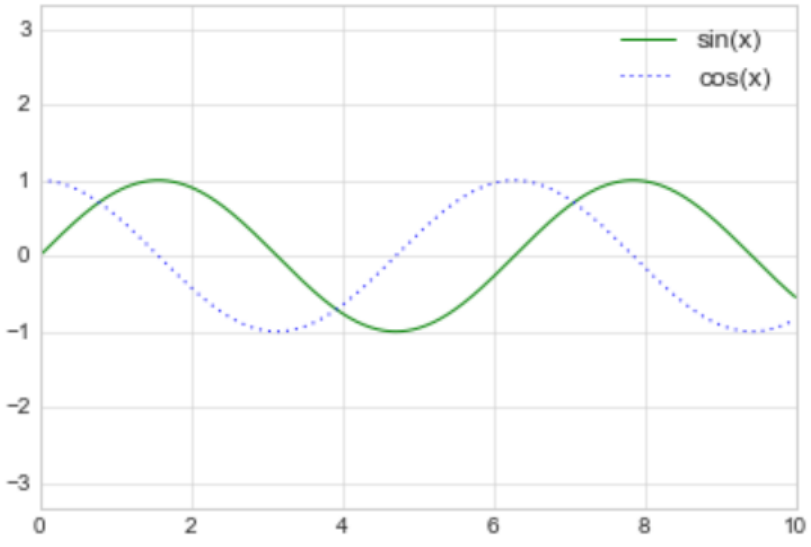
Titles and axis labels are the simplest such labels—there are methods that can be used to quickly set them:

```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```

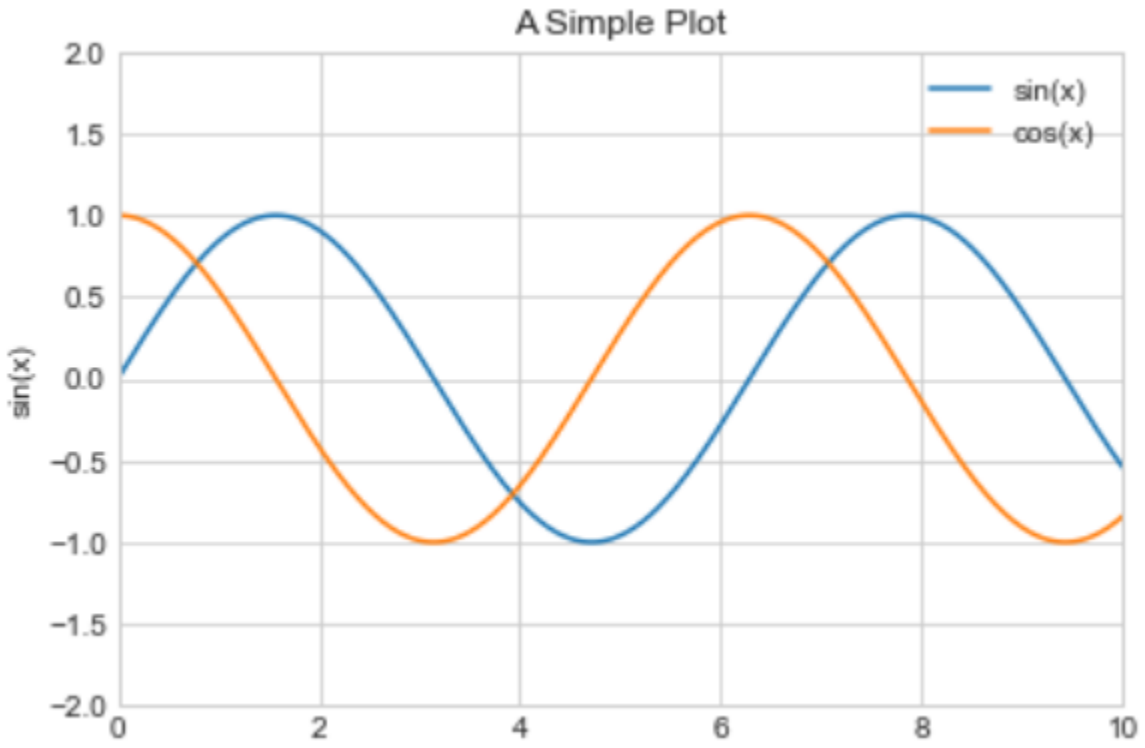


When multiple lines are being shown within a single axes, it can be useful to create a plot legend that labels each line type. Again, Matplotlib has a built-in way of quickly creating such a legend. It is done via the plt.legend() method.

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')

plt.legend();
```



Practice: Plotting with Object-oriented interface



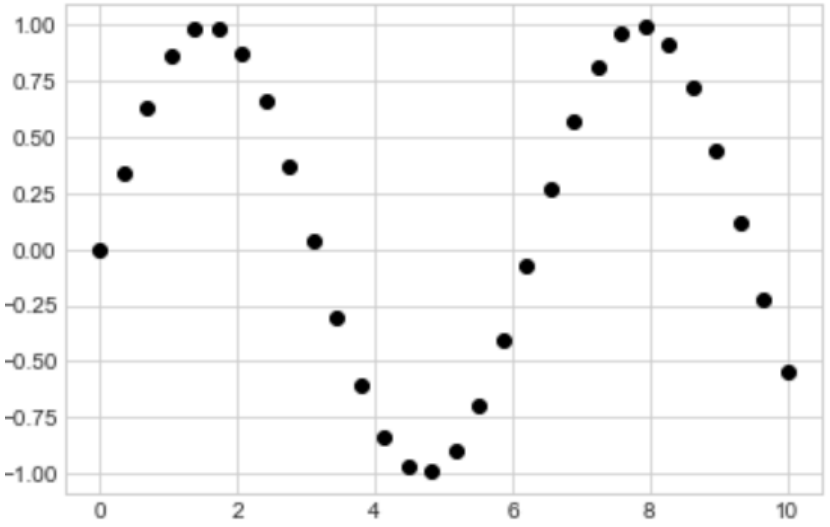
- While most plt functions translate directly to ax methods (such as plt.plot() →ax.plot(), plt.legend() → ax.legend(), etc.); functions to set limits, labels, and titles are slightly modified:
  - *plt.xlabel() → ax.set_xlabel()*
  - *plt.ylabel() → ax.set_ylabel()*
  - *plt.xlim() → ax.set_xlim()*
  - *plt.ylim() → ax.set_ylim()*
  - *plt.title() → ax.set_title()*

```
ax = plt.axes()
ax.plot(x, np.sin(x), label = "sin(x)" )
ax.plot(x, np.cos(x), label = "cos(x)")
ax.legend()
ax.set(xlim=(0, 10), ylim=(-2, 2),
       xlabel='x', ylabel='sin(x)',
       title='A Simple Plot');
```

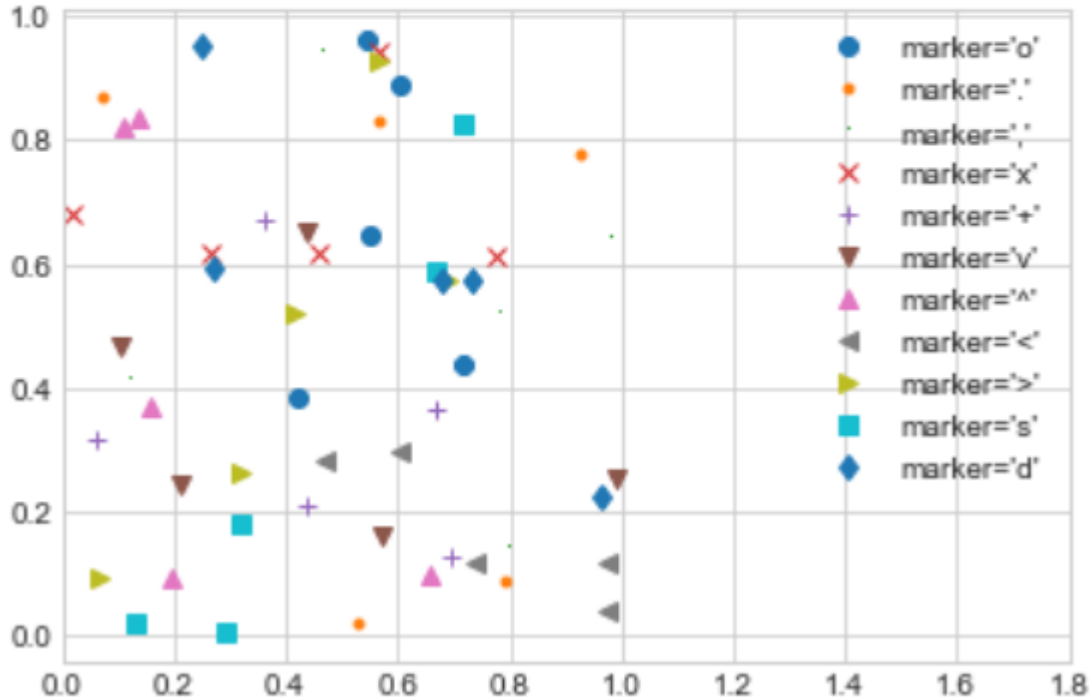
Scatter Plots with plt.plot

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

```
x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, 'o', color='black');
```




```
rng = np.random.RandomState(0)
for marker in ['o', '.', ',', 'x', '+', 'v',
               '^', '<', '>', 's', 'd']:
    plt.plot(rng.rand(5), rng.rand(5), marker,
             label="marker='{0}'".format(marker))
plt.legend(numpoints=1)
plt.xlim(0, 1.8);
```

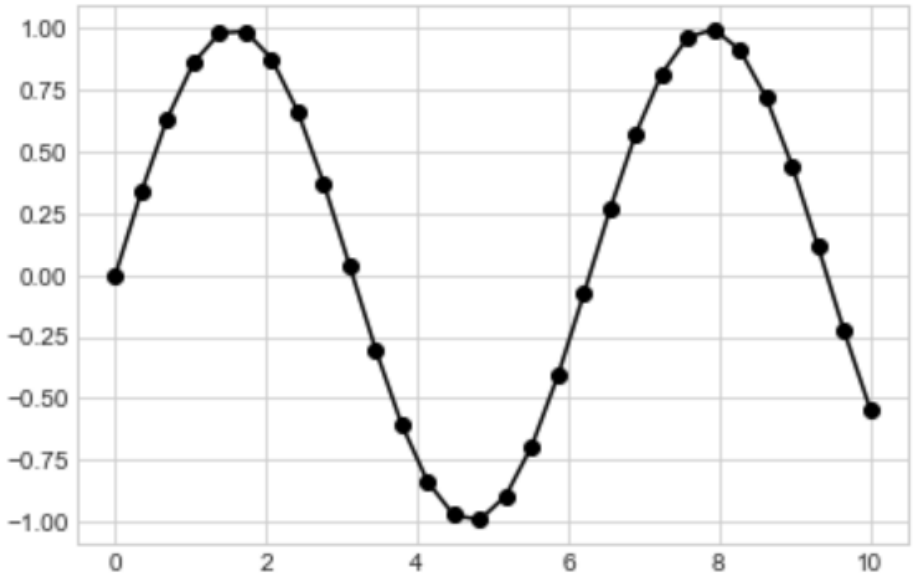


The character that represents the type of symbol used for the plotting.

Simple Scatter Plots

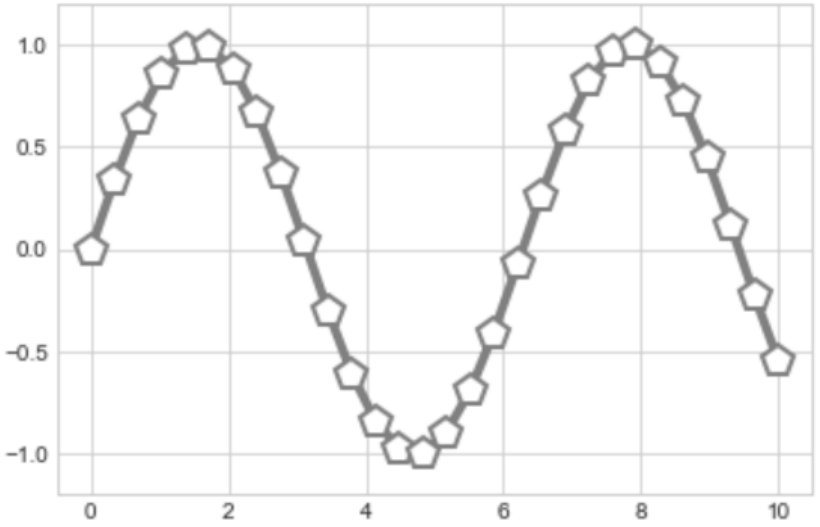
For even more possibilities, these character codes can be used together with line and color codes to plot points along with a line connecting them:

```
plt.plot(x, y, '-ok');
```



Additional keyword arguments to plt.plot specify a wide range of properties of the lines and markers:

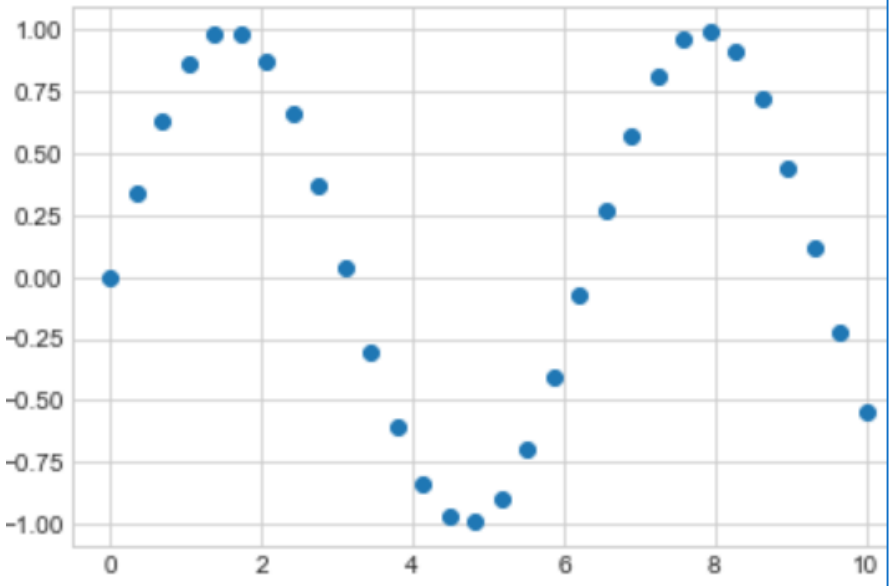
```
plt.plot(x, y, '-p', color='gray',
         markersize=15, linewidth=4,
         markerfacecolor='white',
         markeredgecolor='gray',
         markeredgewidth=2)
plt.ylim(-1.2, 1.2);
```



Scatter Plots with plt.scatter

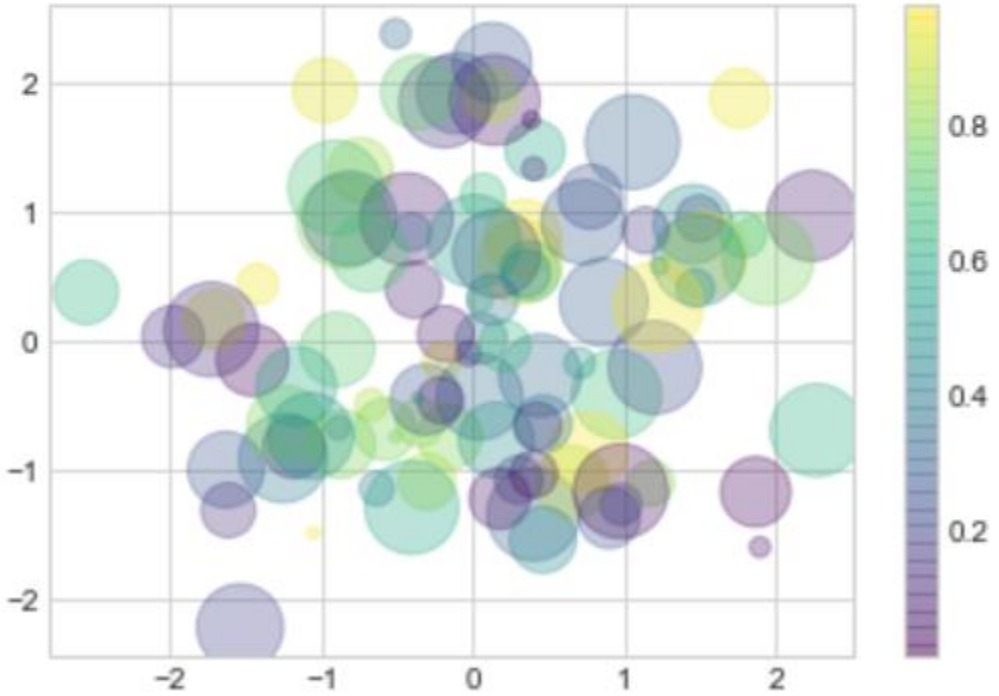
A second, more powerful method of creating scatter plots is the *plt.scatter* function, which can be used very similarly to the *plt.plot* function:

```
plt.scatter(x, y, marker='o');
```



The primary difference of plt.scatter from plt.plot is that it can be used to create scatter plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or

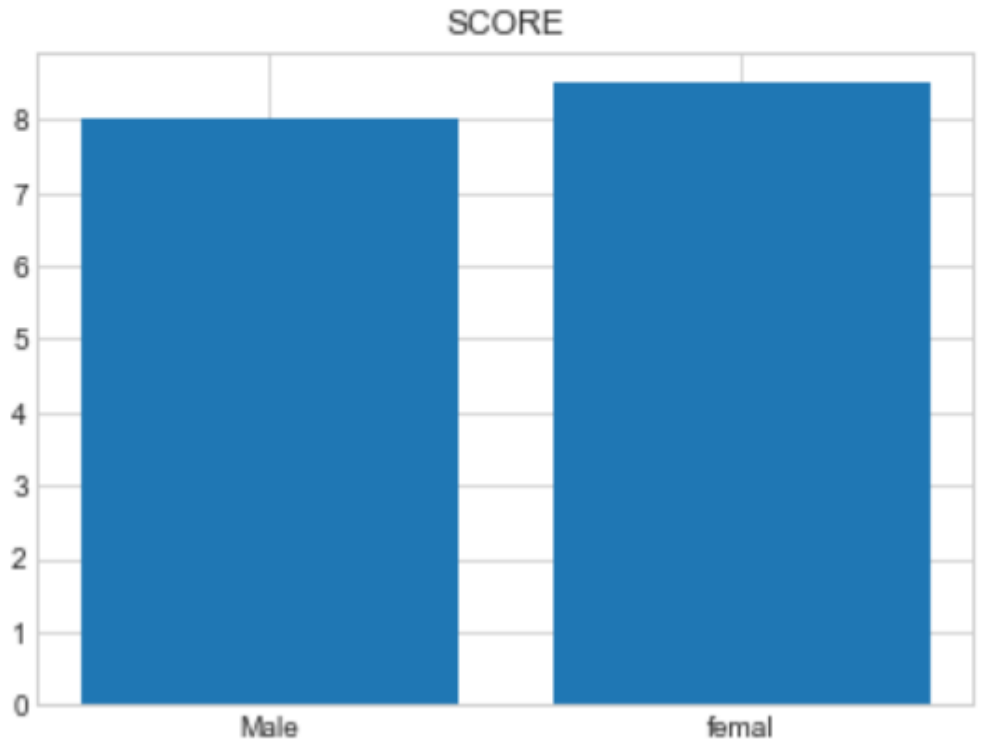
```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
            cmap='viridis')
plt.colorbar();  # show color scale
```



Bar

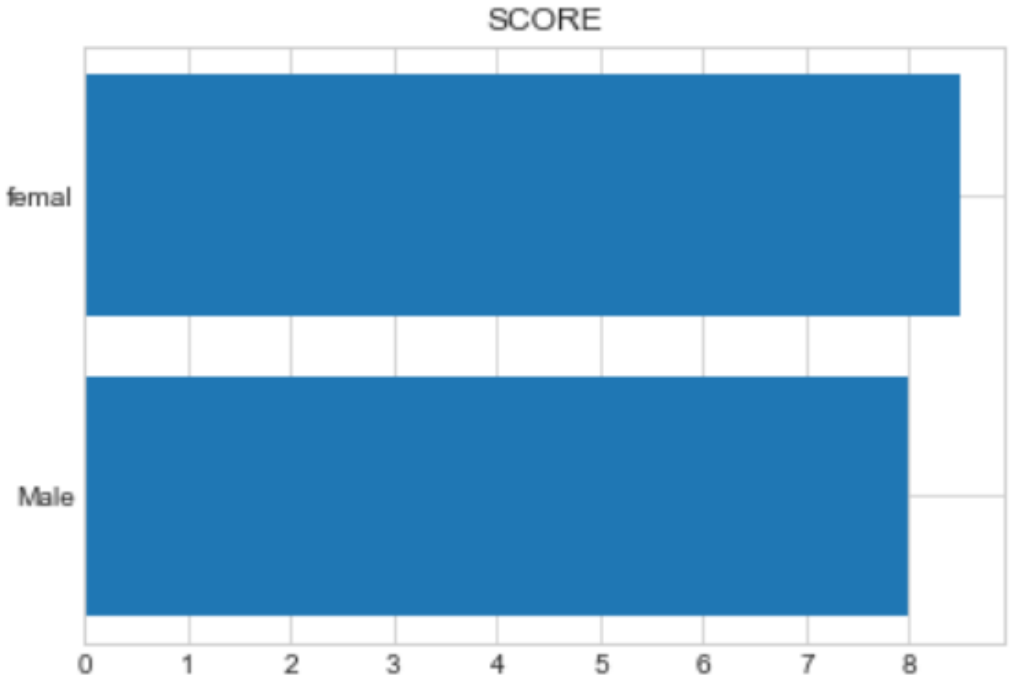
- Vertical

```
fig, ax = plt.subplots()
ax.bar(total_score.keys(),
       total_score.values())
ax.set_title("SCORE");
```



- Horizontal

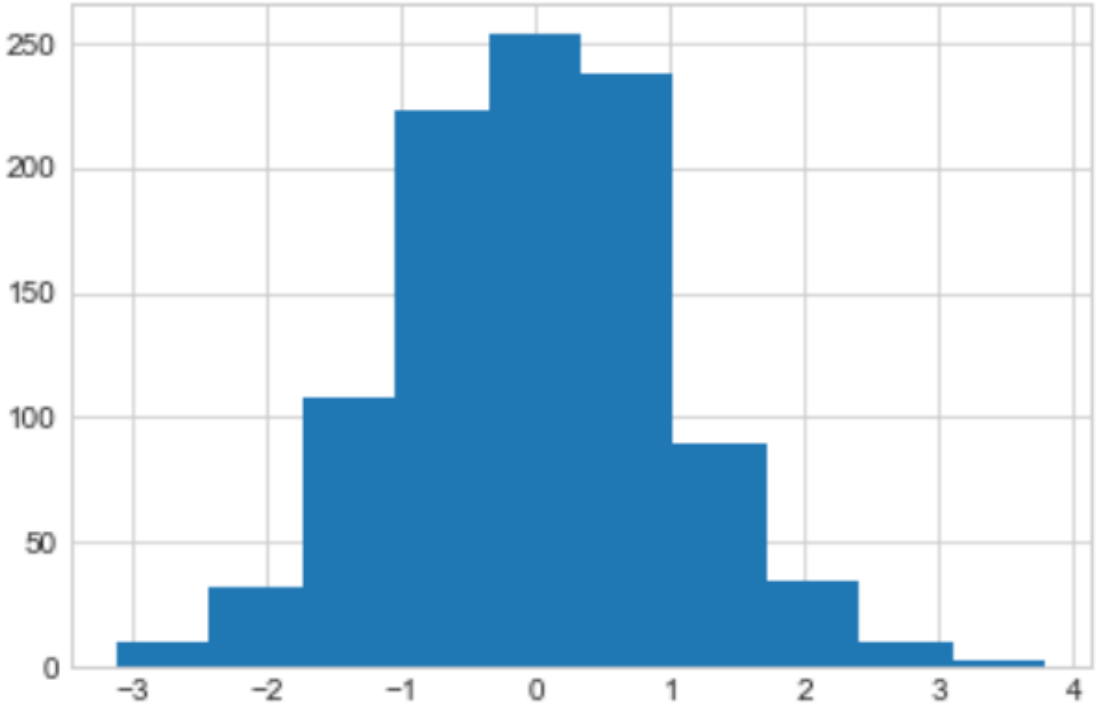
```
fig, ax = plt.subplots()
ax.barh(list(total_score.keys()),
        list(total_score.values()));
ax.set_title("SCORE");
```



Histograms

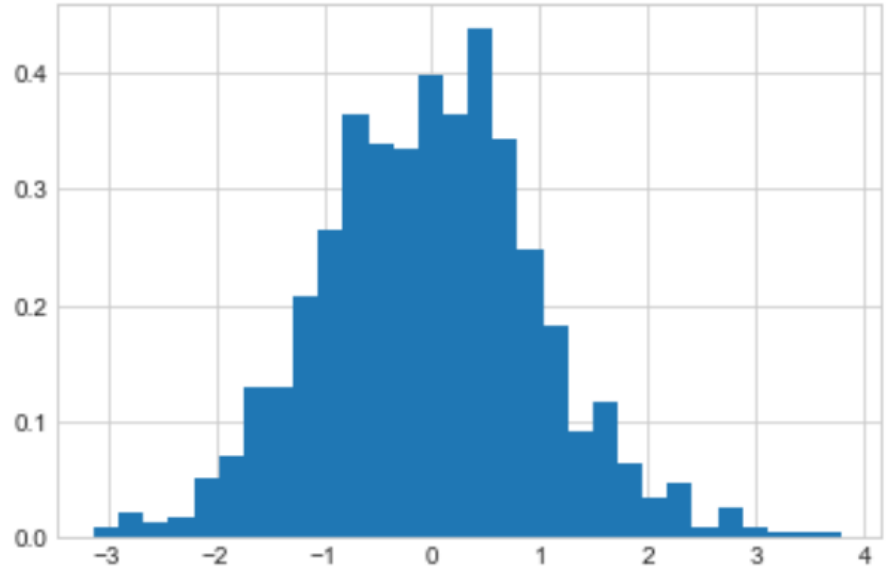
A simple histogram can be a great first step in understanding a dataset.

```
plt.hist(data);
```



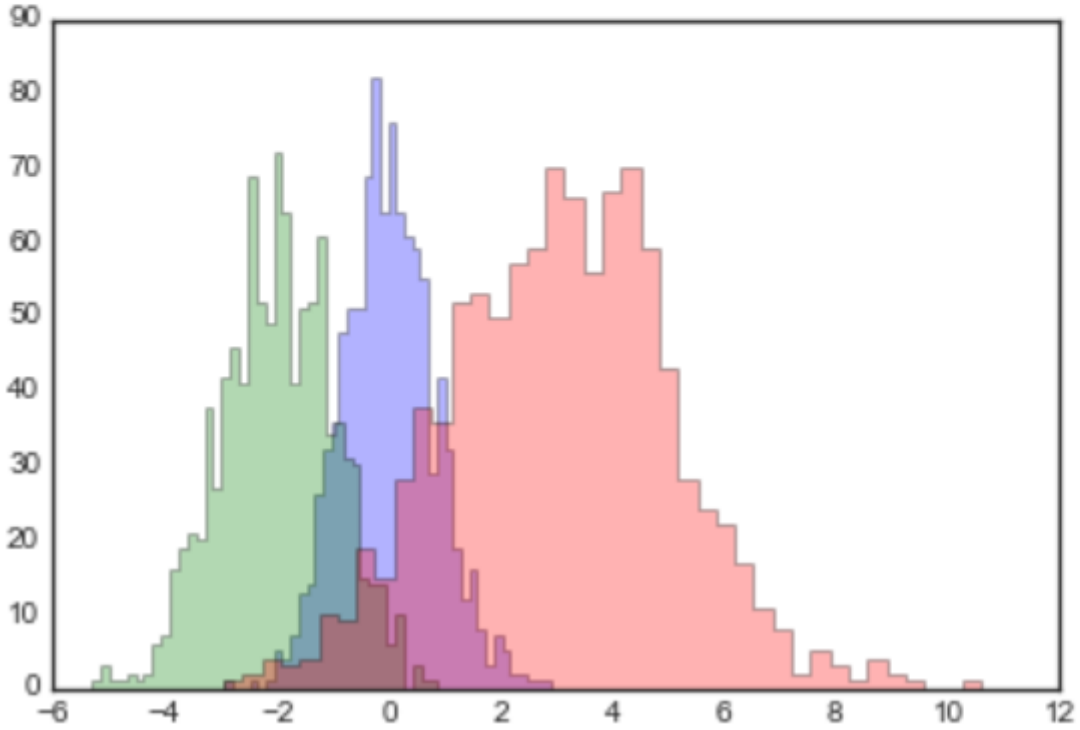
The hist() function has many options to tune both the calculation and the display;

```
plt.hist(data,
         bins=30,
         density=True);
```



Histograms

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40)

plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```



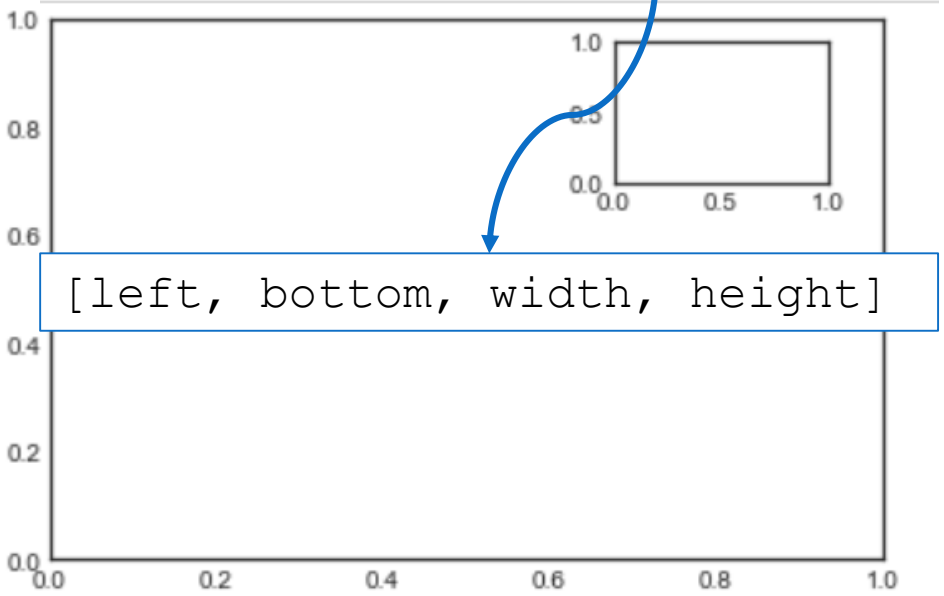
Multiple Subplots

Sometimes it is helpful to compare different views of data side by side. To this end, Matplotlib has the concept of subplots: groups of smaller axes that can exist together within a single figure.

plt.axes: Subplots by Hand

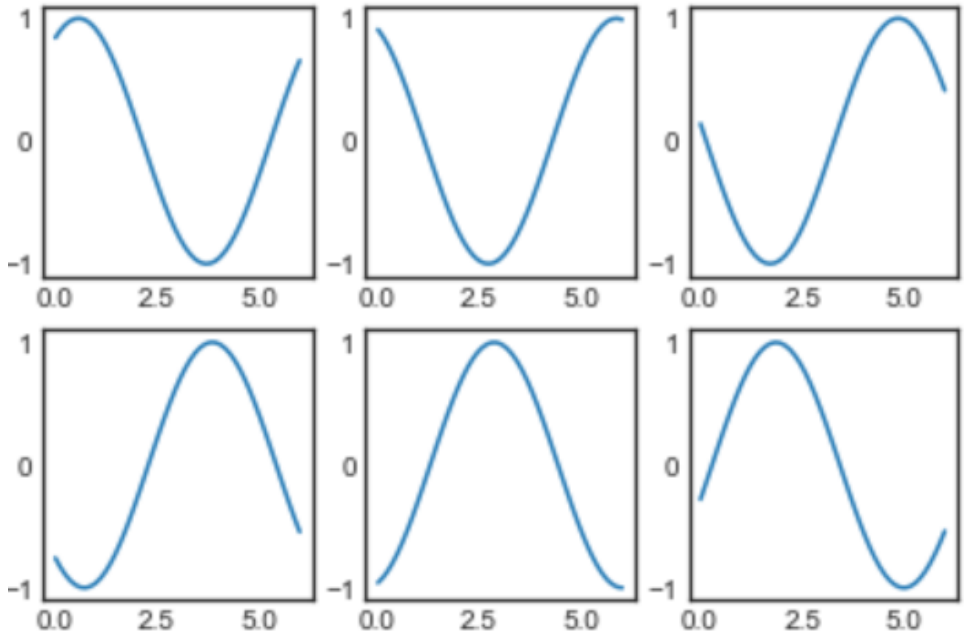
The most basic method of creating an axes is to use the plt.axes function.

```
ax1 = plt.axes()  # standard axes
ax2 = plt.axes([0.65, 0.65, 0.2, 0.2])
```



plt.subplot: Simple Grids of Subplot

```
x = np.linspace(0,6,50)
for i in range(1, 7):
    plt.subplot(2, 3, i)
    plt.plot(x, np.sin(x+i))
```

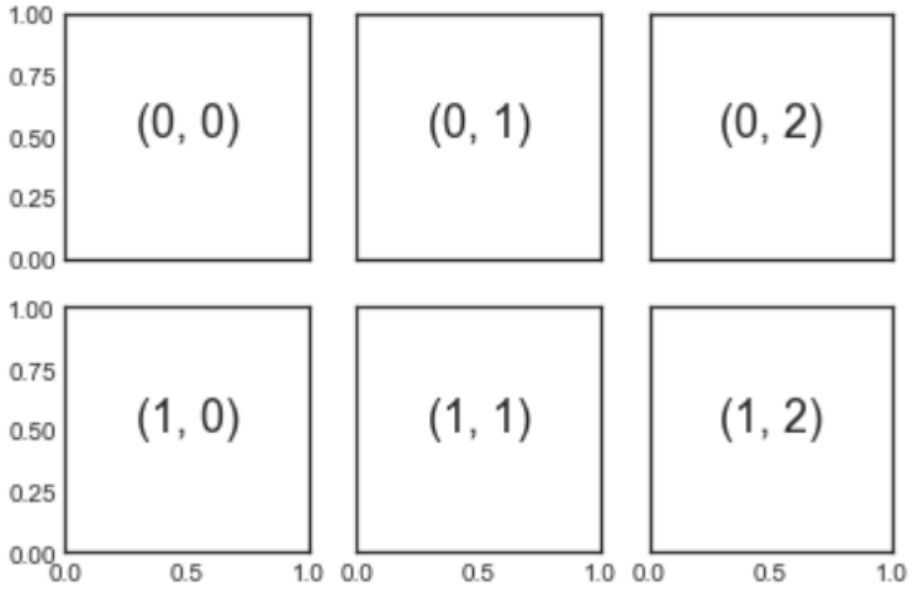


Multiple Subplots

plt.subplots: The Whole Grid in One Go

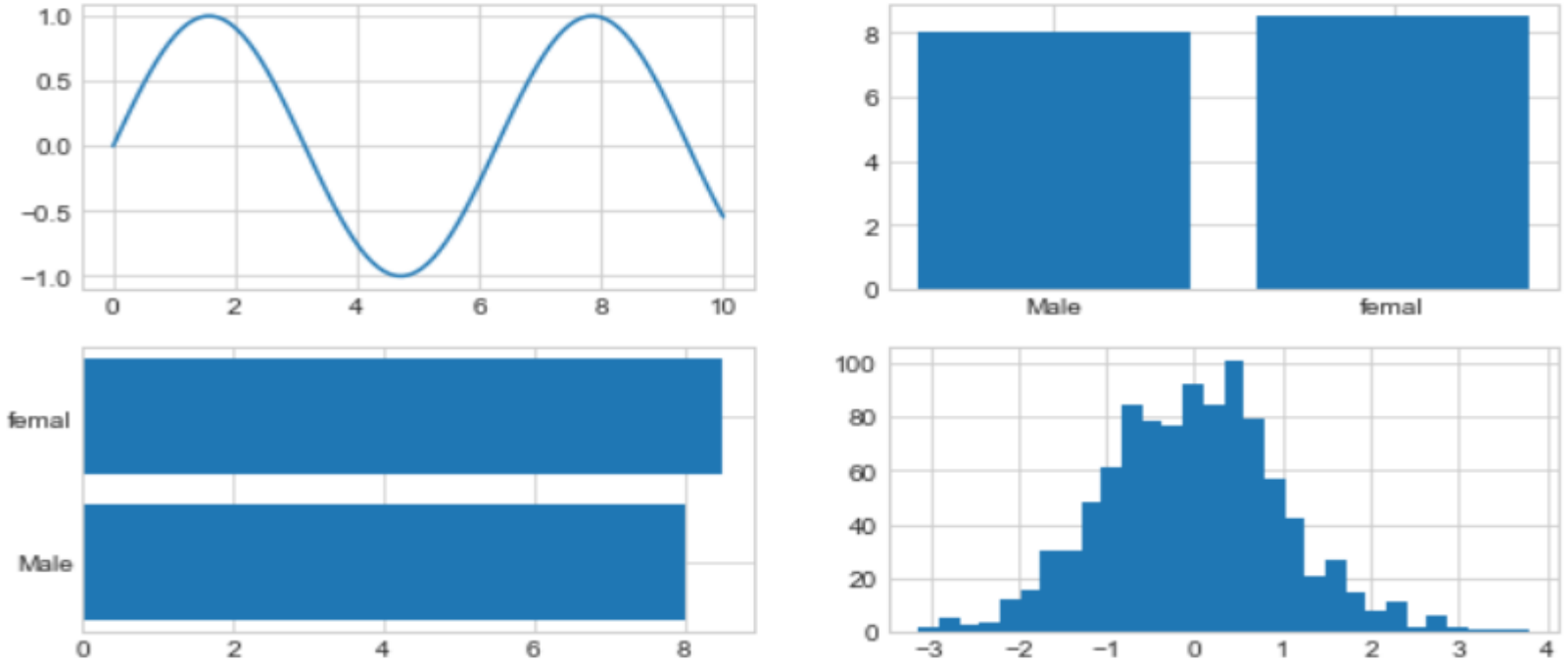
```
fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')
```

```
# axes are in a two-dimensional array, indexed by [row, col]
for i in range(2):
    for j in range(3):
        ax[i, j].text(0.5, 0.5, str((i, j)),
                      fontsize=18, ha='center')
fig
```



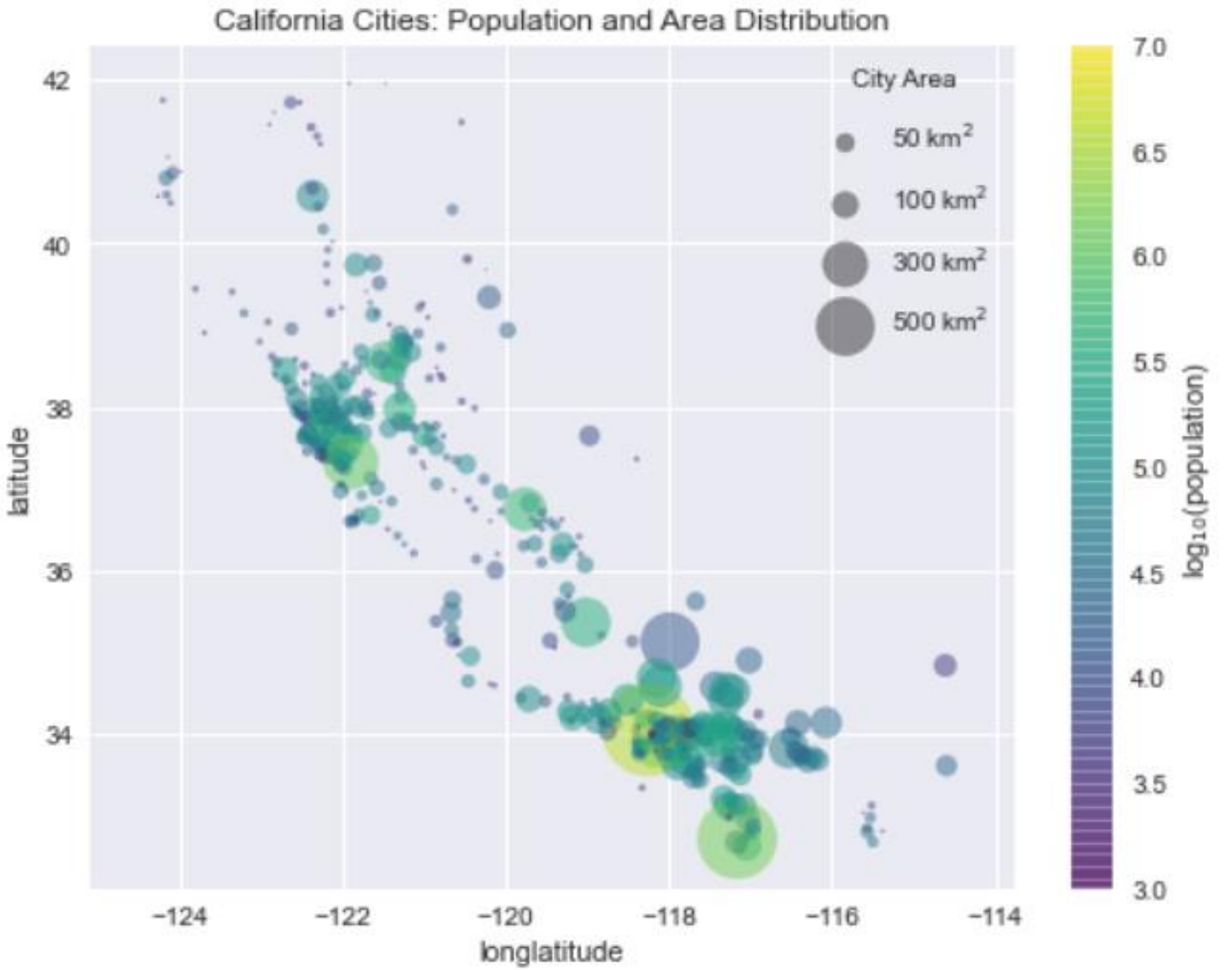
Multiple Subplots

```
fig, ax = plt.subplots(nrows=2,
                       ncols=2,
                       figsize = (10,5))
ax[0,0].plot(x, np.sin(x))
ax[0,1].bar(total_score.keys(),
       total_score.values())
ax[1,0].barh(list(total_score.keys()),
             list(total_score.values()))
ax[1,1].hist(data,bins =30);
```



Practice: Data Visualization - California Cities

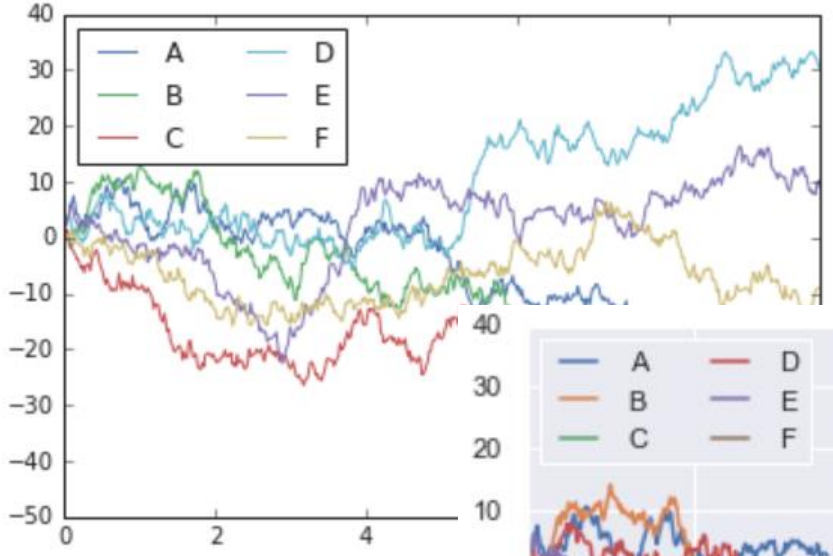
```
cities = pd.read_csv("california_cities.csv")
```

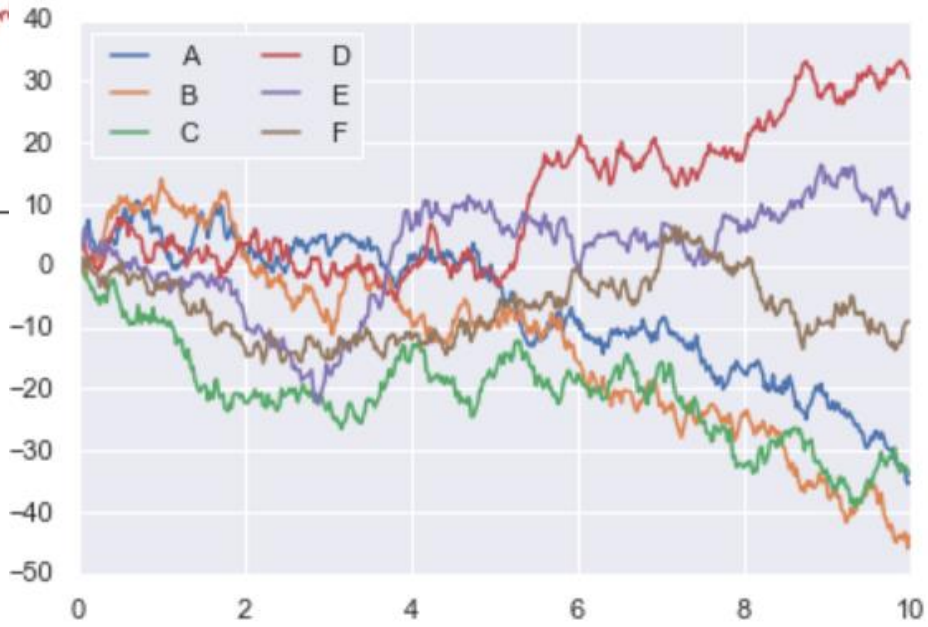


Introduction to Seaborn

The Python visualization library Seaborn is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics. Make use of the following aliases to import the libraries:

```
# import three necessary libraries
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# to ignore the warnings
from warnings import filterwarnings
```





The basic steps to creating plots w
Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

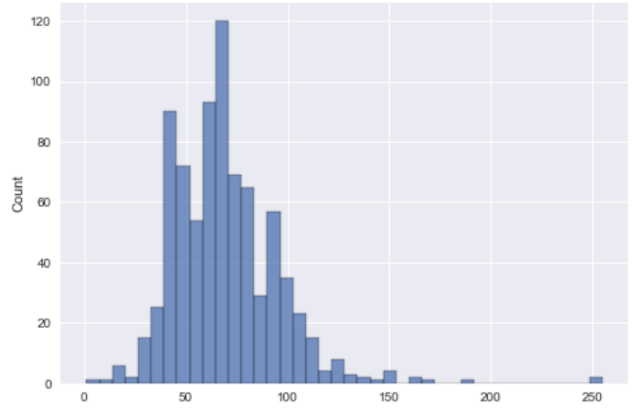
Distribution: Hist, KDE

```
pokemon_df = pd.read_csv("Pokemon.csv")
```

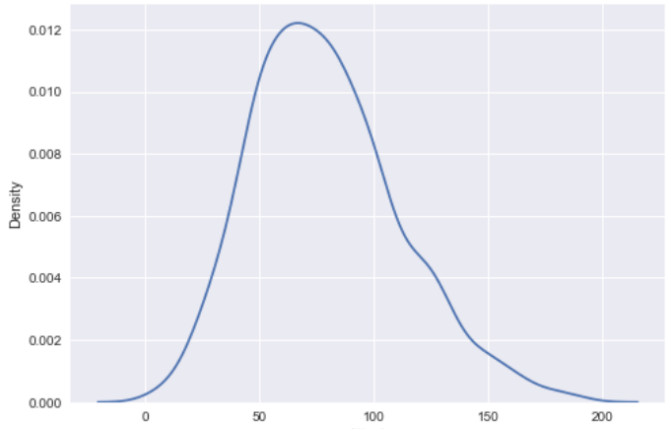
```
pokemon_df.head()
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

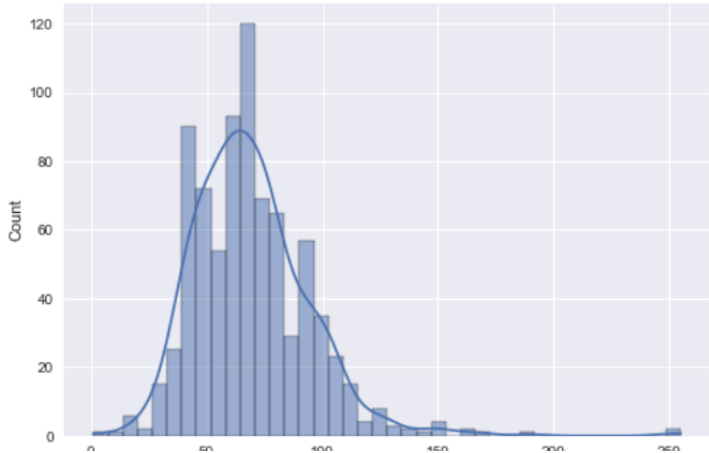
```
sns.histplot(pokemon_df['HP']);
```



sns.kdeplot(pokemon_df.Attack);

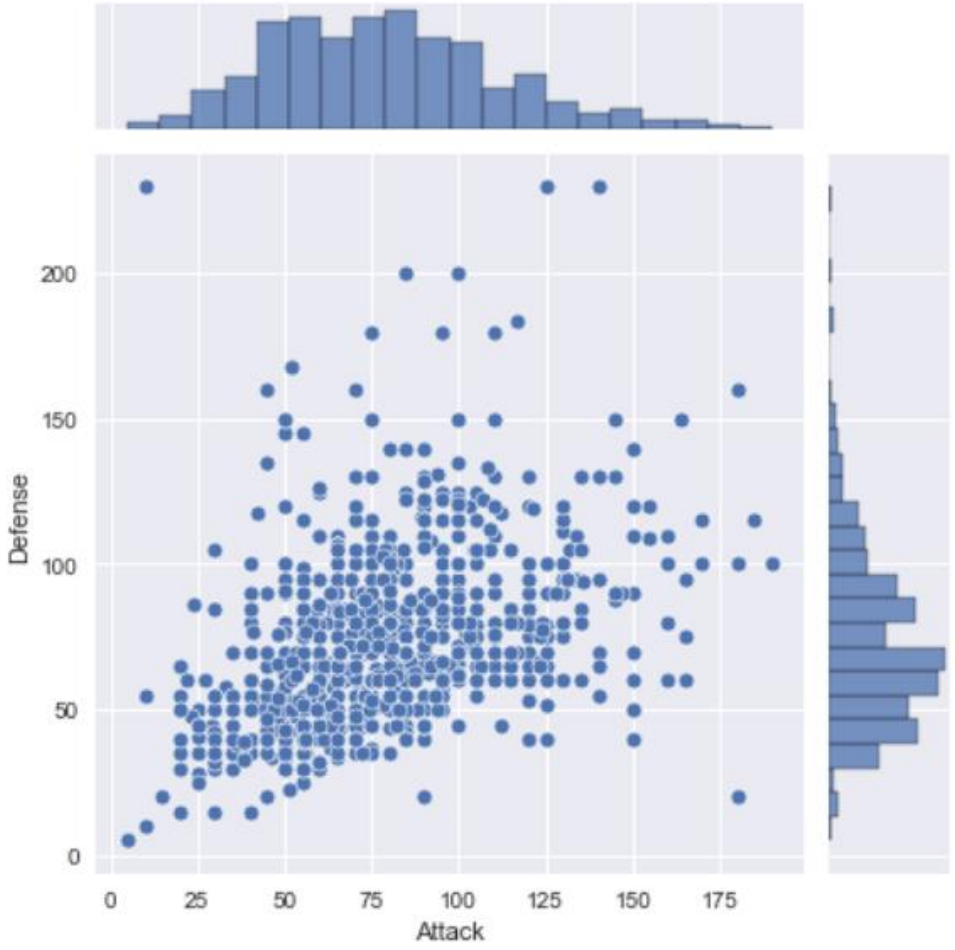



```
sns.histplot(pokemon_df['HP'],kde=True);
```

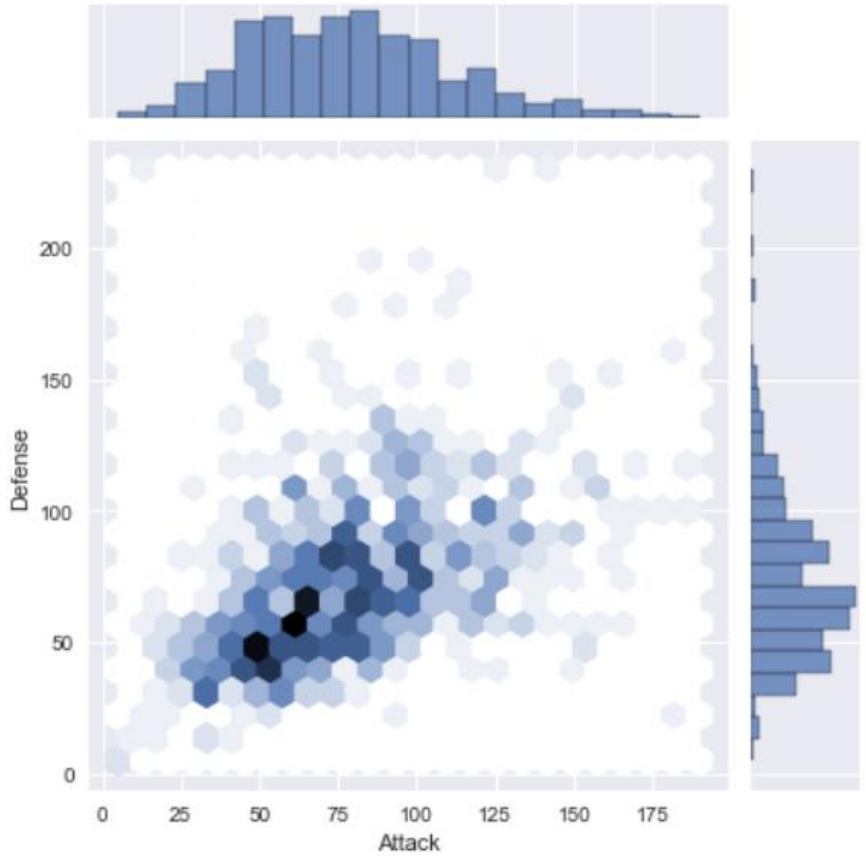


Join Plot

```
sns.jointplot(data=pokemon_df, x ="Attack", y = "Defense");
```




```
sns.jointplot(data=pokemon_df, x ="Attack", y = "Defense", kind="hex" );
```



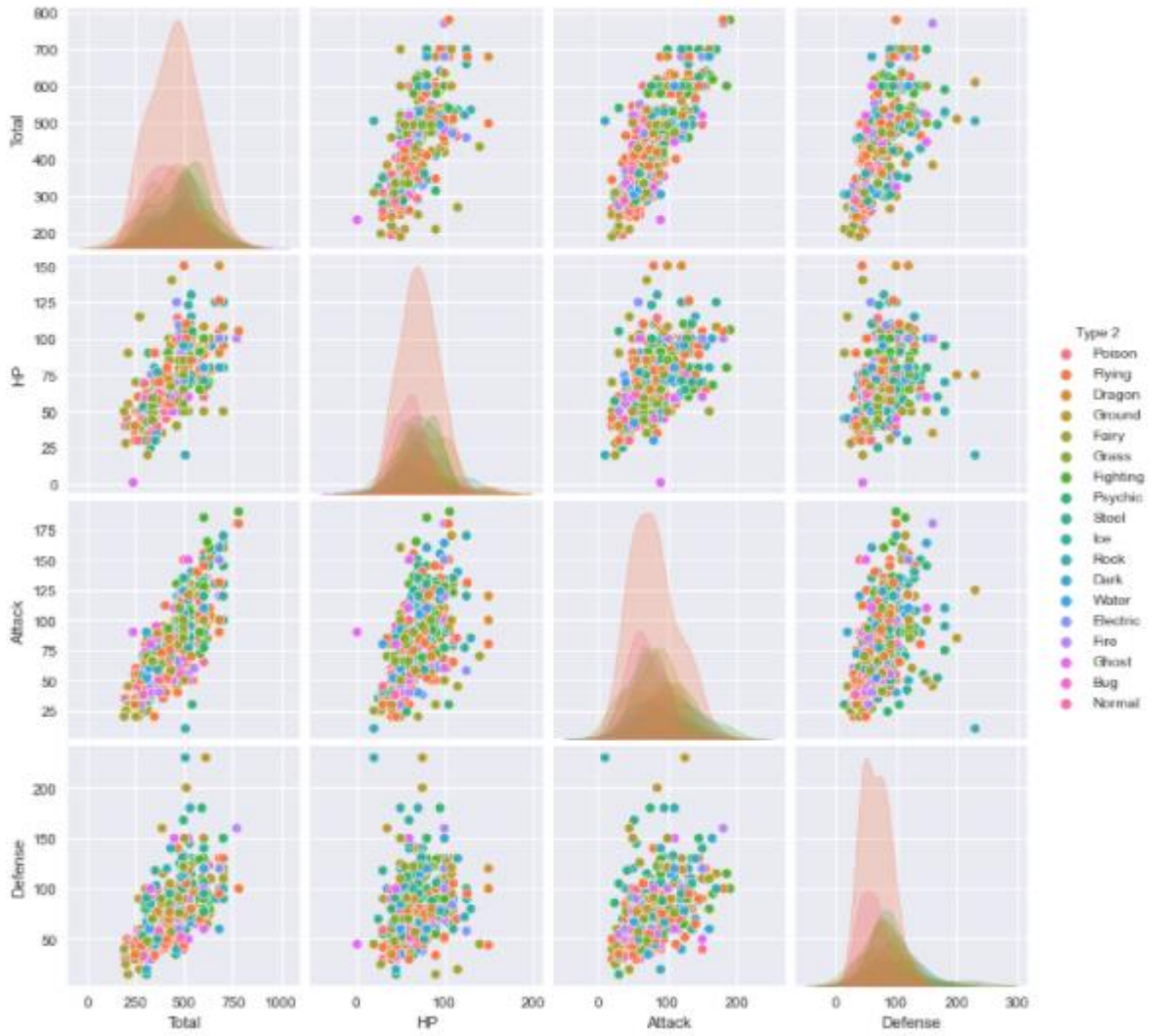
Pair plots

When you generalize joint plots to datasets of larger dimensions, you end up with pair plots. This is very useful for exploring correlations between multidimensional data, when you'd like to plot all pairs of values against each other.

```
df = pokemon_df[['Type 2','Total',
                 'HP','Attack','Defense']]
df.head()
```

	Type 2	Total	HP	Attack	Defense
0	Poison	318	45	49	49
1	Poison	405	60	62	63
2	Poison	525	80	82	83
3	Poison	625	80	100	123
4	NaN	309	39	52	43

```
sns.pairplot(df,hue="Type 2");
```



Practice

```
g = sns.pairplot(df1, hue ='Type 2');
g.map(plt.scatter);
```

```
g = sns.pairplot(df1, hue ='Type 2');
g.map(sns.displot);
```

```
g = sns.pairplot(df1, hue ='Type 2');
g.map(sns.jointplot);
```

Iris dataset

```
iris = sns.load_dataset("iris")
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Facet Plots

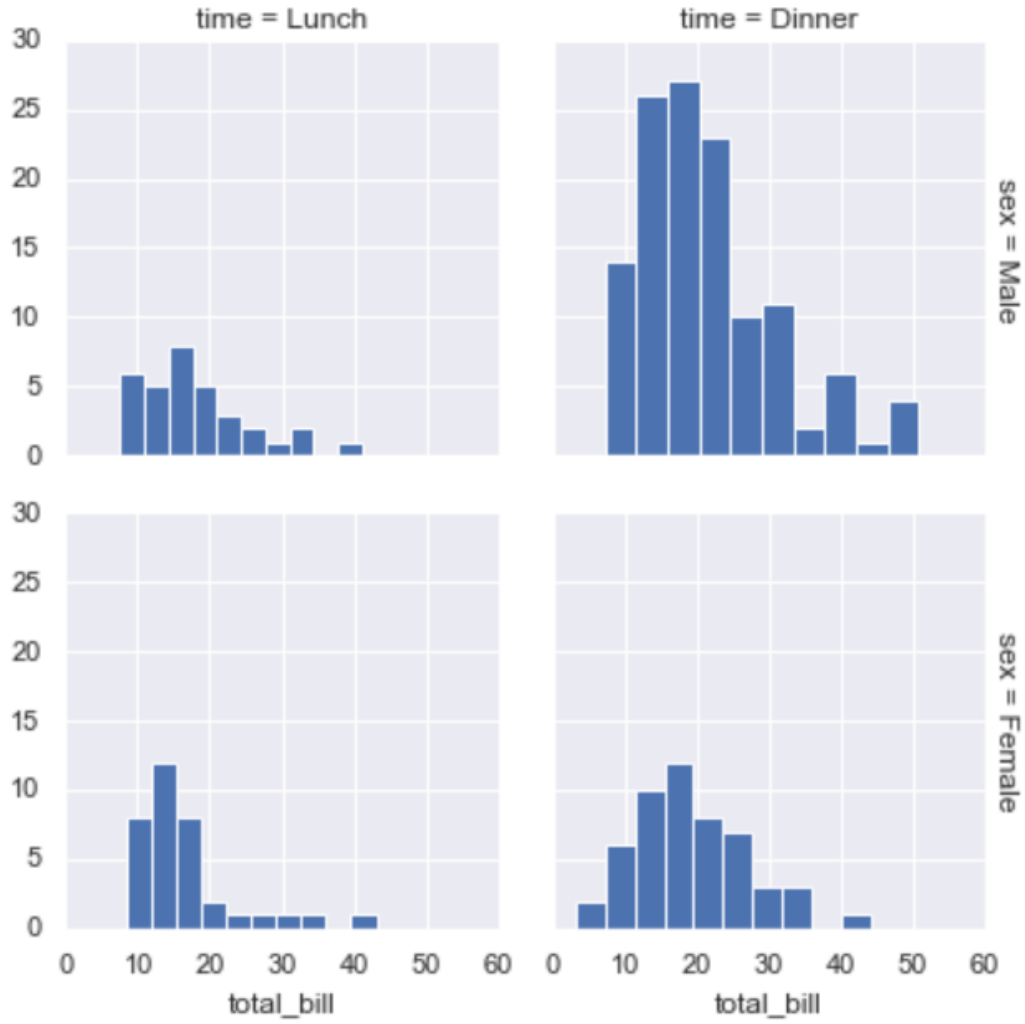
- **Faceted histograms**

Sometimes the best way to view data is via histograms of subsets. Seaborn's FacetGrid makes this extremely simple. We'll take a look at some data that shows the amount that restaurant staff receive in tips

```
tips = sns.load_dataset('tips')
tips.head()
```

	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

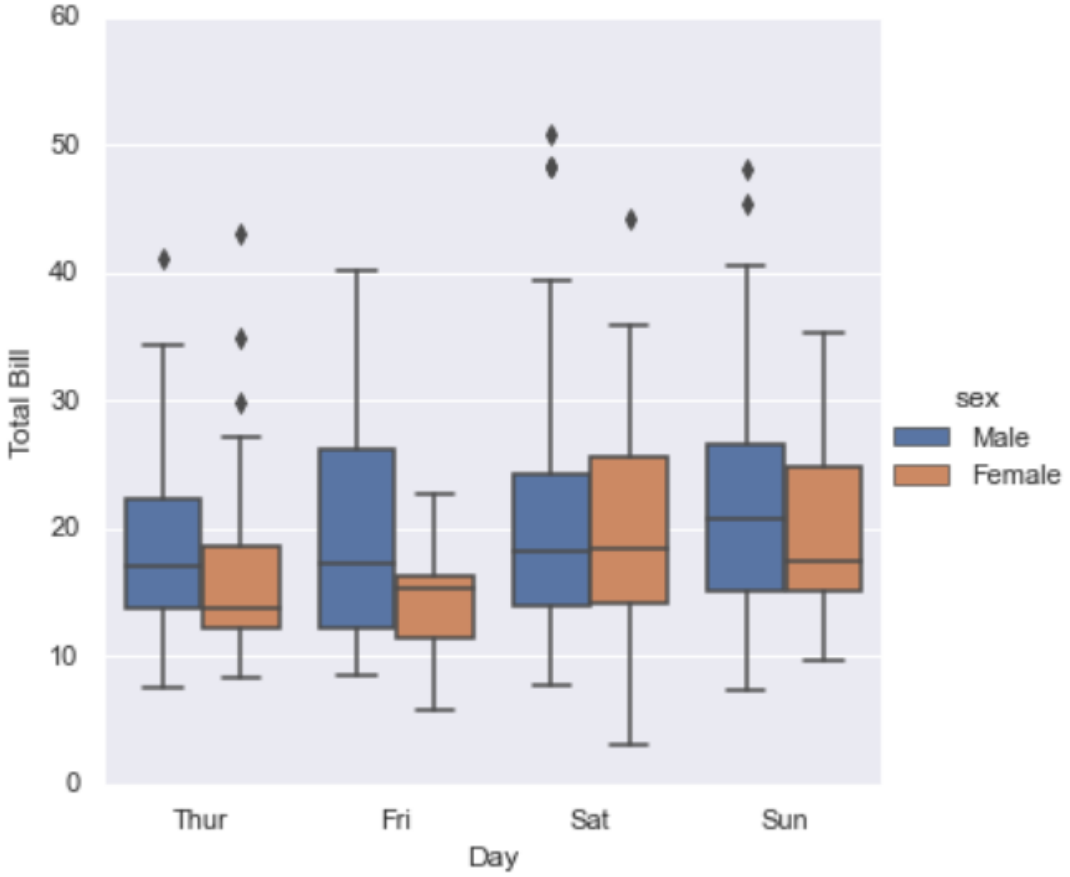
```
grid = sns.FacetGrid(tips, row="sex", col="time",
                     margin_titles=True)
grid.map(plt.hist, "total_bill");
```



Factor and Bar plots

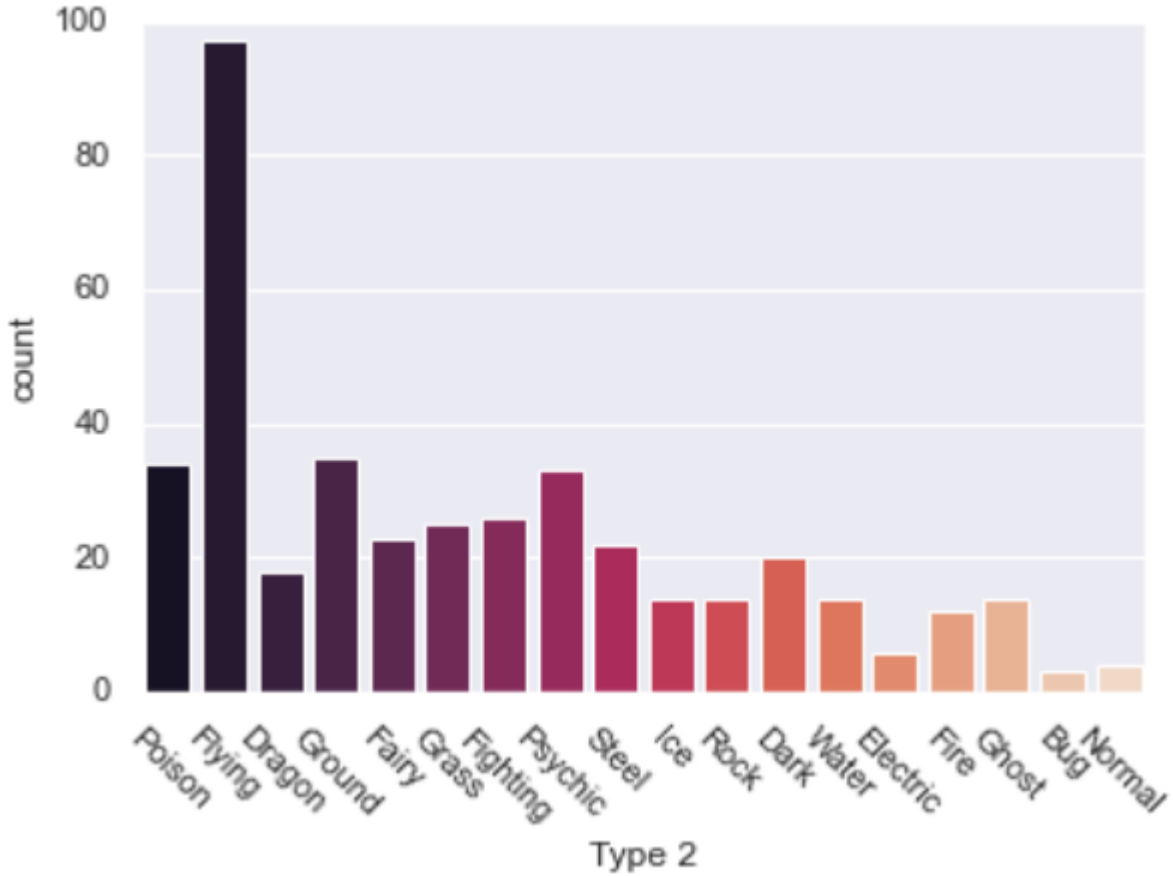
- **Factor plots**

```
g = sns.catplot(x = "day", y="total_bill",
                hue="sex", data=tips, kind="box")
g.set_axis_labels("Day", "Total Bill");
```



- **Bar plots**

```
sns.countplot(x='Type 2', data=df,palette="rocket")
plt.xticks(rotation=-45);
```



Heatmap

```
# Load the example flights dataset
flights_long = sns.load_dataset("flights")
flights = flights_long.pivot(index = "month",
                             columns="year",
                             values ="passengers")
flights.head()
```

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959
month											
Jan	112	115	145	171	196	204	242	284	315	340	360
Feb	118	126	150	180	196	188	233	277	301	318	342
Mar	132	141	178	193	236	235	267	317	356	362	406
Apr	129	135	163	181	235	227	269	313	348	348	396
May	121	125	172	183	229	234	270	318	355	363	420

```
# Draw a heatmap with the numeric values in each
f, ax = plt.subplots(figsize=(9, 6))
sns.heatmap(flights, annot=True, fmt="d",
            linewidths=1.5);
```

Practice: Visualation Correlation

```
diamonds_df = sns.load_dataset('diamonds')
diamonds_df.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
columm_drop = ['cut','clarity','color']
diamonds_new = diamonds_df.drop(columm_drop, axis=1)
diamonds_new.head()
```

```
diamon_corr = diamonds_new.corr()
diamon_corr
```