Matplotlib

What is Matplotlib?

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt
```

Plotting x and y points

The plot() function is used to draw points (markers) in a diagram.

By default, the plot() function draws a line from point to point.

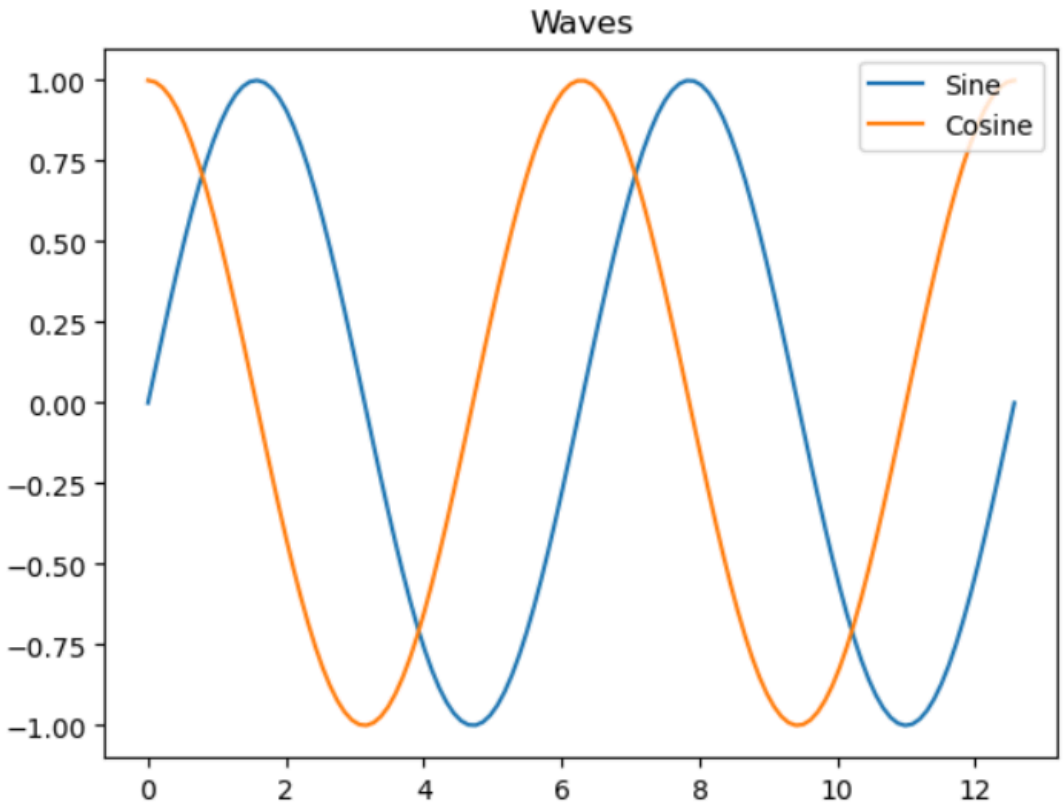
The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the x-axis.

Parameter 2 is an array containing the points on the y-axis.

Visualizing Sine and Cosine Functions

```
x = np.linspace(0, 4*np.pi, 120)
sin_y = np.sin(x)
cos_y = np.cos(x)

plt.plot(x,sin_y, label="Sine")
plt.plot(x,cos_y, label="Cosine")
plt.title("Waves")
plt.legend(loc='upper right')
plt.show()
```



1. Legend is used to add labels to the graph given in the plot function. The loc argument takes an integer or string that specifies the location of the legend on the plot. different values for loc, such as lower right, center, upper right, lower left, etc.
2. title function adds a title to the plot.

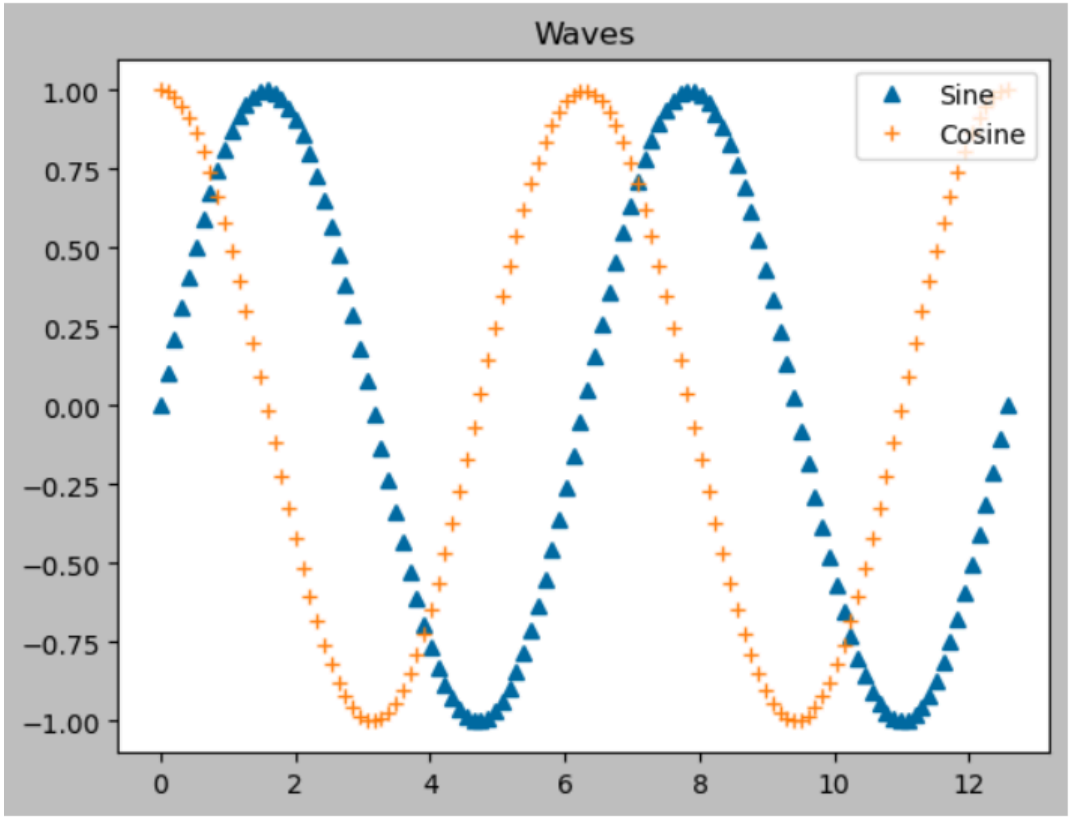
Setting Styles to the Graph

One way to use styles is use plt.style.use('style_name').

```
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks', 'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

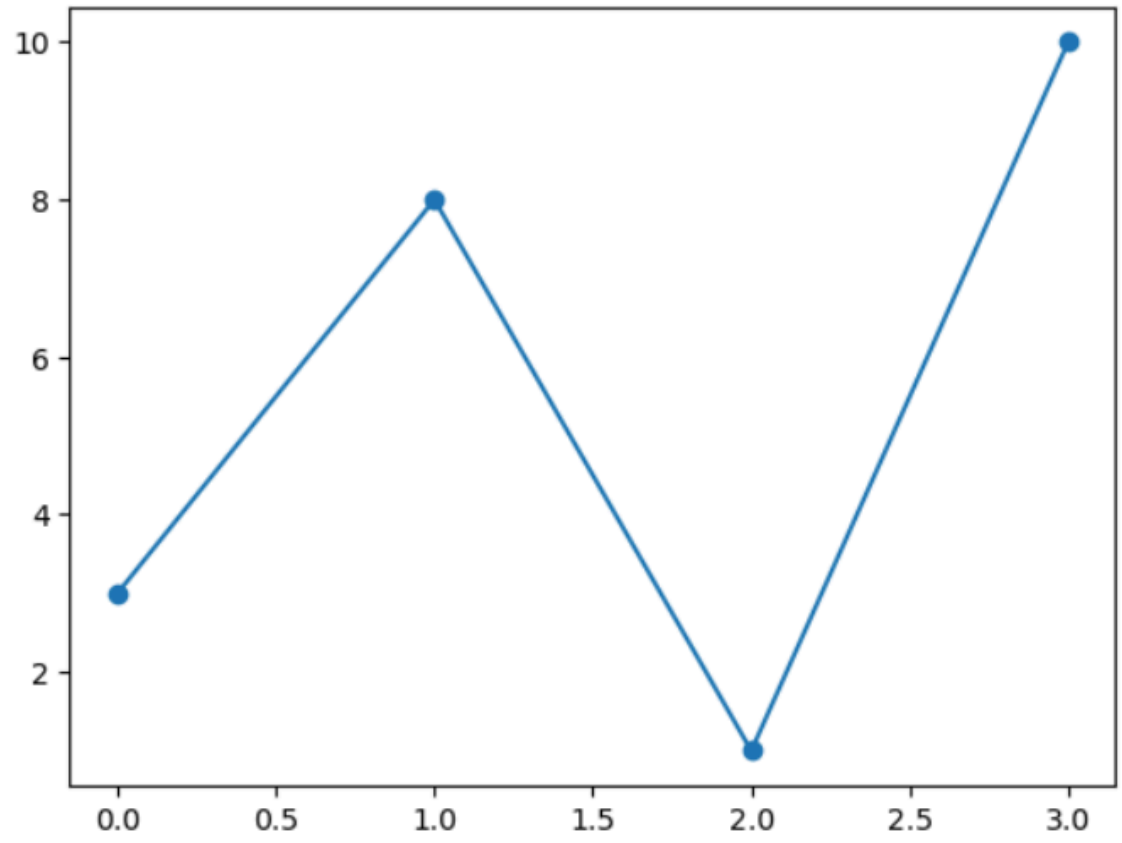
```
plt.style.use("tableau-colorblind10")

plt.plot(x, sin_y, "^", label = "Sine")
plt.plot(x, cos_y, "+", label = "Cosine")

plt.title("Waves")
plt.legend(loc = "upper right")
plt.show()
```



Markers

You can use the keyword argument marker to emphasize each point with a specified marker.

```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```



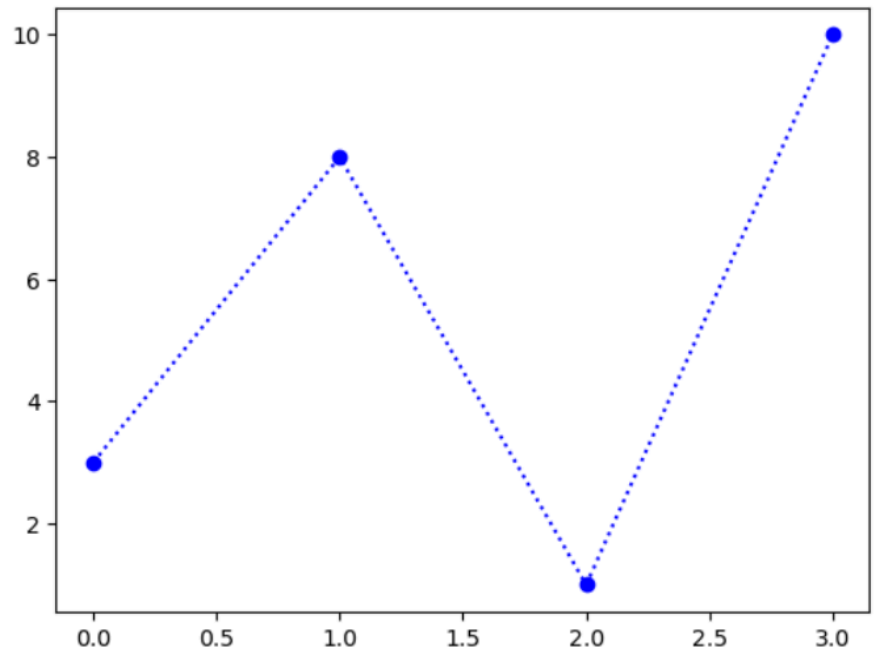
Format Strings fmt

You can also use the shortcut *string notation* parameter to specify the marker.

This parameter is also called fmt, and is written with this syntax:

Marker|line|color

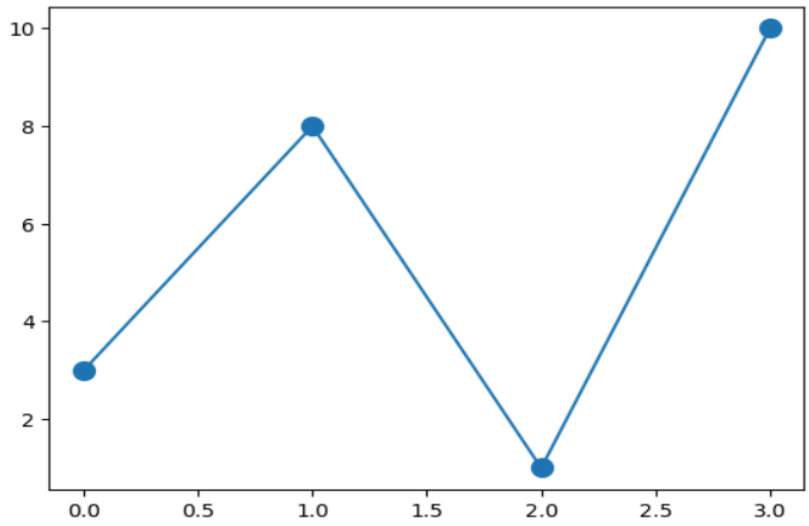
```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:b')
plt.show()
```



Marker Size

You can use the keyword argument markersize or the shorter version, ms to set the size of the markers.

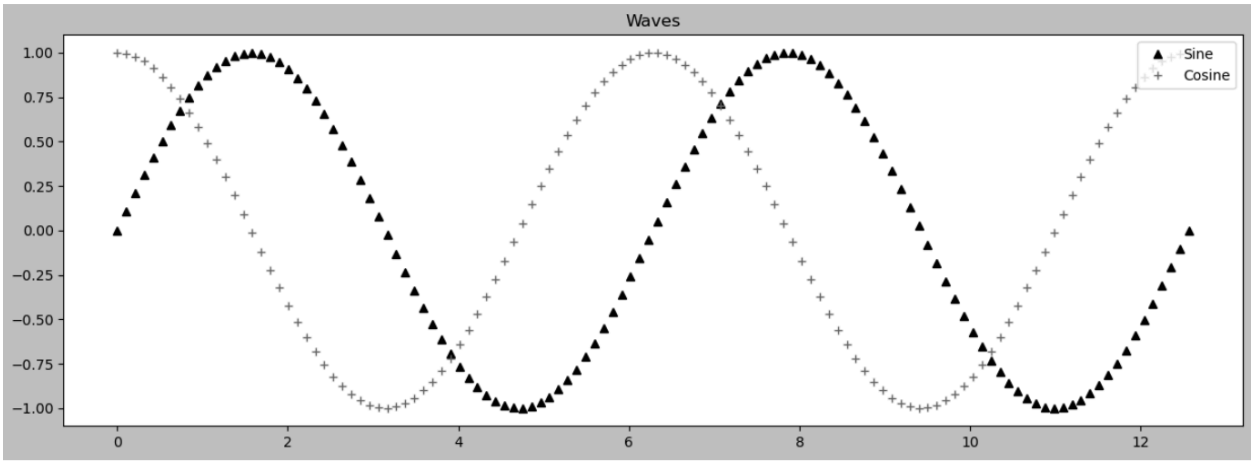
```
plt.plot(ypoints, marker = 'o', ms = 10)
plt.show()
```



Size of Plot

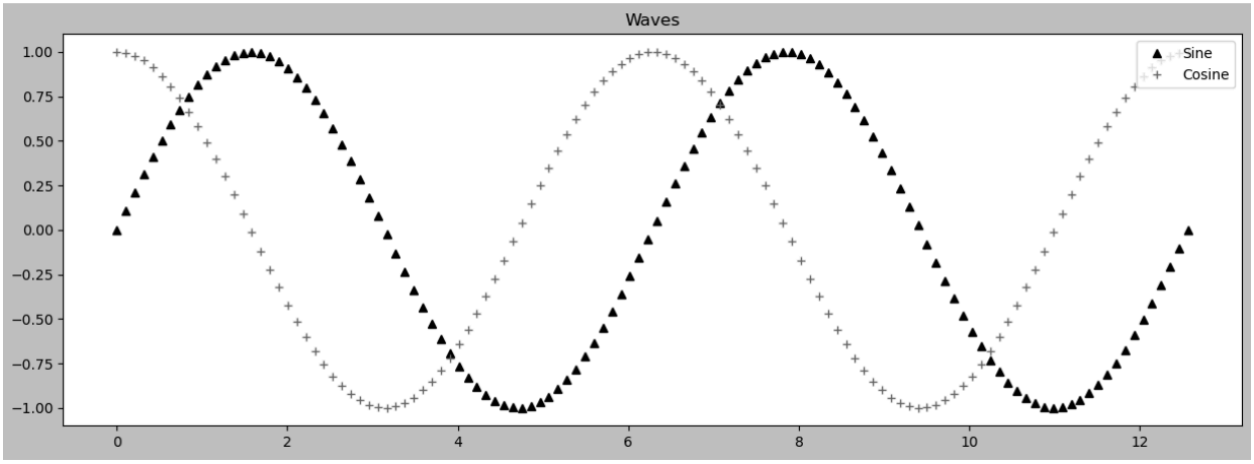
plt.figure(figsize=(length,height))

```
plt.figure(figsize = (15,5))

plt.style.use("grayscale")

plt.plot(x, sin_y, "^", label = "Sine")
plt.plot(x, cos_y, "+", label = "Cosine")

plt.title("Waves")
plt.legend(loc = "upper right")
plt.show()
```



Save plot in machine

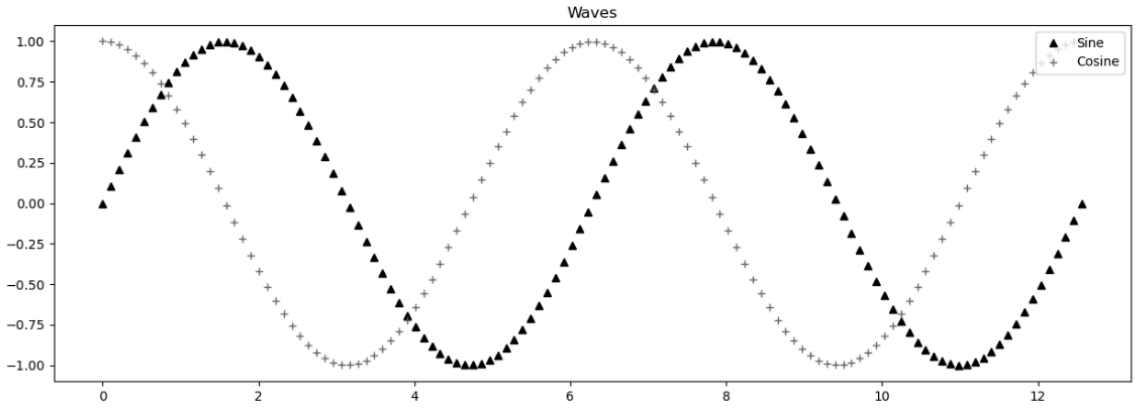
```
plt.figure(figsize = (15,5))

plt.style.use("grayscale")

plt.plot(x, sin_y, "^", label = "Sine")
plt.plot(x, cos_y, "+", label = "Cosine")

plt.title("Waves")
plt.legend(loc = "upper right")

plt.savefig("Waves")

plt.show()
```



Display Saved Image

```
from IPython.display import Image
```

```
Image("Waves.png")
```



Different Types of Plots

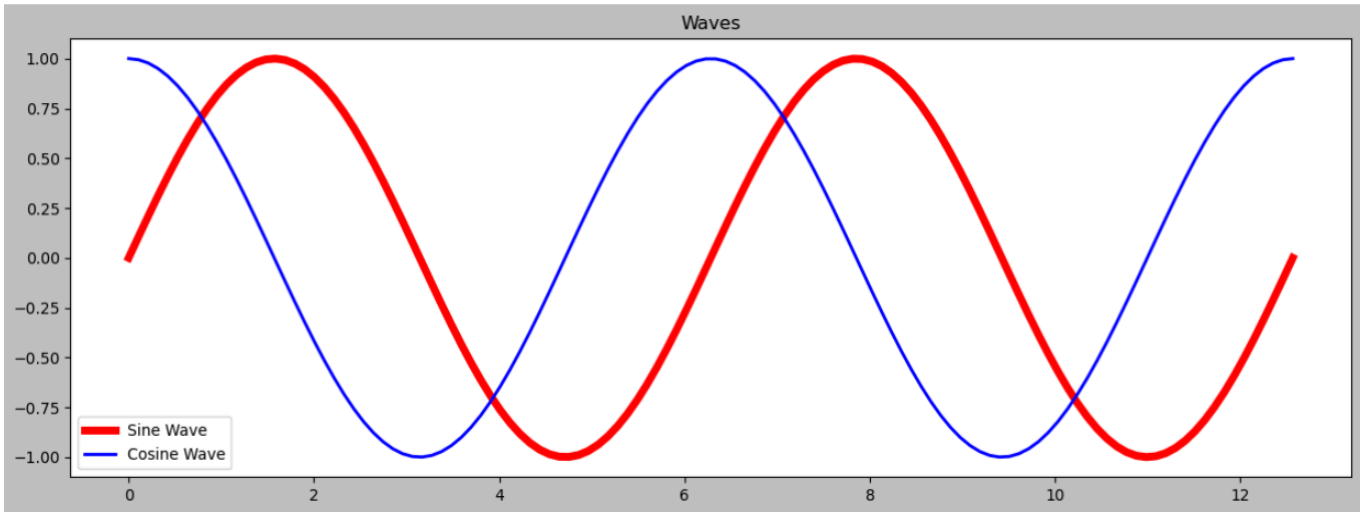
Line Plot

A line plot is a type of plot used to visualize the relationship between two continuous variables. It is a plot that displays data points as connected line segments. Line plots are commonly used to visualize trends and patterns in data, especially when the data is continuous and has a large number of observations.

In a line plot, the x-axis represents one variable and the y-axis represents another variable. The data points are connected by lines to create a plot that shows how the values of the two variables change with respect to each other. Line plots are often used to visualize time-series data, where the x-axis represents time and the y-axis represents a measurement.

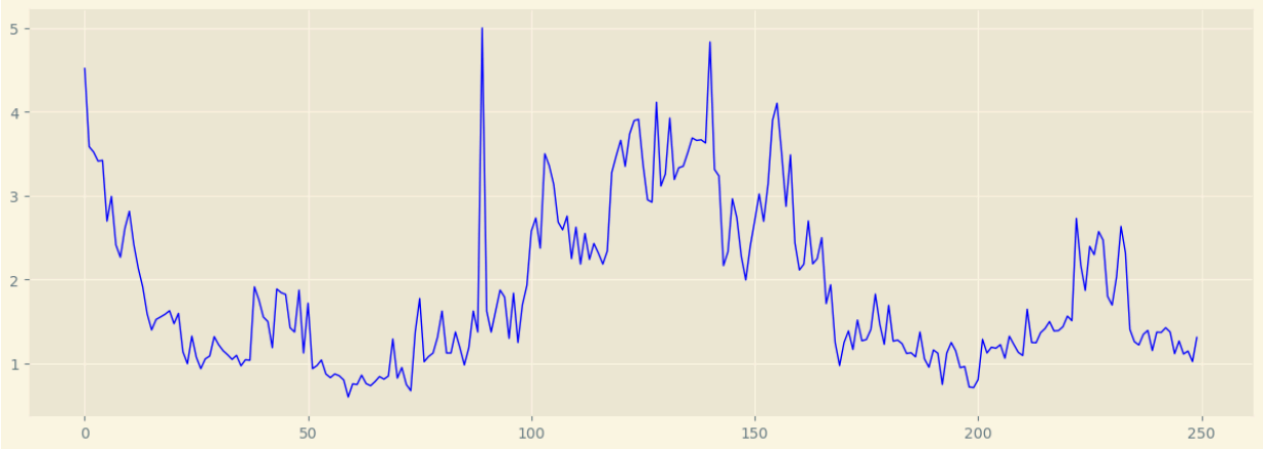
In Matplotlib, the plot function is used to create a line plot. You can specify the x and y values of the data points, along with other optional arguments such as color, line style, and markers . You can also add labels, titles, and legends to your line plot to provide context and clarity.

```
x = np.linspace(0, 4*np.pi, 120)
sin_y = np.sin(x)
cos_y = np.cos(x)

plt.figure(figsize = (15,5))

plt.plot(x, sin_y, color="Red", linewidth = 5, label = "Sine Wave")
plt.plot(x, cos_y, color="blue", linewidth = 2, label = "Cosine Wave")

plt.title("Waves")
plt.legend()

plt.show()
```



```
dataset = fetch_california_housing()
target = dataset['target']
```

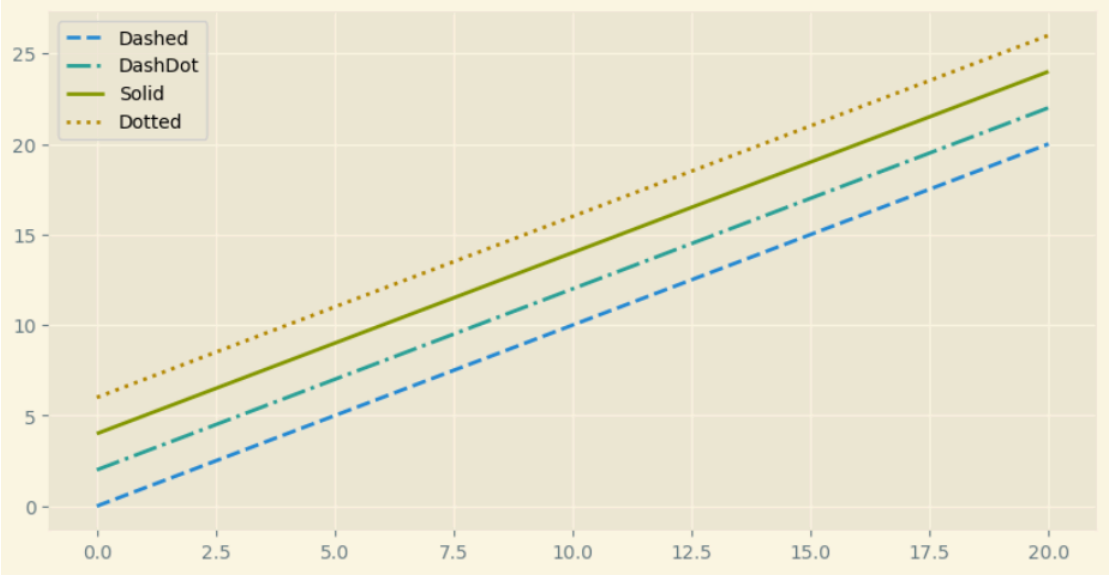
```
plt.figure(figsize = (15,5))
plt.style.use('Solarize_Light2')

plt.plot(target[:250], color = 'blue', linewidth = 1)

plt.show()
```



Linestyle command

It has 4 main styles solid, dashed, dashdot and dotted

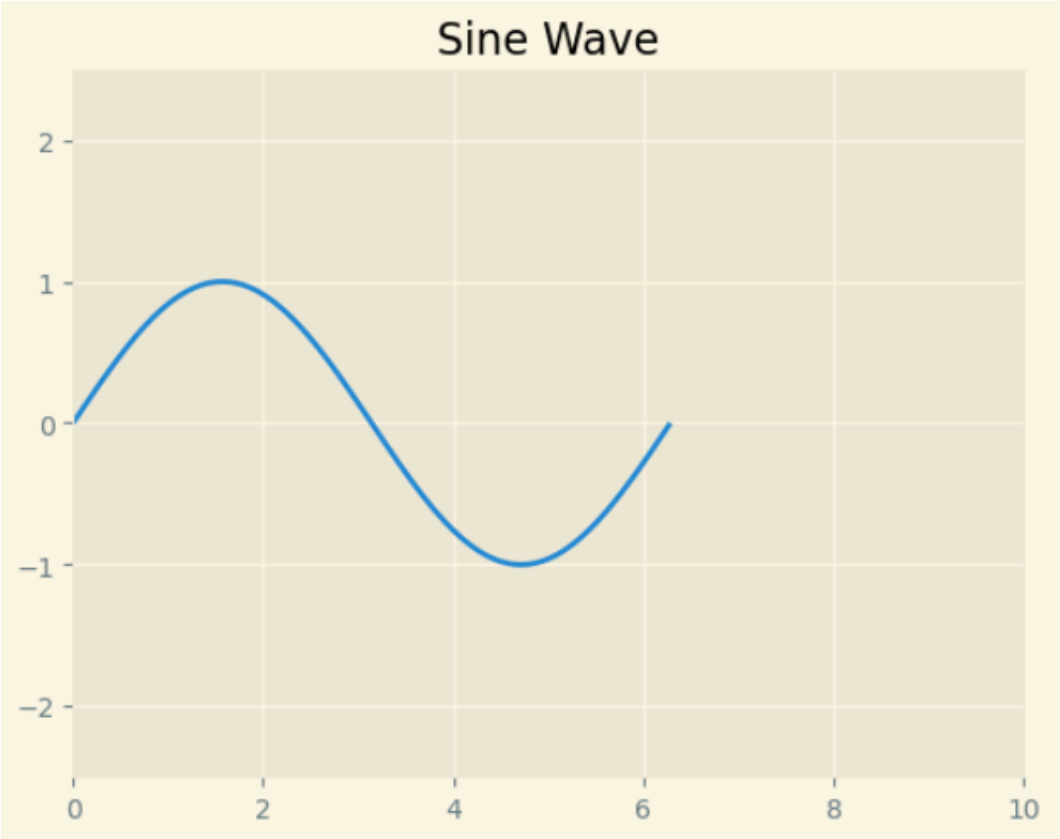
```
x = np.linspace(0, 20, 21, dtype = 'int')

plt.figure(figsize = (10,5))
plt.plot(x, x + 0, linestyle = 'dashed', label = 'Dashed')
plt.plot(x, x + 2, linestyle = 'dashdot', label = 'DashDot')
plt.plot(x, x + 4, linestyle = 'solid', label = 'Solid')
plt.plot(x, x + 6, linestyle = 'dotted', label = 'Dotted')

plt.legend()
plt.show()
```



Adjusting the Plot Axes

Can be done using plt.xlim for x axis and plt.ylim for y axis

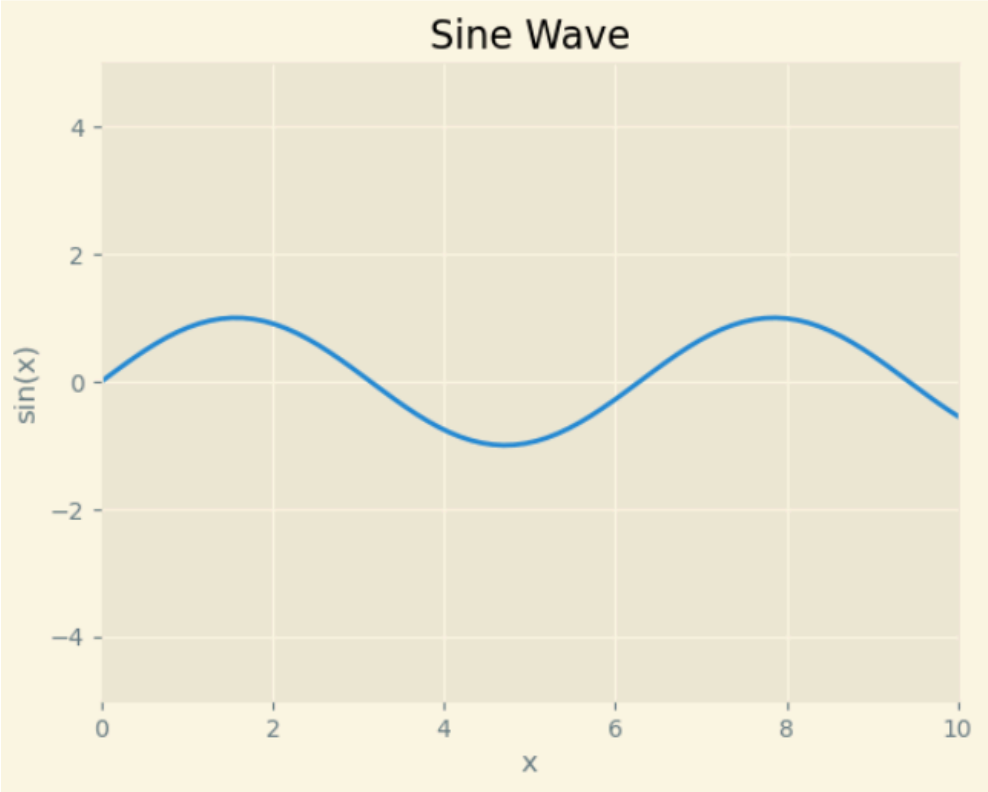
```
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

plt.xlim([0,10])
plt.ylim([-2.5,2.5])

plt.plot(x,y)
plt.title('Sine Wave')
plt.show()
```



.set to combine all of the commands

```
x = np.linspace(0, 4 * np.pi, 100)

fig = plt.axes()
fig.plot(x, np.sin(x))
fig.set(xlim = (0,10), ylim = (-5,5), xlabel = 'x', ylabel = "sin(x)", title = "Sine Wave",)
fig.plot()
```

```
[]
```

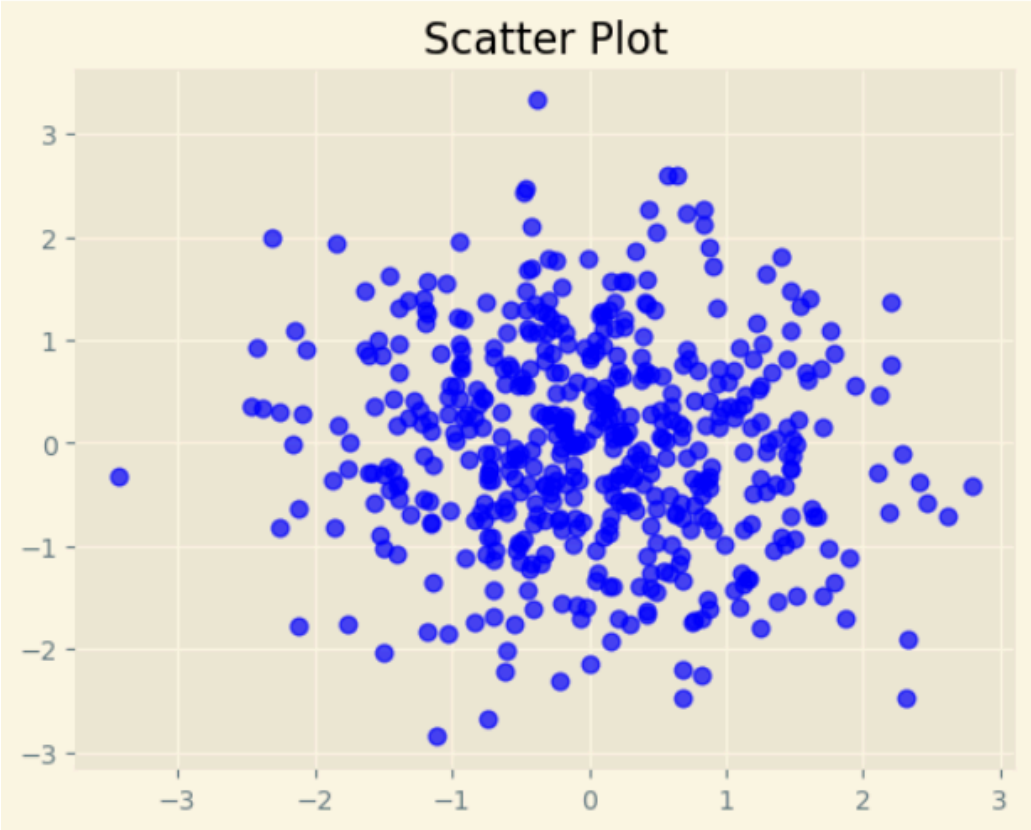


Scatter Plots

A scatter plot is a type of data visualization in which individual data points are represented as points in a two-dimensional plane. The x-coordinate of each data point represents one feature of the data, and the y-coordinate represents another feature. Scatter plots are commonly used to visualize the relationship between two variables and to identify patterns or outliers in the data.

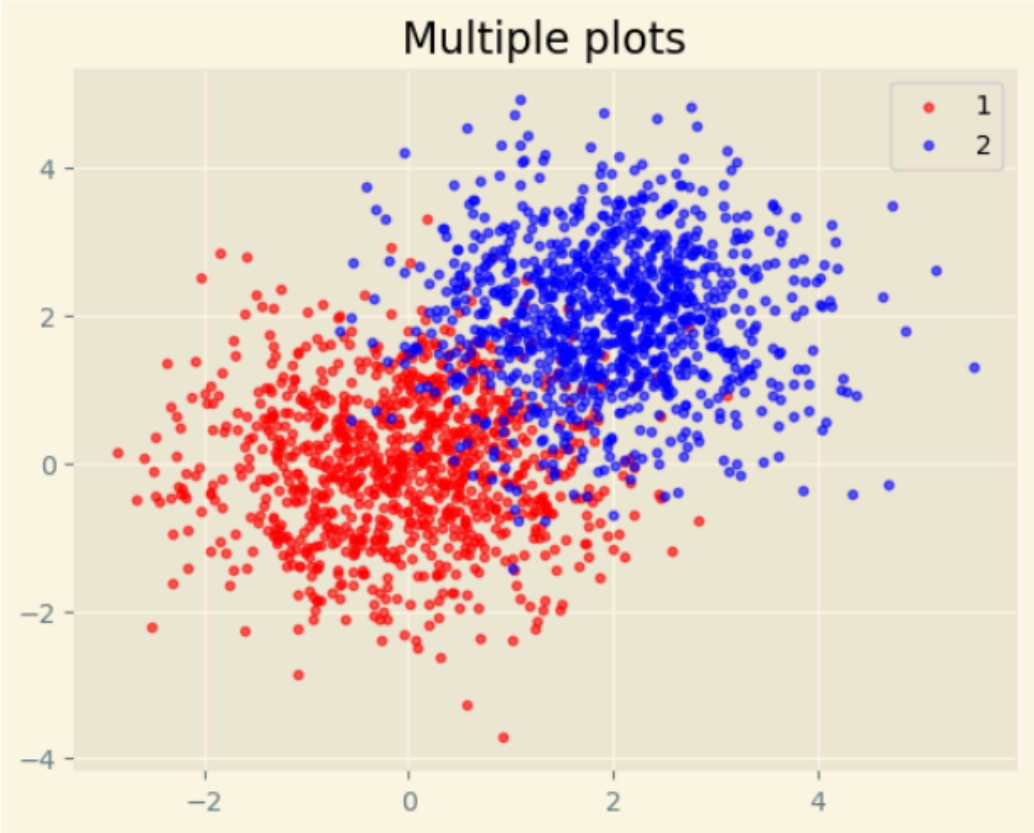
In Matplotlib, scatter plots can be created using the scatter function. The scatter function accepts two arrays of values, representing the x-coordinates and y-coordinates of the data points. You can also specify additional arguments to customize the appearance of the scatter plot, such as the size and color of the markers, the transparency of the markers, and the marker shape.

```
n = 500
x = np.random.randn(n)
y = np.random.randn(n)

plt.scatter(x,y, s = 40, c = 'blue', alpha = 0.7)
plt.title('Scatter Plot')
plt.show()
```



Multiple Scatter Plots

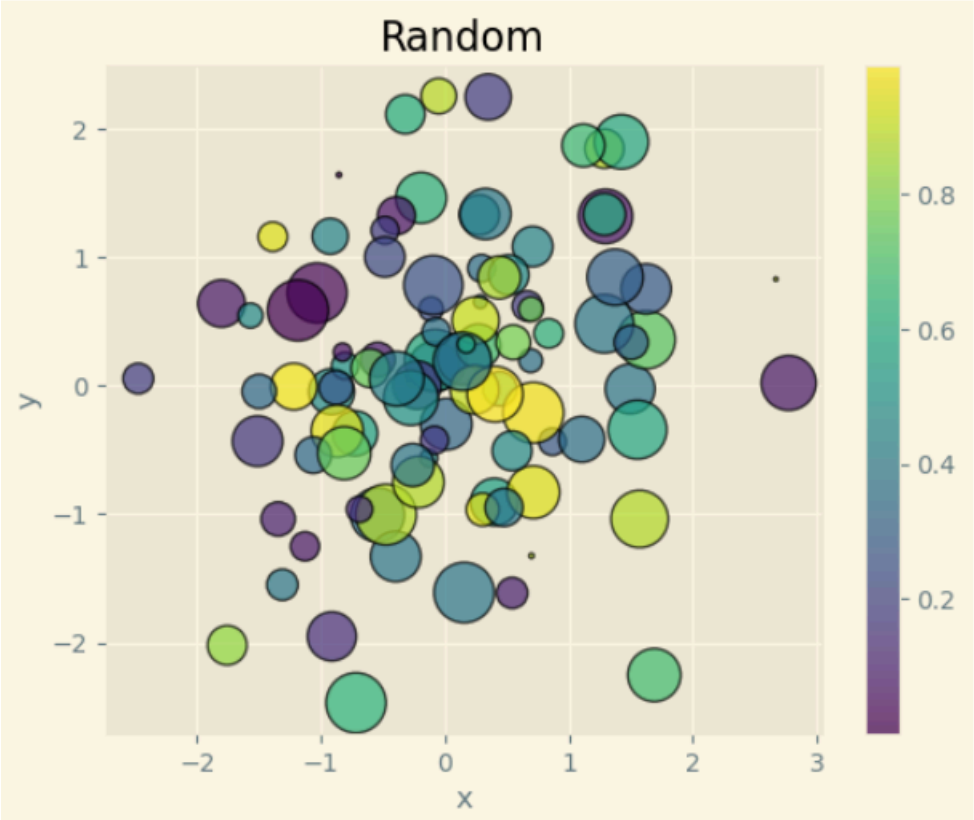
```
n = 1000
x1 = np.random.randn(n)
y1 = np.random.randn(n)

x2 = np.random.randn(n) + 2
y2 = np.random.randn(n) + 2

plt.scatter(x1,y1, c = 'red', s = 10, alpha = .6, label = '1')
plt.scatter(x2,y2, c = 'blue', s = 10, alpha = .6, label = '2')
plt.title('Multiple plots')
plt.legend()
plt.show()
```



The primary difference between plt.scatter and plt.plot is that plt.scatter gives us the freedom to control the properties of each individual point (size,face_color,edge_color, transparency level etc)

- Alpha: Transparency of the markers in the scatter plot.
- Colorbar: A bar showing the colors used in the plot and the corresponding values.
- Colormap: A color mapping used to convert scalar data to colors.
- Edgecolors: The color of the edges of the markers in a scatter plot.
- Sizes: The size of the markers in a scatter plot.

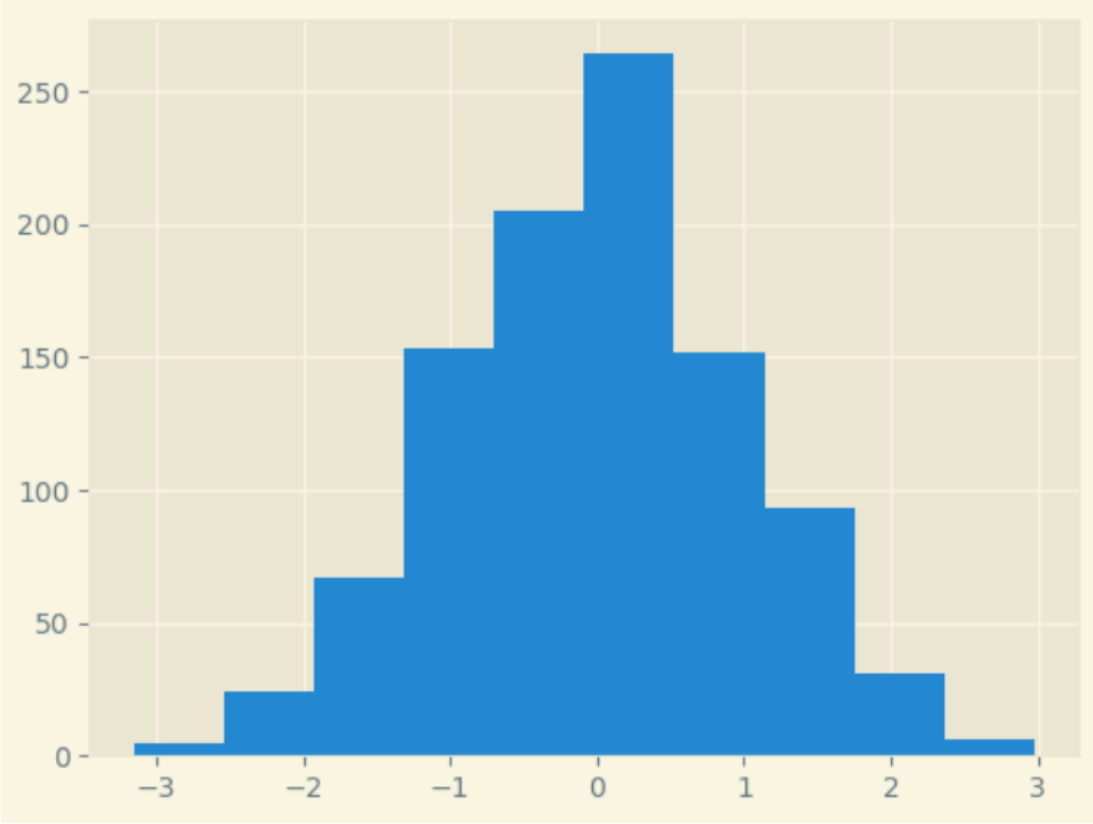
```
n = 100
x = np.random.randn(n)
y = np.random.randn(n)

colors = np.random.rand(n)
sizes = np.random.rand(n) * 600

plt.scatter(x,y, c = colors, s = sizes, cmap = 'viridis', alpha = .7, edgecolor = 'black')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Random')
plt.colorbar()
plt.show()
```



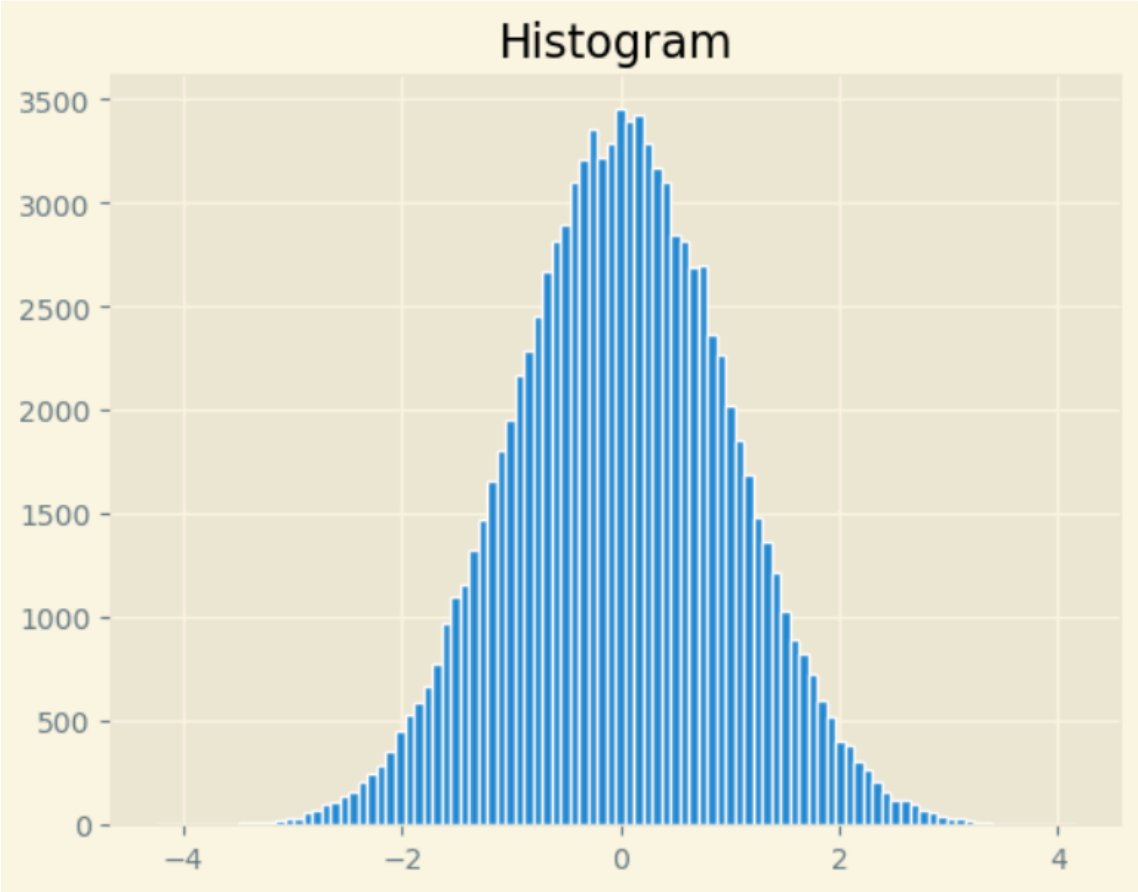
Histograms and Binnings

A histogram is a plot that displays the distribution of a set of continuous or discrete data. The data is divided into a set of intervals or bins, and the height of each bar in the plot represents the number of data points that fall within that bin. Binnings refers to the process of dividing the data into these bins. The choice of bin size can greatly affect the appearance of the histogram, so it's important to choose a bin size that accurately represents the underlying distribution of the data.

```
data = np.random.randn(1000)
plt.hist(data)
plt.show()
```

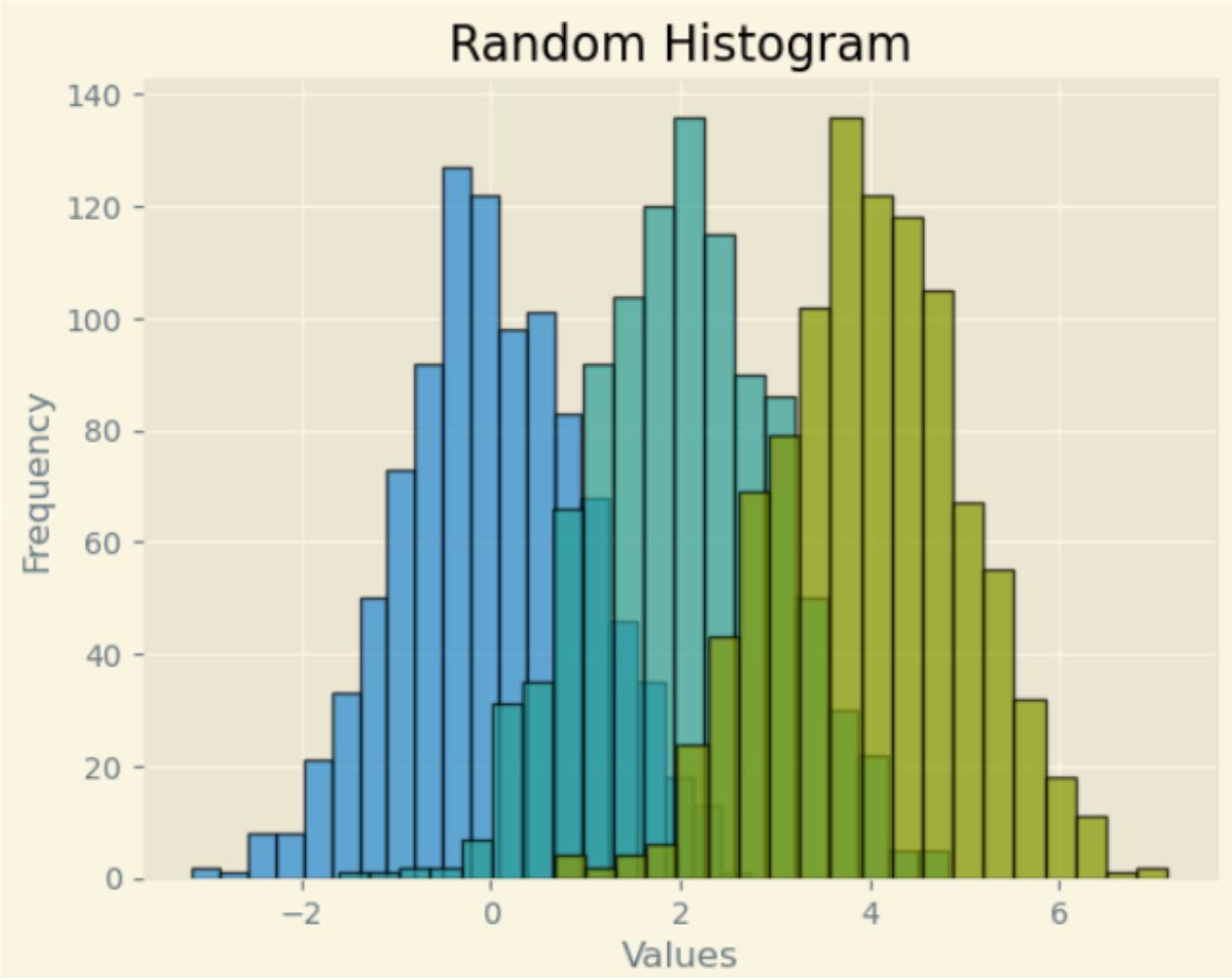


```
data = np.random.randn(100000)
plt.hist(data, edgecolor = 'white', alpha = 1, bins = 100)
plt.title("Histogram")
plt.show()
```



Multiple histograms

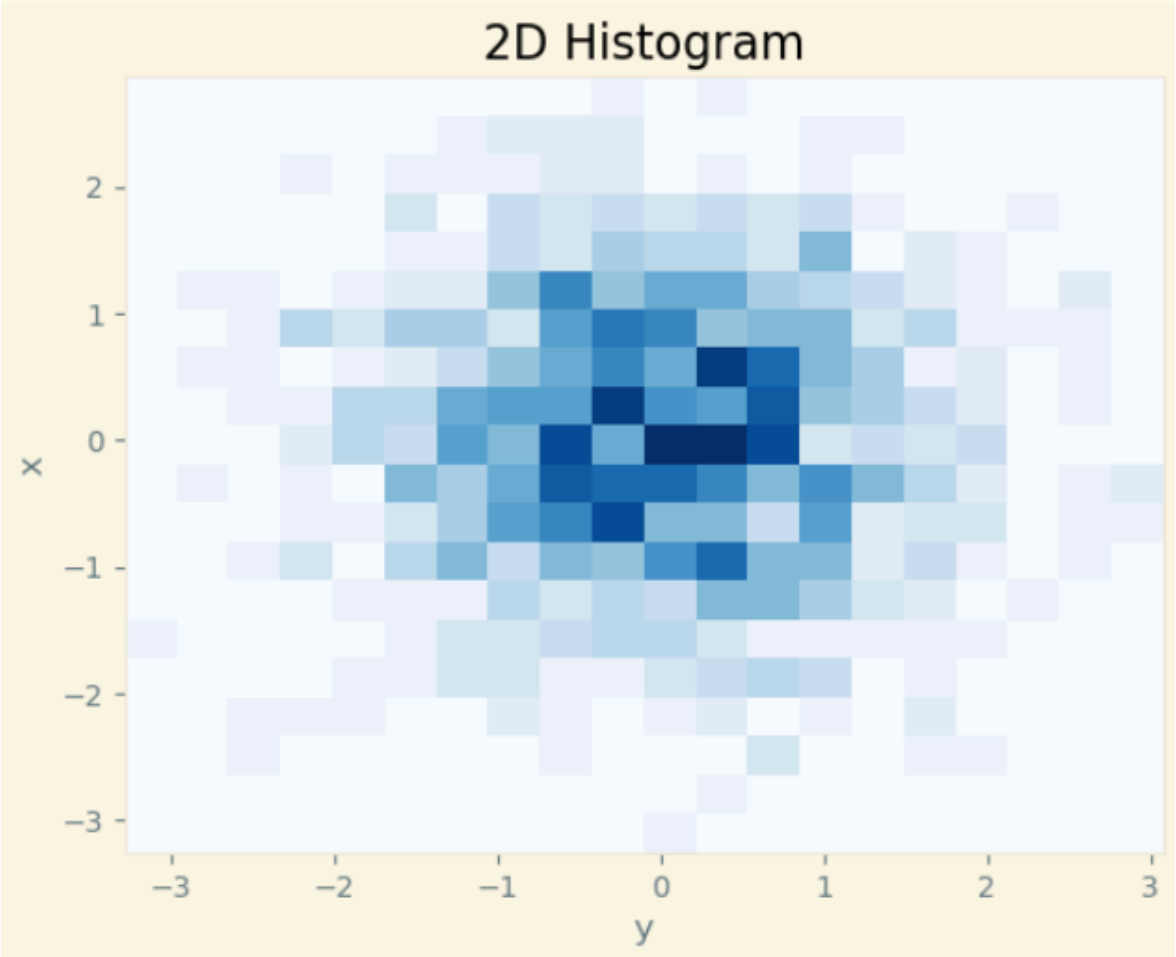
```
x1 = np.random.randn(1000)
x2 = np.random.randn(1000) + 2
x3 = np.random.randn(1000) + 4

plt.hist(x1, bins = 20,  alpha = .7, edgecolor = 'black')
plt.hist(x2, bins = 20,  alpha = .7, edgecolor = 'black')
plt.hist(x3, bins = 20, alpha = .7, edgecolor = 'black')

plt.title('Random Histogram')
plt.ylabel('Frequency')
plt.xlabel('Values')

plt.show()
```



2D Histograms

plt.hist2d is a function in matplotlib that generates a 2D histogram. A 2D histogram is a representation of the distribution of data over two variables. In contrast to a regular histogram which shows the distribution of a single variable, a 2D histogram shows the relationship between two variables. The function creates a heatmap that shows the density of data points in a 2D space.

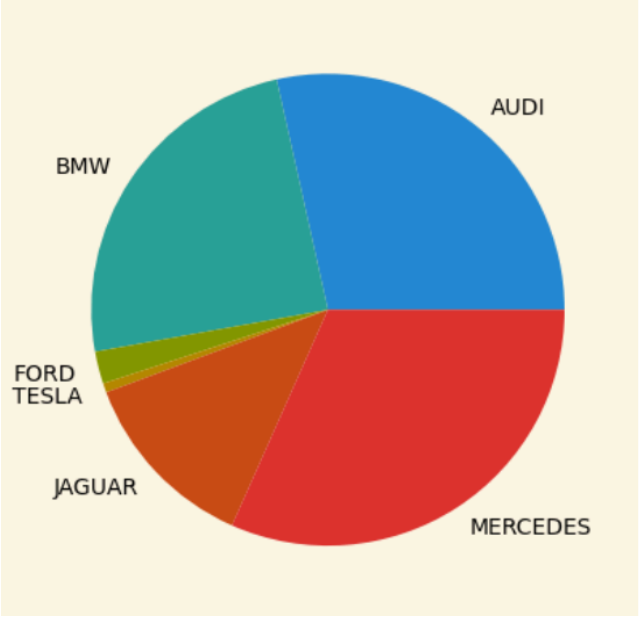
```
x = np.random.randn(1000)
y = np.random.randn(1000)

plt.hist2d(x,y, bins = 20 , cmap = 'Blues')

plt.title('2D Histogram')
plt.ylabel('x')
plt.xlabel('y')

plt.show()
```



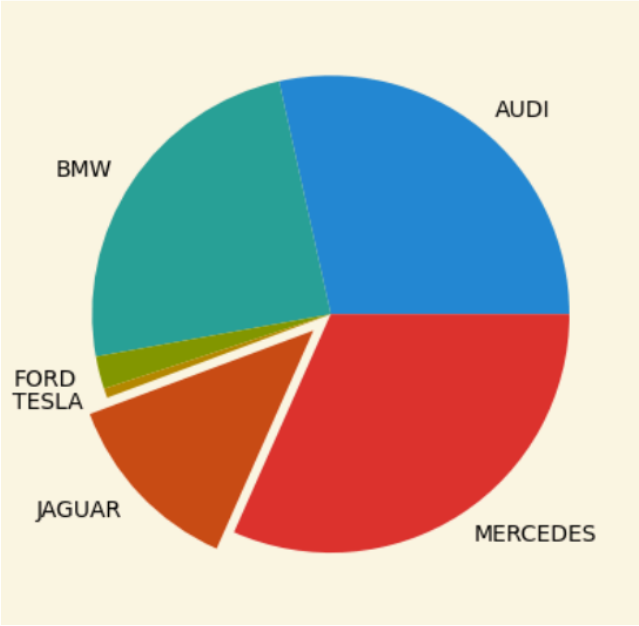
Pie Charts

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportion.

```
cars = ['AUDI','BMW','FORD','TESLA','JAGUAR','MERCEDES']
data = [ 450  , 385 , 35   ,   10  ,  200   , 500      ]

plt.pie(data, labels = cars)
plt.show()
```

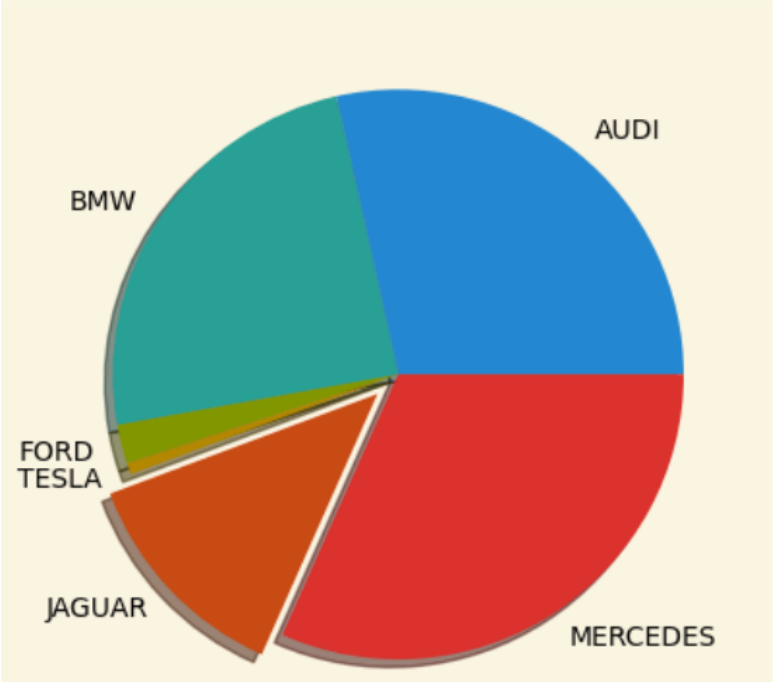



```
explodes = [0,0,0,0,0.1,0]
plt.pie(data, labels = cars, explode = explodes)
plt.show()
```



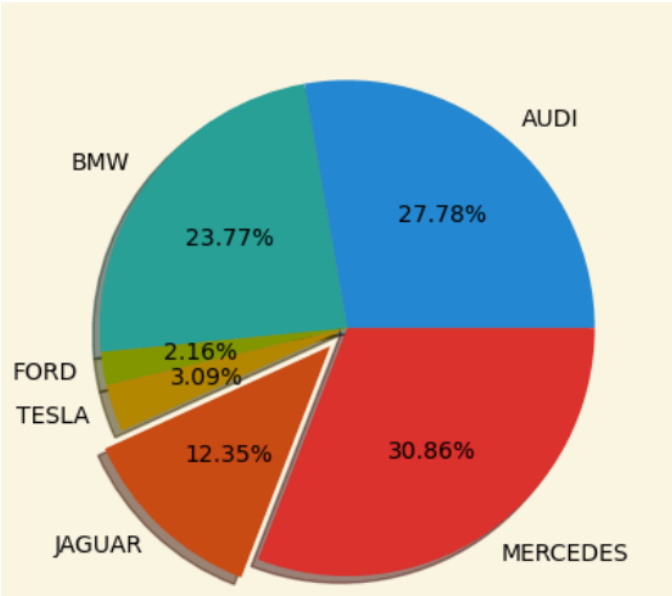
Adding shadows

```
plt.pie(data, labels = cars, explode = explodes, shadow = True)
plt.show()
```



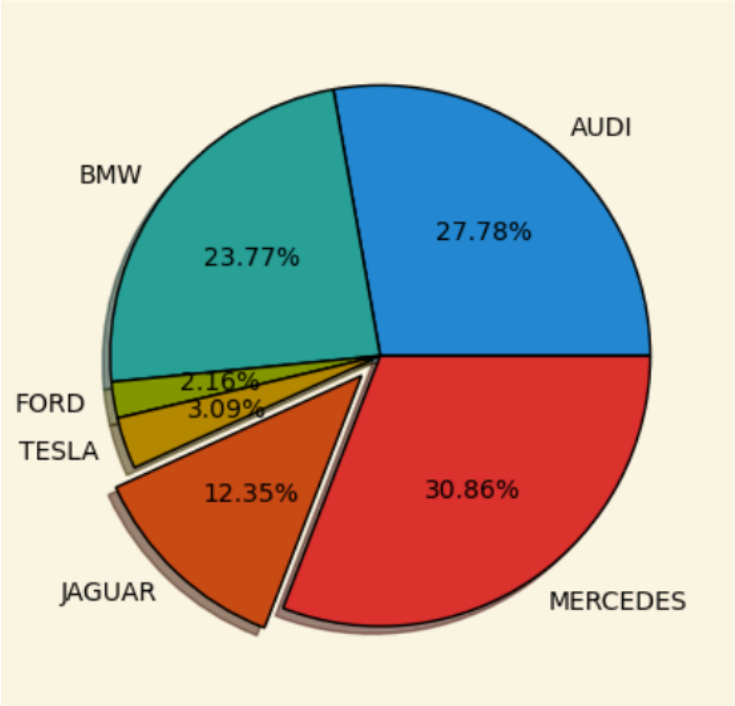
Adding Percentage

```
cars = ['AUDI','BMW','FORD','TESLA','JAGUAR','MERCEDES']
data = [ 450  , 385 , 35   ,   50  ,  200   , 500      ]
explode = [0, 0, 0 , 0, .1, 0]

plt.pie(data, labels = cars, explode = explode, shadow = True, autopct = '%2.2f%%')
plt.show()
```



Adding Wedget

```
plt.pie(data, labels = cars, explode = explode, shadow = True, autopct = '%2.2f%%',
        wedgeprops = {'edgecolor' : 'black', 'linewidth' : 1, 'antialiased' : True})
plt.show()
```



Subplots

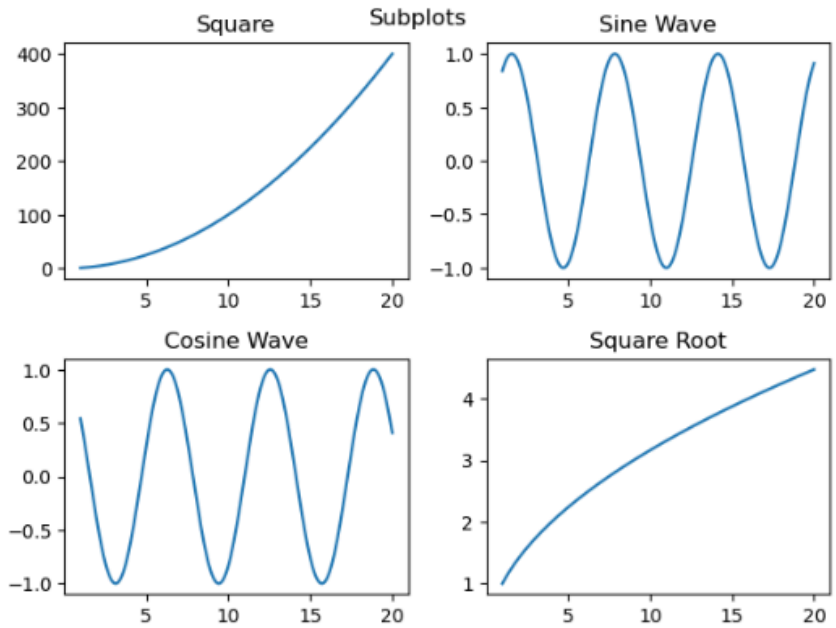
The subplot() Function

The subplot() function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the first and second argument.

The third argument represents the index of the current plot.

```
x = np.linspace(1,20,200)

y = x ** 2
plt.subplot(2, 2, 1)
plt.plot(x, y)
plt.title('Square')

y = np.sin(x)
plt.subplot(2, 2, 2)
plt.plot(x, y)
plt.title('Sine Wave')

y = np.cos(x)
plt.subplot(2, 2, 3)
plt.plot(x, y)
plt.title('Cosine Wave')

y = x ** 0.5
plt.subplot(2, 2, 4)
plt.plot(x, y)
plt.title('Square Root')

plt.tight_layout()
plt.suptitle('Subplots')
plt.show()
```



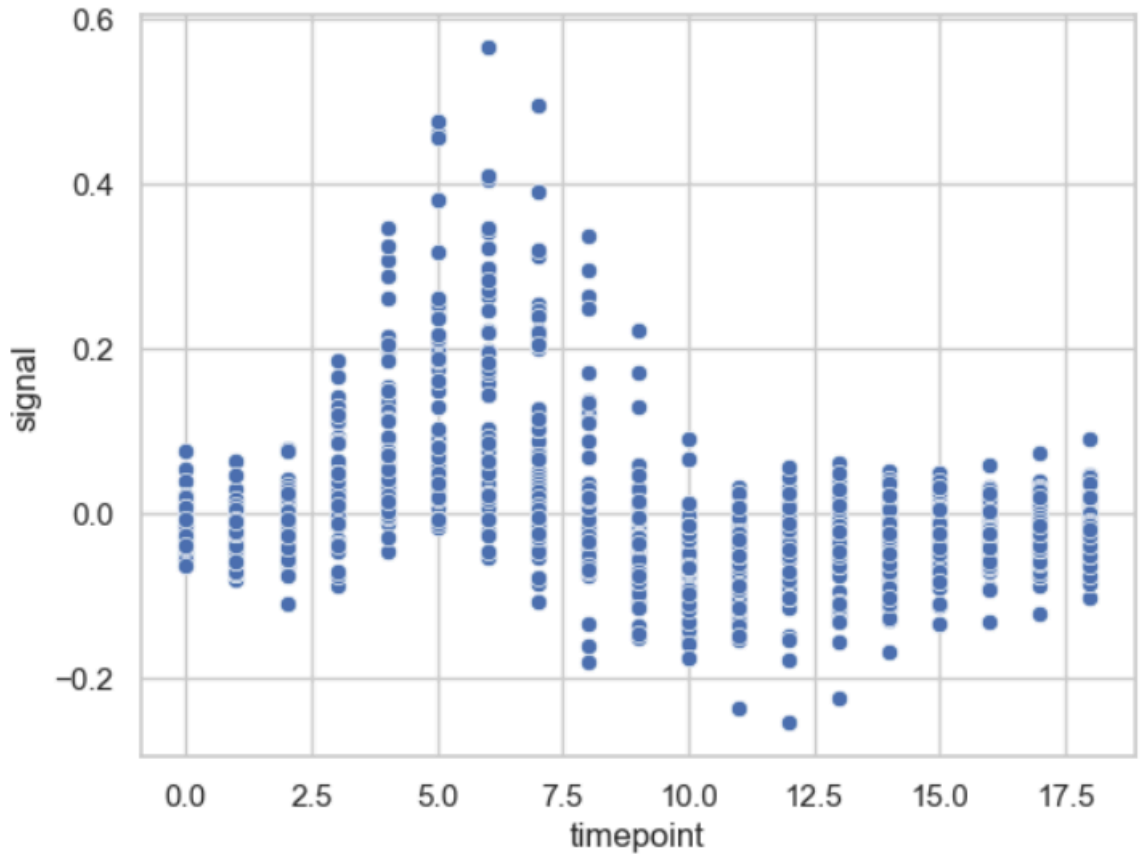
Seaborn

What is Seaborn

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on top matplotlib library and is also closely integrated with the data structures from pandas.

Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs so that we can switch between different visual representations for the same variables for a better understanding of the dataset.

```
fmri = seaborn.load_dataset("fmri")

seaborn.scatterplot(x="timepoint",y="signal",data=fmri)
seaborn.set(style='whitegrid')

plt.show()
```



x, y: Input data variables that should be numeric.

data: Dataframe where each column is a variable and each row is an observation.

size: Grouping variable that will produce points with different sizes.

style: Grouping variable that will produce points with different markers.

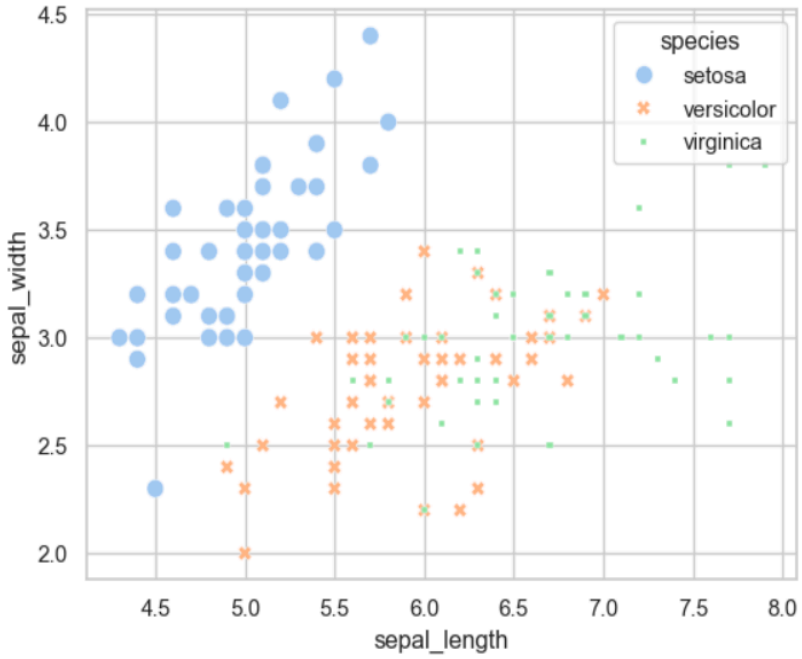
palette: Grouping variable that will produce points with different markers.

markers: Object determining how to draw the markers for different levels.

alpha: Proportional opacity of the points.

Returns: This method returns the Axes object with the plot drawn onto it.

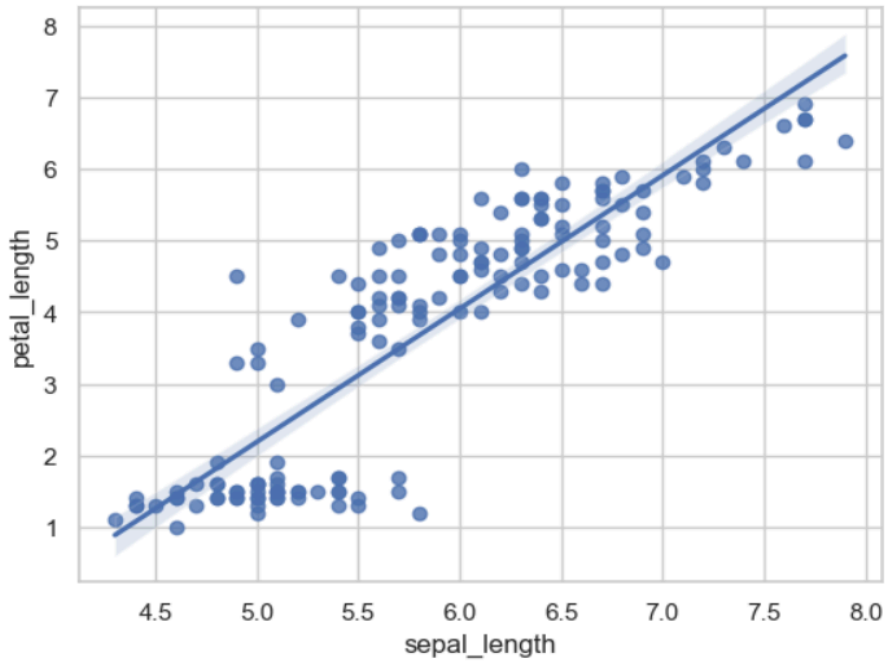
```
sns.scatterplot(iris, x ='sepal_length', y = 'sepal_width', hue = 'species', style = 'species', size = 'species', palette='pastel')
plt.show()
```



Regression Line Plot in Seaborn

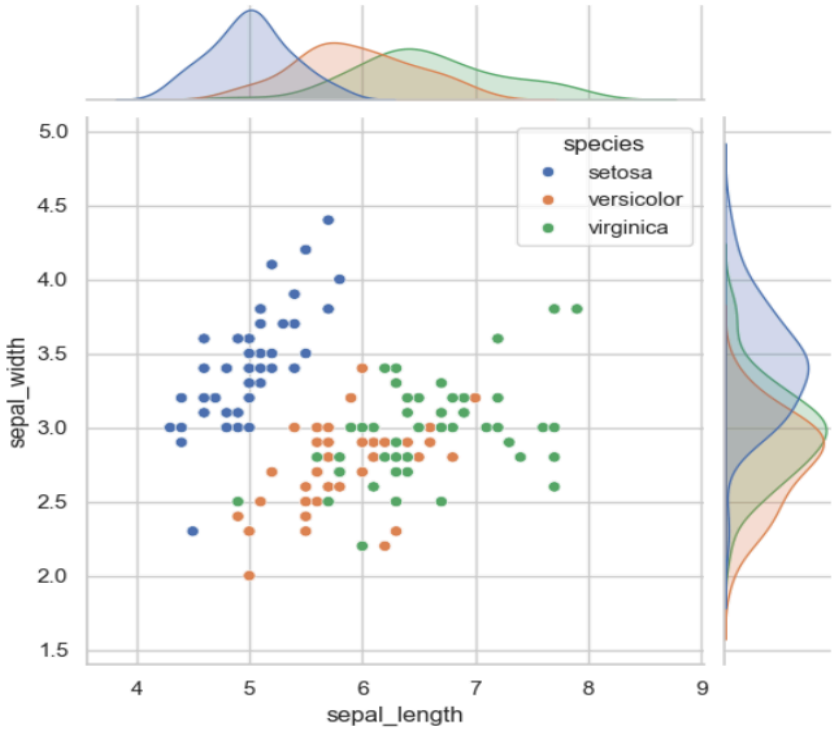
```
sns.regplot(iris, x = 'sepal_length', y = 'petal_length')
```

```
<Axes: xlabel='sepal_length', ylabel='petal_length'>
```



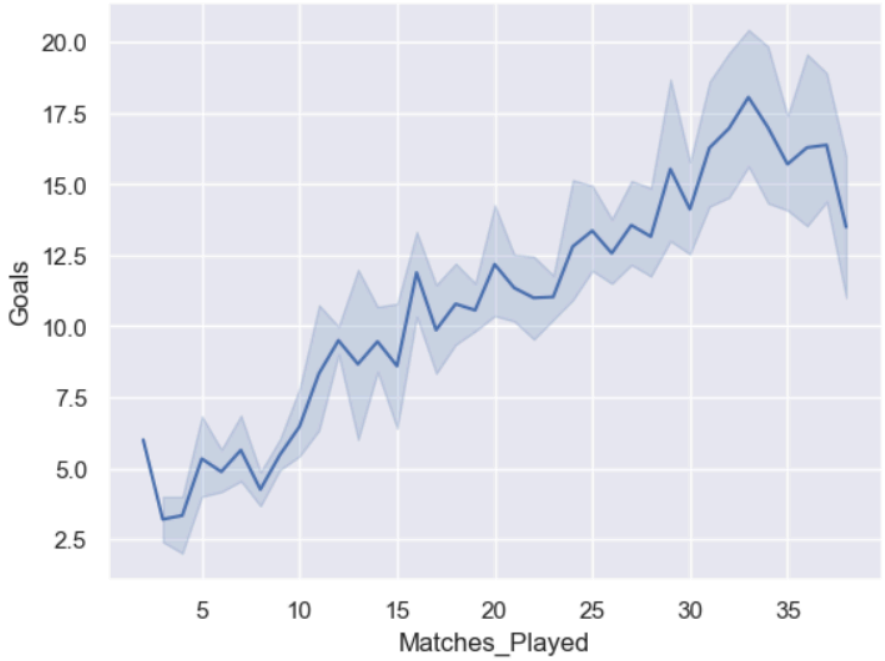
Joint Plot

```
sns.jointplot(iris, x ='sepal_length', y = 'sepal_width',hue = 'species')
plt.show()
```



Line Plot

```
sns.lineplot(data, x = 'Matches_Played', y = 'Goals')
plt.show()
```



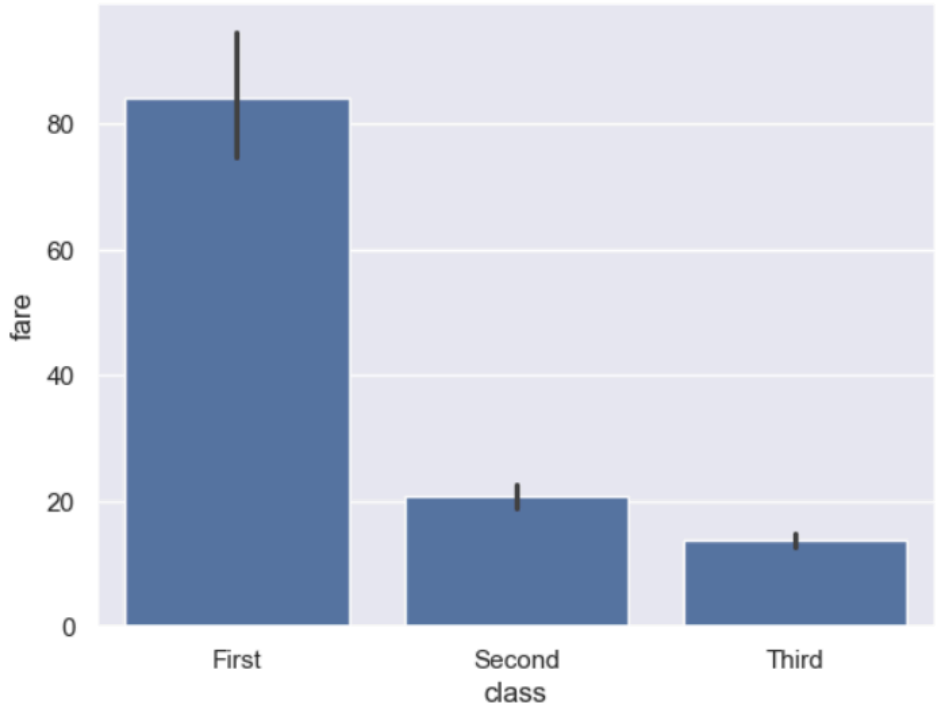
Bar Plot

seaborn.barplot() method is used to draw a barplot. 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.

- x, y : This parameter take names of variables in data or vector data, Inputs for plotting long-form data.
- hue : (optional) This parameter take column name for colour encoding.
- data : (optional) This parameter take DataFrame, array, or list of arrays, Dataset for plotting. If x and y are absent, this is interpreted as wide-form. Otherwise it is expected to be long-form.
- color : (optional) This parameter take matplotlib color, Color for all of the elements, or seed for a gradient palette.

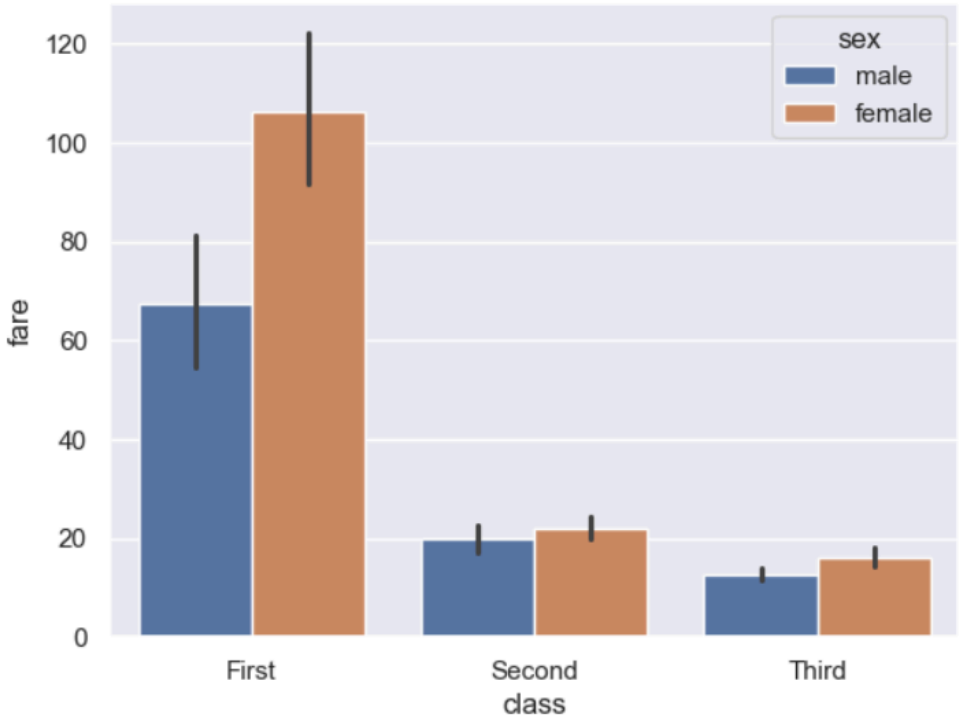
```
sns.barplot(titanic, x = 'class', y = 'fare')
```

```
<Axes: xlabel='class', ylabel='fare'>
```




```
sns.barplot(titanic, x = 'class', y = 'fare', hue = 'sex')
```

```
<Axes: xlabel='class', ylabel='fare'>
```

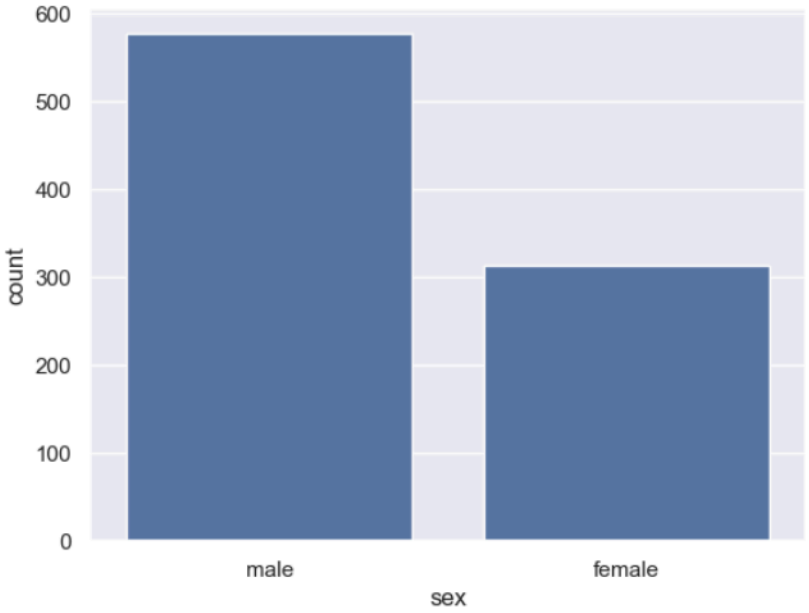


Count Plot

seaborn.countplot() method is used to Show the counts of observations in each categorical bin using bars.

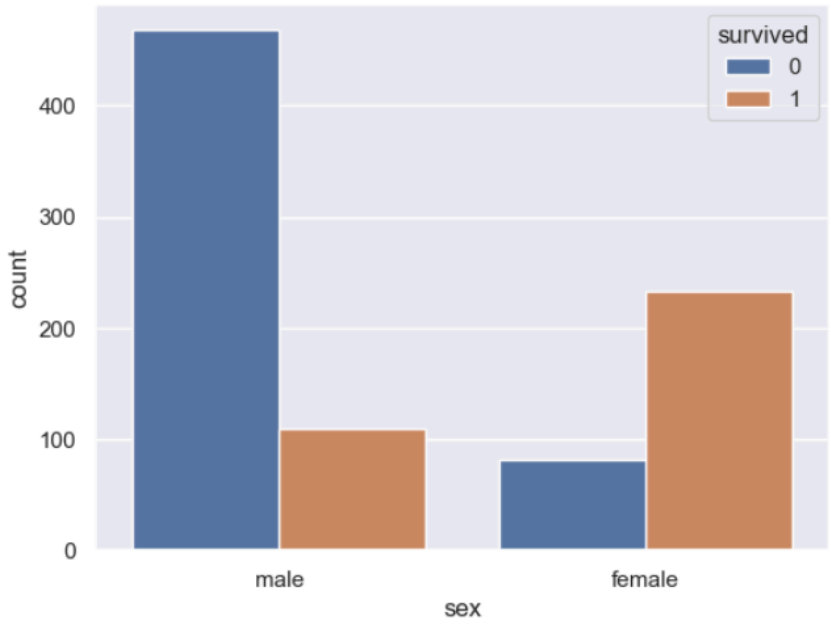
```
sns.countplot(titanic, x = 'sex')
```

```
<Axes: xlabel='sex', ylabel='count'>
```




```
sns.countplot(titanic, x = 'sex', hue = 'survived')
```

```
<Axes: xlabel='sex', ylabel='count'>
```

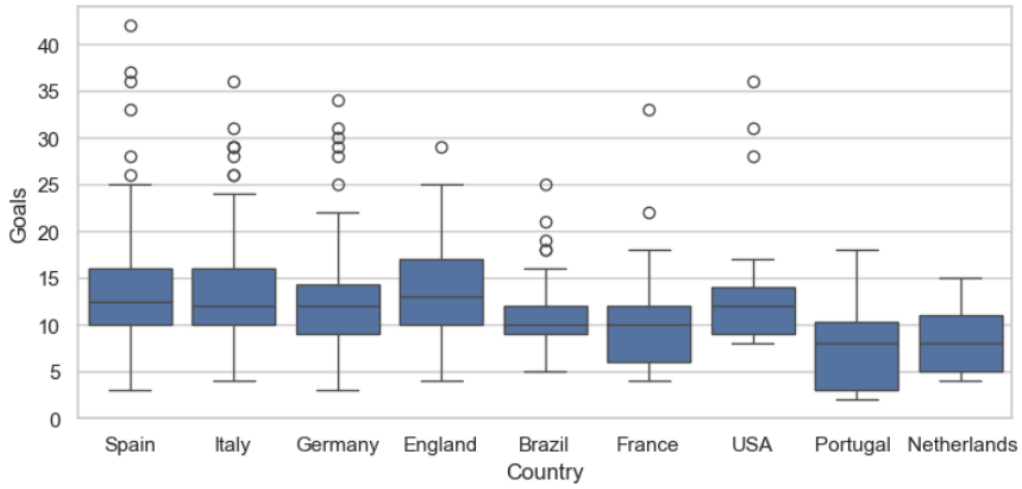


Box Plot

A box plot helps to maintain the distribution of quantitative data in such a way that it facilitates the comparisons between variables or across levels of a categorical variable. The main body of the box plot showing the quartiles and the median's confidence intervals if enabled. The medians have horizontal lines at the median of each box and while whiskers have the vertical lines extending to the most extreme, non-outlier data points and caps are the horizontal lines at the ends of the whiskers.

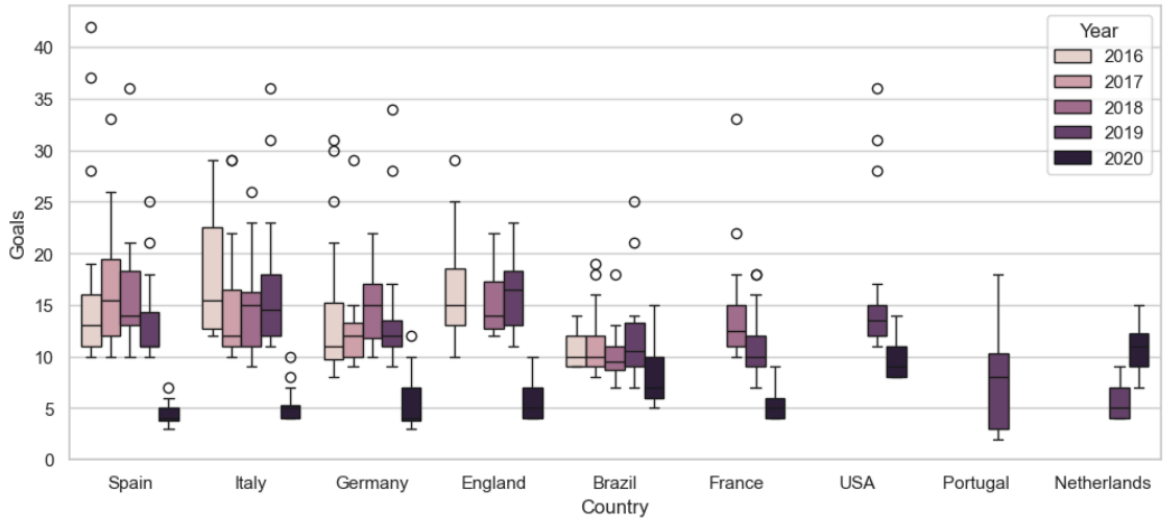
```
plt.figure(figsize = (9,4))
sns.boxplot(data, x = 'Country', y = 'Goals')
```

```
<Axes: xlabel='Country', ylabel='Goals'>
```




```
plt.figure(figsize = (12,5))
sns.boxplot(data, x = 'Country', y = 'Goals', hue = 'Year')
```

```
<Axes: xlabel='Country', ylabel='Goals'>
```



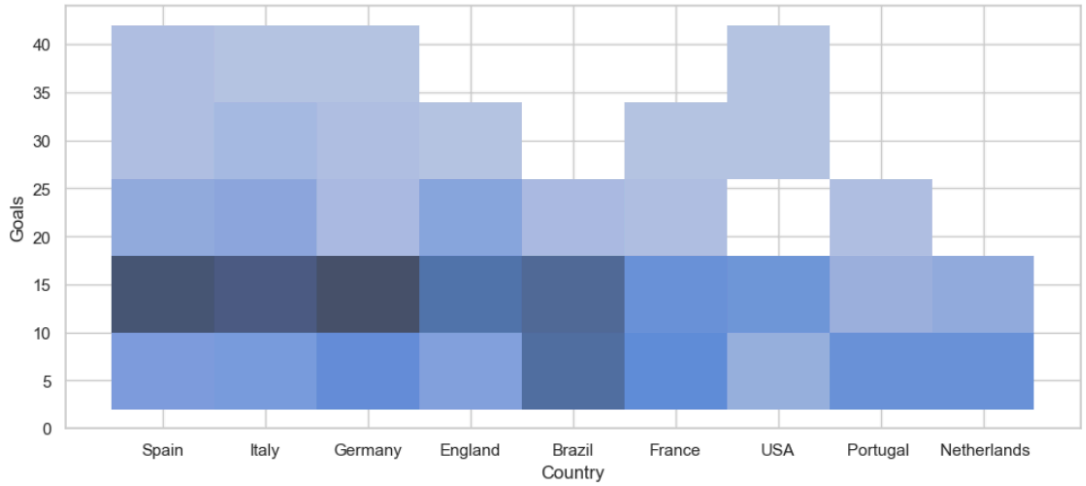
Histogram

Histograms are visualization tools that represent the distribution of a set of continuous data. In a histogram, the data is divided into a set of intervals or bins (usually on the x-axis) and the count of data points that fall into each bin corresponding to the height of the bar above that bin. These bins may or may not be equal in width but are adjacent (with no gaps).

A density plot (also known as kernel density plot) is another visualization tool for evaluating data distributions. It can be considered as a smoothed histogram. The peaks of a density plot help display where values are concentrated over the interval. There are a variety of smoothing techniques. Kernel Density Estimation (KDE) is one of the techniques used to smooth a histogram.

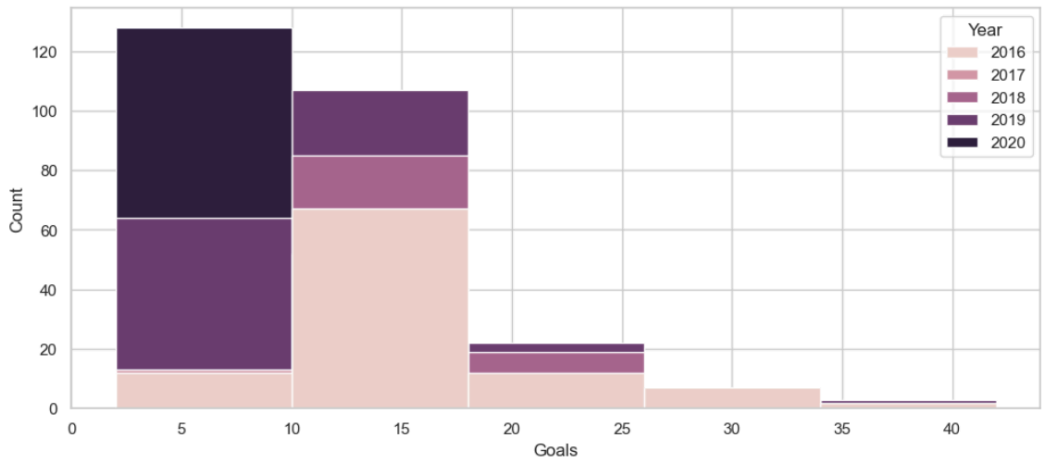
```
plt.figure(figsize = (12,5))
sns.histplot(data, x = 'Country', y = 'Goals', bins = 5)
```

```
<Axes: xlabel='Country', ylabel='Goals'>
```




```
plt.figure(figsize = (12,5))
sns.histplot(data, x = 'Goals', bins = 5, hue = 'Year',alpha = 1)
```

```
<Axes: xlabel='Goals', ylabel='Count'>
```

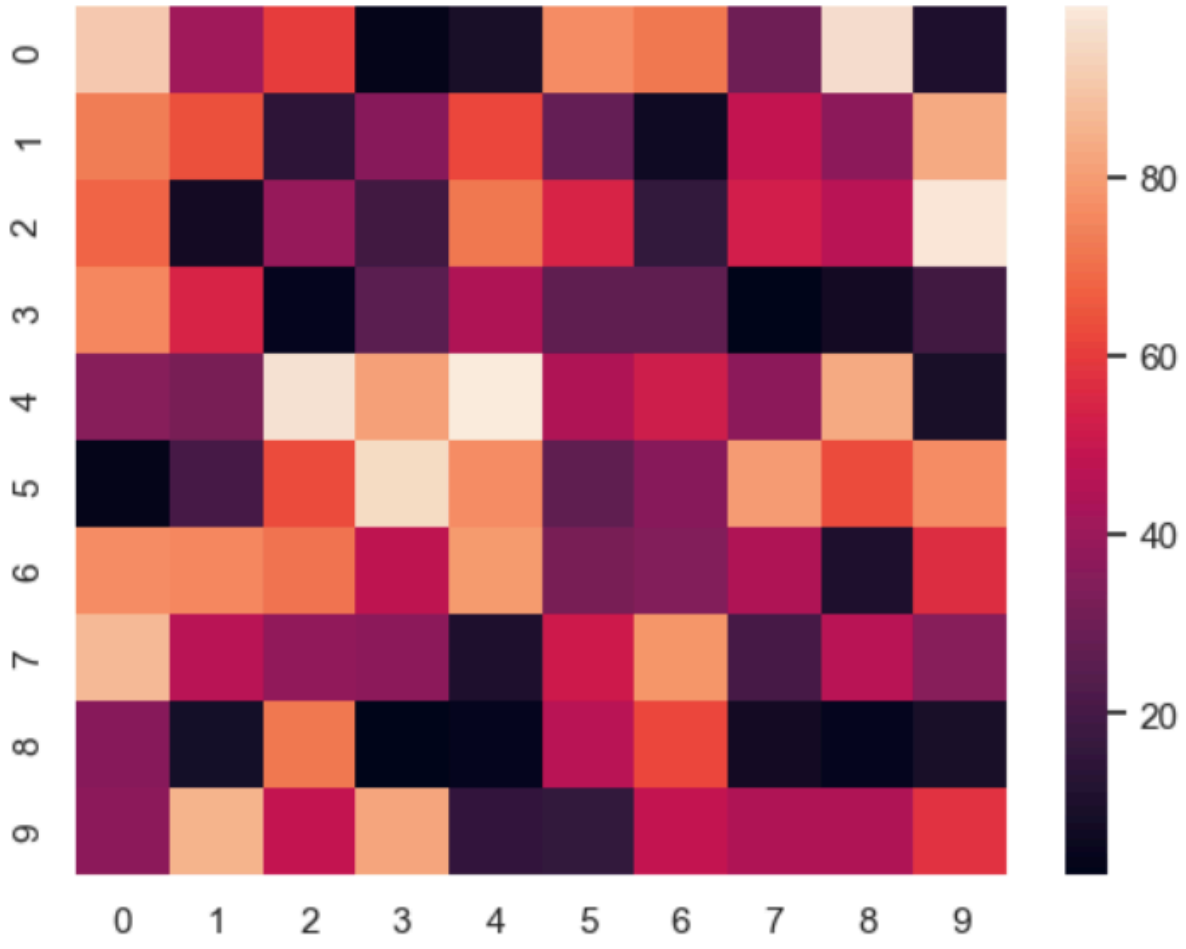


HeatMap

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.

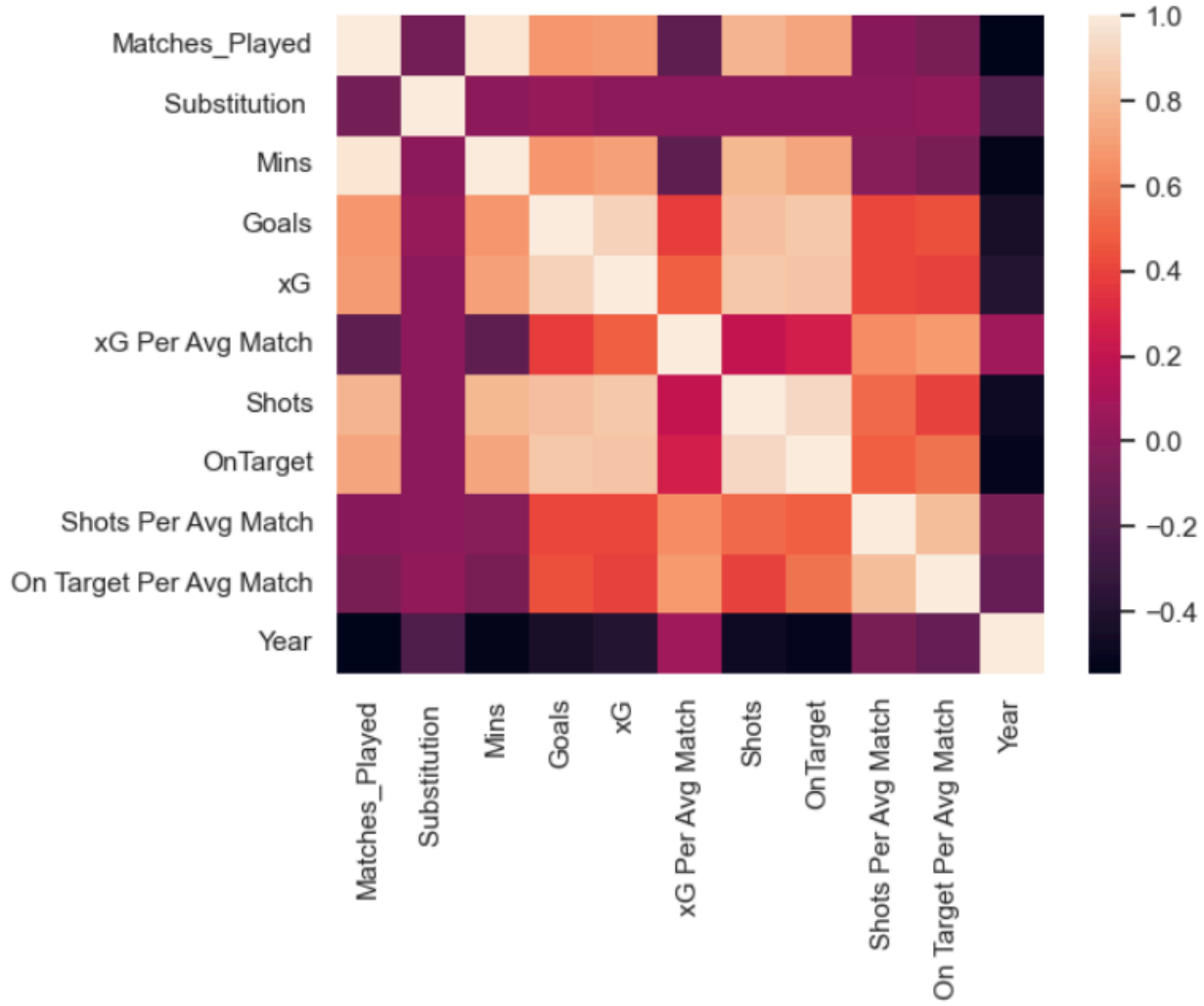
```
xd = np.random.randint(1,100,(10,10))
sns.heatmap(xd)
```

```
<Axes: >
```



```
sns.heatmap(data.corr(numeric_only=True))
```

```
<Axes: >
```

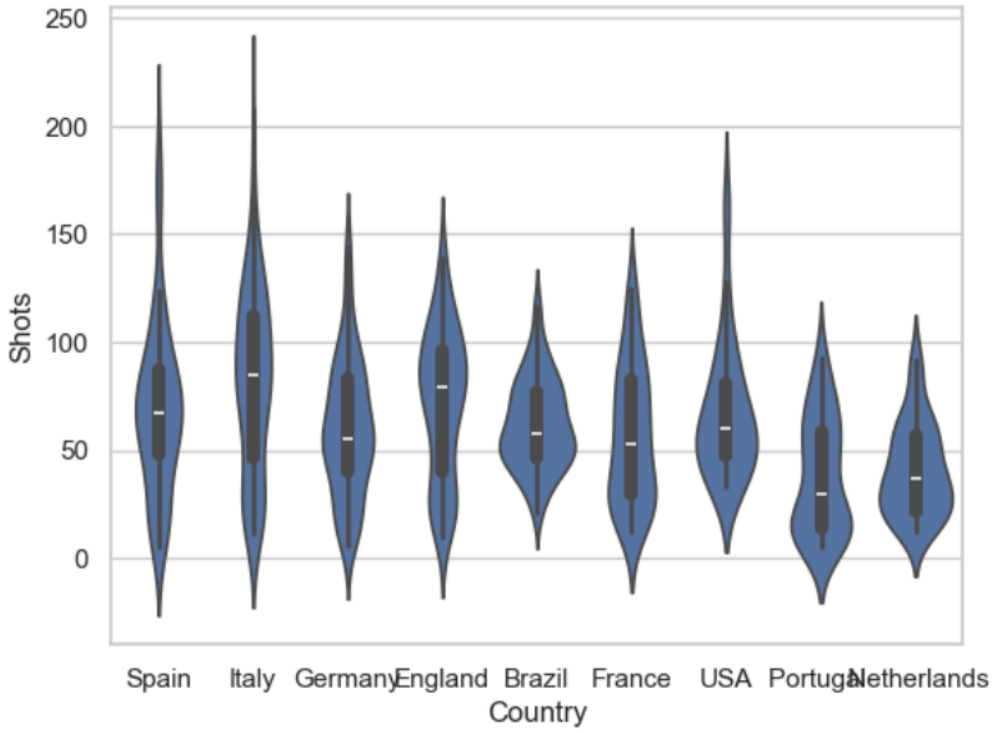


Violin Plot

A violin plot plays a similar activity that is pursued through whisker or box plot do. As it shows several quantitative data across one or more categorical variables. It can be an effective and attractive way to show multiple data at several units. A “wide-form” Data Frame helps to maintain each numeric column which can be plotted on the graph. It is possible to use NumPy or Python objects, but pandas objects are preferable because the associated names will be used to annotate the axes.

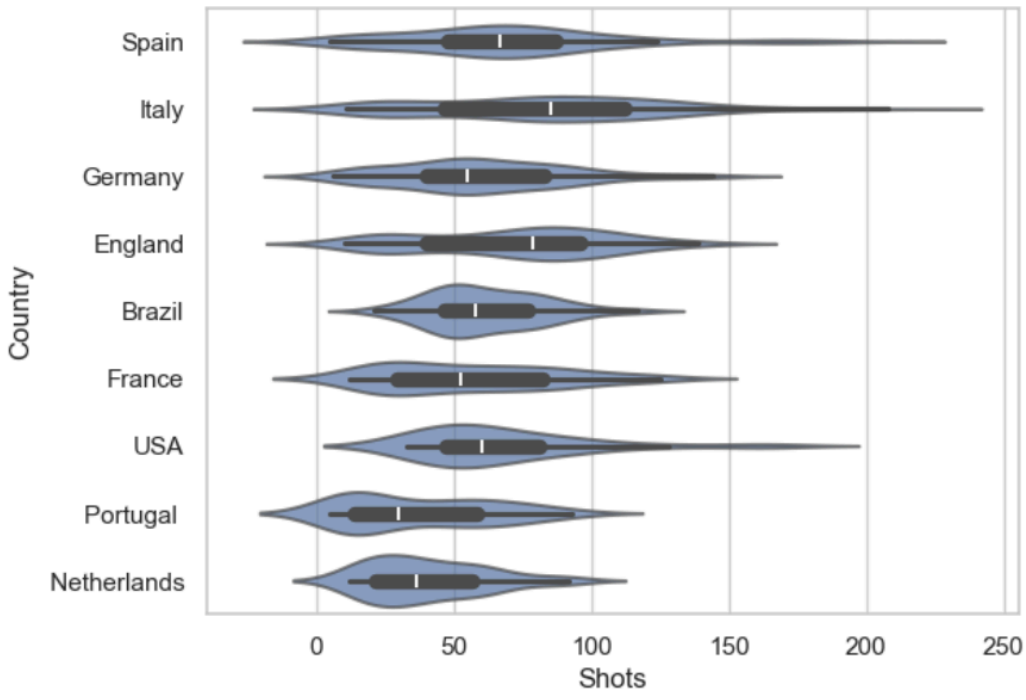
```
sns.violinplot(data, x = 'Country', y = 'Shots')
```

```
<Axes: xlabel='Country', ylabel='Shots'>
```




```
sns.violinplot(data, x = 'Shots', y = 'Country',  linewidth = 1.5, alpha = 0.7)
```

```
<Axes: xlabel='Shots', ylabel='Country'>
```

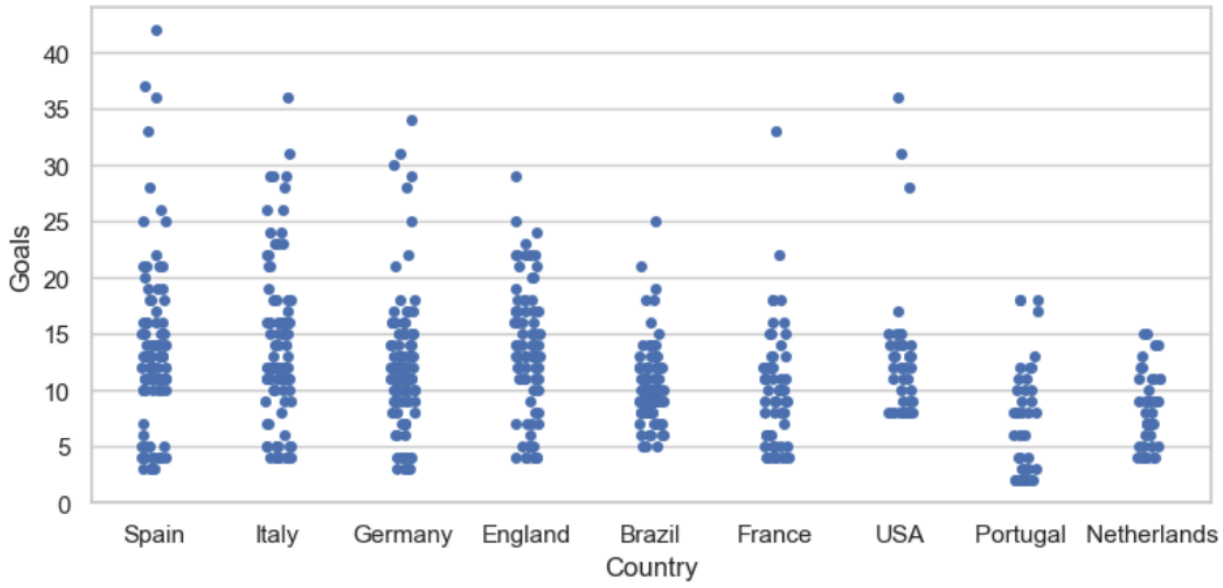


Strip Plot

A strip plot is drawn on its own. It is a good complement to a boxplot or violinplot in cases where all observations are shown along with some representation of the underlying distribution. It is used to draw a scatter plot based on the category.

```
plt.figure(figsize = (9,4))
sns.stripplot(data, x = 'Country', y = 'Goals')
```

```
<Axes: xlabel='Country', ylabel='Goals'>
```




```
plt.figure(figsize = (9,4))
sns.stripplot(data, x = 'Country', y = 'Goals', marker="s", alpha=0.2, hue = 'Country', size = 20)
```

```
<Axes: xlabel='Country', ylabel='Goals'>
```