UNIT 4

Data Visualization using Matplotlib

Data Visualization is the process of presenting data in the form of graphs or charts. It helps to understand large and complex amounts of data very easily. It allows the decision-makers to make decisions very efficiently and also allows them in identifying new trends and patterns very easily. It is also used in high-level data analysis for Machine Learning and Exploratory Data Analysis (EDA). Data visualization can be done with various tools like Tableau, Power BI, Python.

In this article, we will discuss how to visualize data with the help of the Matplotlib library of Python.

Matplotlib

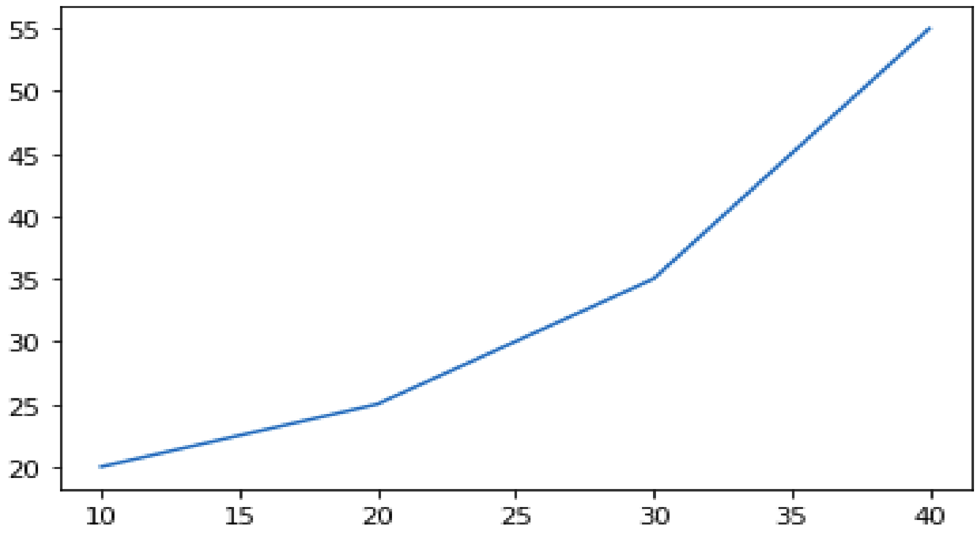
Matplotlib is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consist of several plots like line chart, bar chart, histogram, etc. It provides a lot of flexibility but at the cost of writing more code.

Pyplot

Pyplot is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. The various plots we can utilize using Pyplot are Line Plot, Histogram, Scatter, 3D Plot, Image, Contour, and Polar.

After knowing a brief about Matplotlib and pyplot let's see how to create a simple plot.

```
import matplotlib.pyplot as plt


# initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

# plotting the data

plt.plot(x, y)

plt.show()
```



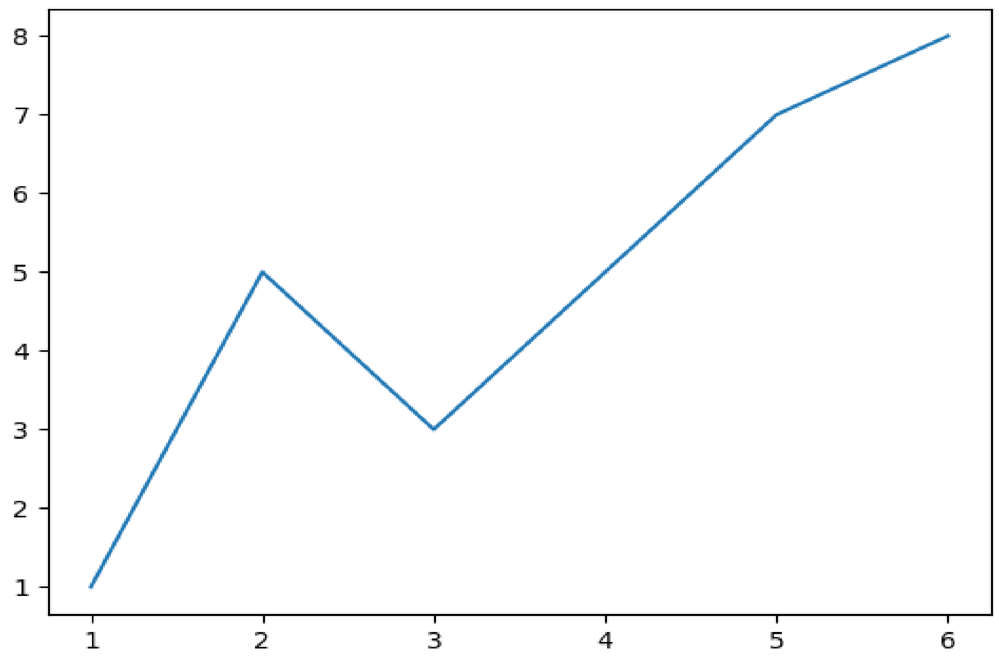
Plot a Line Plot in Matplotlib

To plot a line plot in Matplotlib, you use the generic plot() function from the PyPlot instance. There's no specific lineplot() function - the generic one automatically plots using lines or markers.

Let's make our own small dataset to work with:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5, 6]
y = [1, 5, 3, 5, 7, 8]

plt.plot(x, y)
plt.show()
```

This results in a simple line plot:



matplotlib.pyplot.scatter() in Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is used for plotting various plots in Python like scatter plot, bar charts, pie charts, line plots, histograms, 3-D plots and many more.

matplotlib.pyplot.scatter()

Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The **scatter()** method in the matplotlib library is used to draw a scatter plot. Scatter plots are widely used to represent relation among variables and how change in one affects the other.

Syntax

The syntax for scatter() method is given below:

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

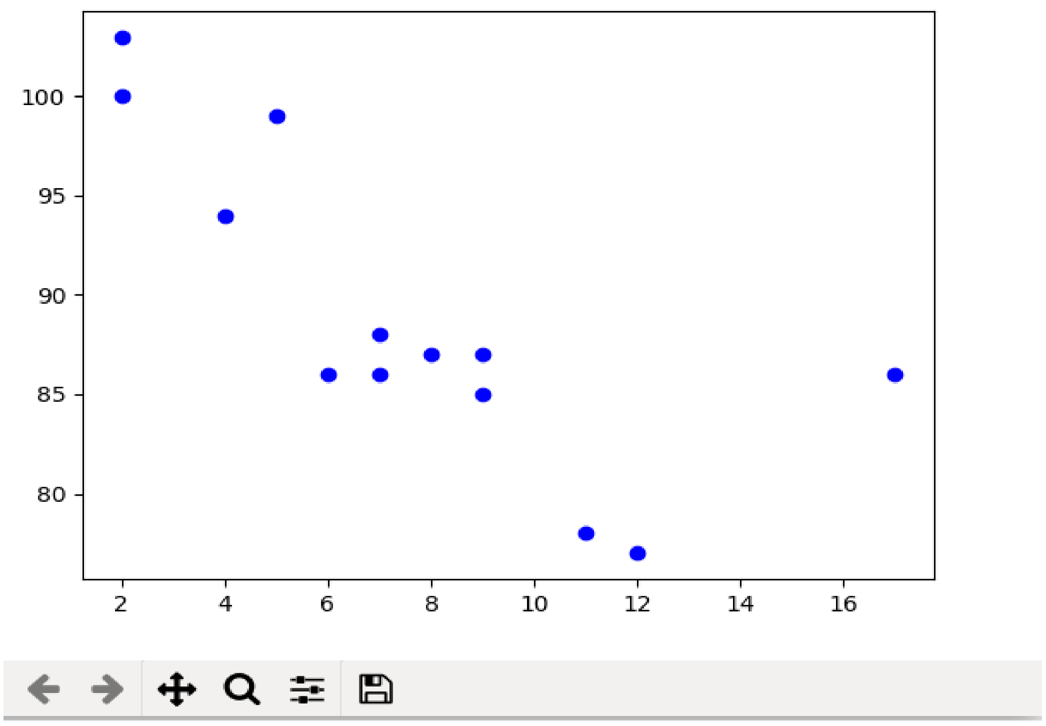
The scatter() method takes in the following parameters:

- **x_axis_data-** An array containing x-axis data
- **y_axis_data-** An array containing y-axis data
- **s-** marker size (can be scalar or array of size equal to size of x or y)
- **c-** color of sequence of colors for markers
- marker- marker style
- **cmap-** cmap name

- **linewidths-** width of marker border
- **edgecolor-** marker border color
- **alpha-** blending value, between 0 (transparent) and 1 (opaque)

Except x_axis_data and y_axis_data all other parameters are optional and their default value is None. Below are the scatter plot examples with various parameters.
Example 1: This is the most basic example of a scatter plot.

```
import matplotlib.pyplot as plt

x =[5, 7, 8, 7, 2, 17, 2, 9,

     4, 11, 12, 9, 6]

y =[99, 86, 87, 88, 100, 86,

     103, 87, 94, 78, 77, 85, 86]

plt.scatter(x, y, c ="blue")

# To show the plot

plt.show()
```

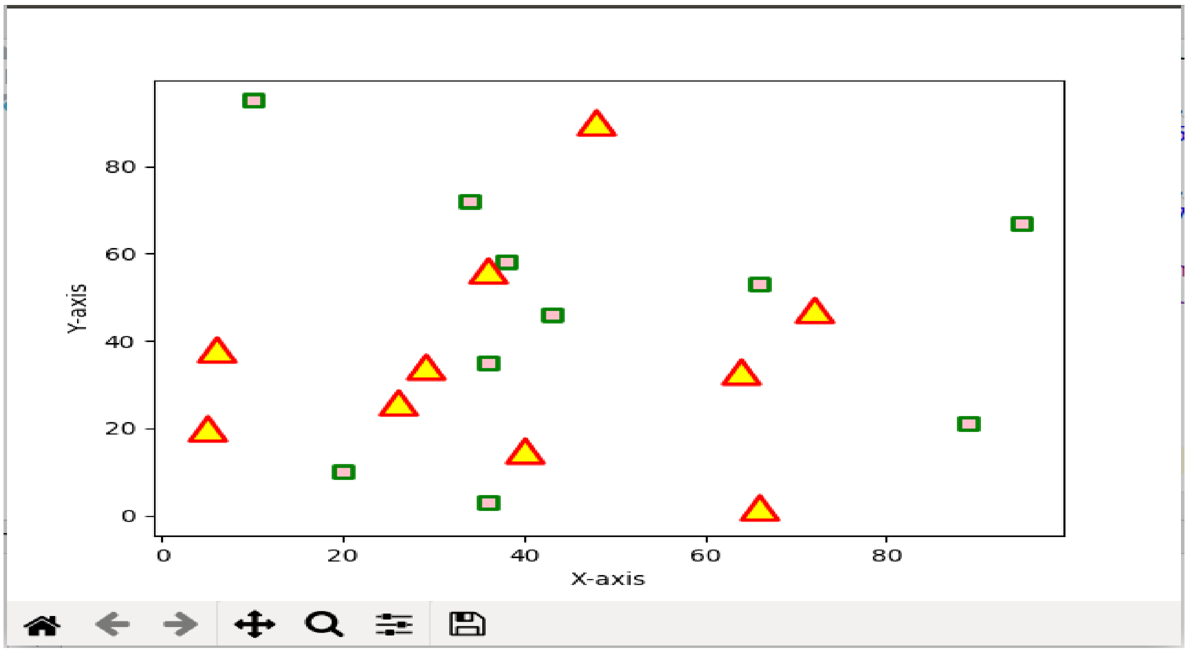


Example 2: Scatter plot with different shape and colour for two datasets.

- Python3

```
importmatplotlib.pyplot as plt

# dataset-1
x1 =[89, 43, 36, 36, 95, 10,
      66, 34, 38, 20]

y1 =[21, 46, 3, 35, 67, 95,
      53, 72, 58, 10]

# dataset2
x2 =[26, 29, 48, 64, 6, 5,
      36, 66, 72, 40]

y2 =[26, 34, 90, 33, 38,
      20, 56, 2, 47, 15]

plt.scatter(x1, y1, c ="pink",
            linewidths =2,
            marker ="s",
            edgecolor ="green",
            s =50)

plt.scatter(x2, y2, c ="yellow",
            linewidths =2,
            marker ="^",
            edgecolor ="red",
            s =200)

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Output



Python - Data visualization using Bokeh

Bokeh is a data visualization library in Python that provides high-performance interactive charts and plots. Bokeh output can be obtained in various mediums like notebook, html and server. It is possible to embed bokeh plots in Django and flask apps.

Bokeh provides two visualization interfaces to users:

bokeh.models : *A low level interface that provides high flexibility to application developers.*

bokeh.plotting : *A high level interface for creating visual glyphs.*

To install bokeh package, run the following command in the terminal:

```
pip install bokeh
```

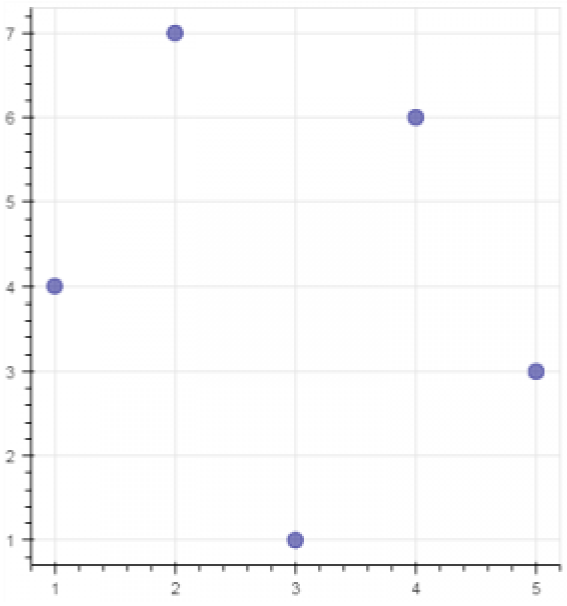
The dataset used for generating bokeh graphs is collected from Kaggle.

Code #1: Scatter Markers
To create scatter circle markers, circle() method is used.

```
# import modules
frombokeh.plotting importfigure, output_notebook, show

# output to notebook
output_notebook()

# create figure
p =figure(plot_width =400, plot_height =400)

# add a circle renderer with
# size, color and alpha
p.circle([1, 2, 3, 4, 5], [4, 7, 1, 6, 3],
        size =10, color ="navy", alpha =0.5)

# show the results
show(p)
```

Output :

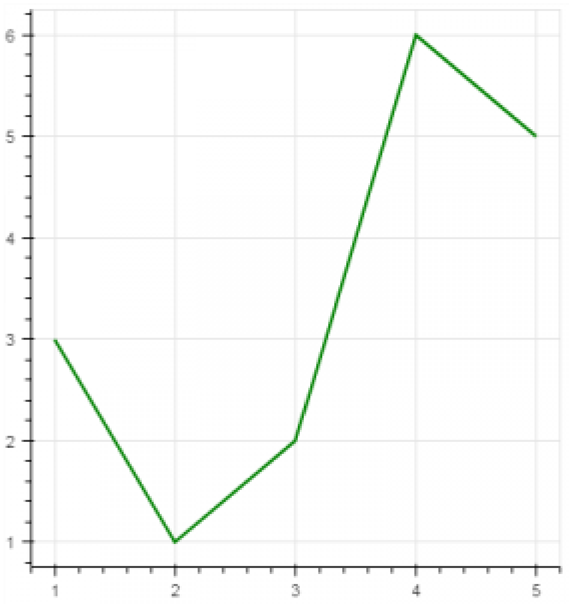
The image part with relationship ID rId12 was not found in the file.



Code #2: Single line
To create a single line, line() method is used.

```
# import modules
frombokeh.plotting importfigure, output_notebook, show

# output to notebook
output_notebook()

# create figure
p =figure(plot_width =400, plot_height =400)

# add a line renderer
p.line([1, 2, 3, 4, 5], [3, 1, 2, 6, 5],
        line_width =2, color ="green")

# show the results
show(p)
```

Output :

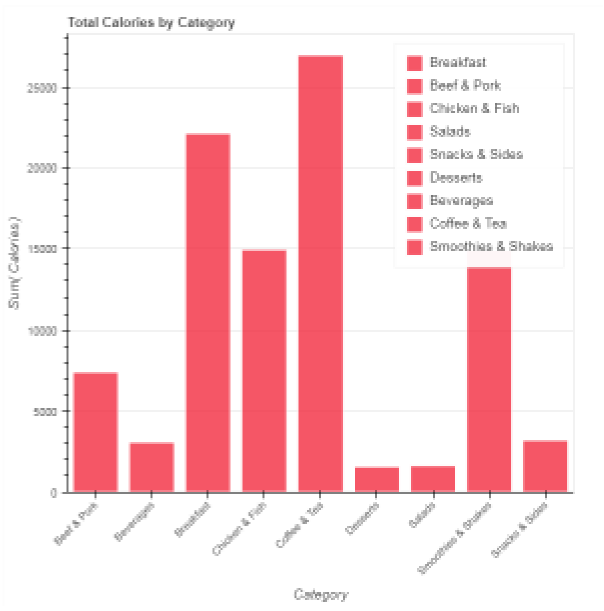


Code #3: Bar Chart
Bar chart presents categorical data with rectangular bars. The length of the bar is proportional to the values that are represented.

```
# import necessary modules
importpandas as pd
frombokeh.charts importBar, output_notebook, show

# output to notebook
output_notebook()
```

```
# read data in dataframe
df =pd.read_csv(r"D:/kaggle/mcdonald/menu.csv")

# create bar
p =Bar(df, "Category", values ="Calories",
        title ="Total Calories by Category",
                        legend ="top_right")

# show the results
show(p)
```

Output :



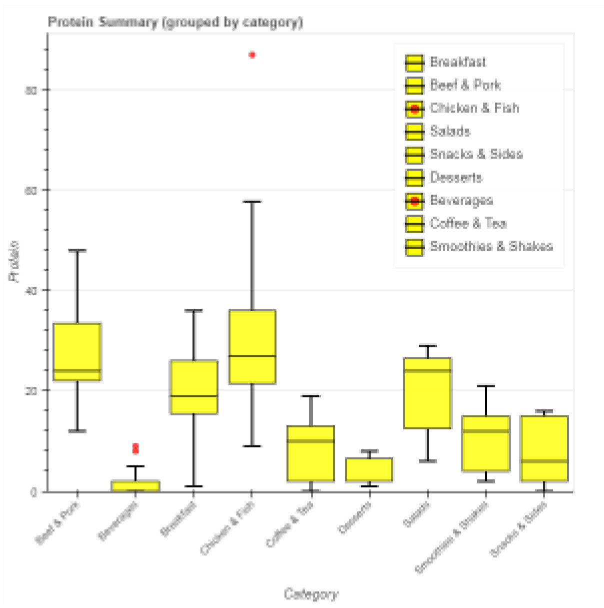
Code #4: Box Plot

Box plot is used to represent statistical data on a plot. It helps to summarize statistical properties of various data groups present in the data.

```
# import necessary modules
frombokeh.charts importBoxPlot, output_notebook, show
importpandas as pd

# output to notebook
output_notebook()

# read data in dataframe
df =pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")
```

```
# create bar
p =BoxPlot(df, values ="Protein", label ="Category",
          color ="yellow", title ="Protein Summary (grouped by category)",
           legend ="top_right")

# show the results
show(p)
```

Output :

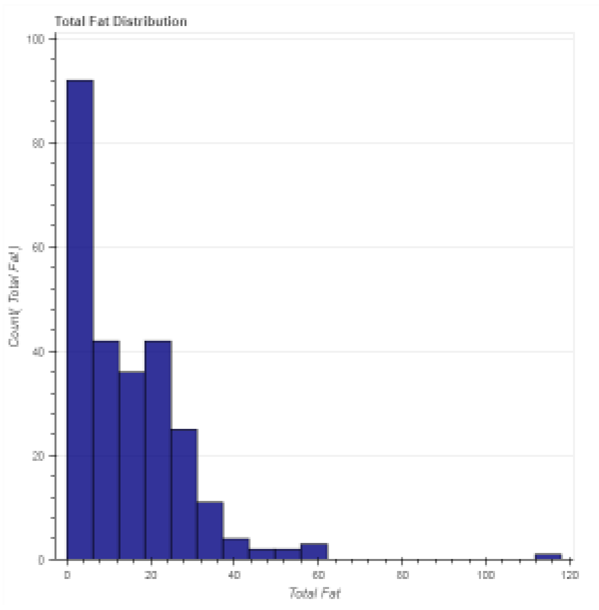


Code #5: Histogram
Histogram is used to represent distribution of numerical data. The height of a rectangle in a histogram is proportional to the frequency of values in a class interval.

```
# import necessary modules
frombokeh.charts importHistogram, output_notebook, show
importpandas as pd

# output to notebook
output_notebook()

# read data in dataframe
df =pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")

# create histogram
p =Histogram(df, values ="Total Fat",
            title ="Total Fat Distribution",
            color ="navy")

# show the results
show(p)
```

Output :

The image part with relationship ID rId17 was not found in the file.



Code #6: Scatter plot

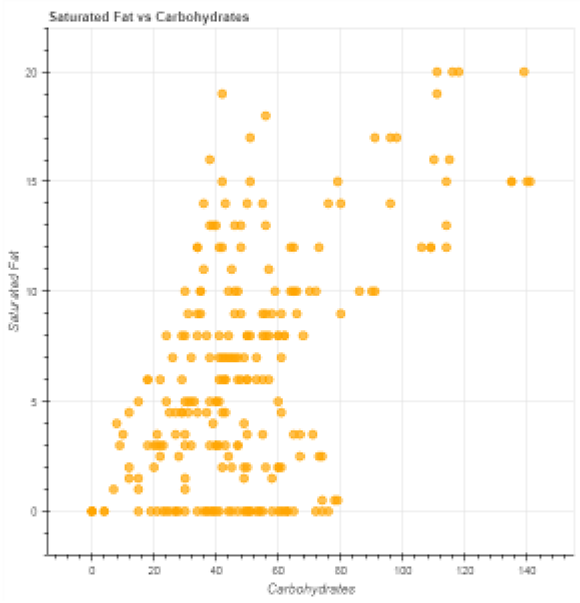
Scatter plot is used to plot values of two variables in a dataset. It helps to find correlation among the two variables that are selected.

```
# import necessary modules
frombokeh.charts importScatter, output_notebook, show
importpandas as pd

# output to notebook
output_notebook()

# read data in dataframe
df =pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")

# create scatter plot
p =Scatter(df, x ="Carbohydrates", y ="Saturated Fat",
         title ="Saturated Fat vs Carbohydrates",
```

•

```
            xlabel ="Carbohydrates", ylabel ="Saturated Fat",
            color ="orange")

# show the results
show(p)
```

Output :



Three-dimensional Plotting in Python using Matplotlib

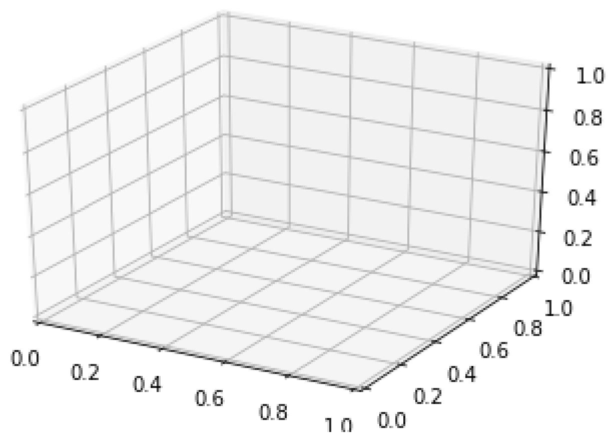
[Matplotlib]() was introduced keeping in mind, only two-dimensional plotting. But at the time when the release of 1.0 occurred, the 3d utilities were developed upon the 2d and thus, we have 3d implementation of data available today! The 3d plots are enabled by importing the mplot3d toolkit. In this article, we will deal with the 3d plots using matplotlib.
Example:

• Python3

```
importnumpy as np
```

```
importmatplotlib.pyplot as plt


fig =plt.figure()
ax =plt.axes(projection ='3d')
```

Output:



With the above syntax three -dimensional axes are enabled and data can be plotted in 3 dimensions. 3 dimension graph gives a dynamic approach and makes data more interactive. Like 2-D graphs, we can use different ways to represent 3-D graph. We can make a scatter plot, contour plot, surface plot, etc. Let’s have a look at different 3-D plots.

Plotting 3-D Lines and Points

Graph with lines and point are the simplest 3 dimensional graph. **ax.plot3d and ax.scatter** are the function to plot line and point graph respectively.

Example 1: 3 dimensional line graph

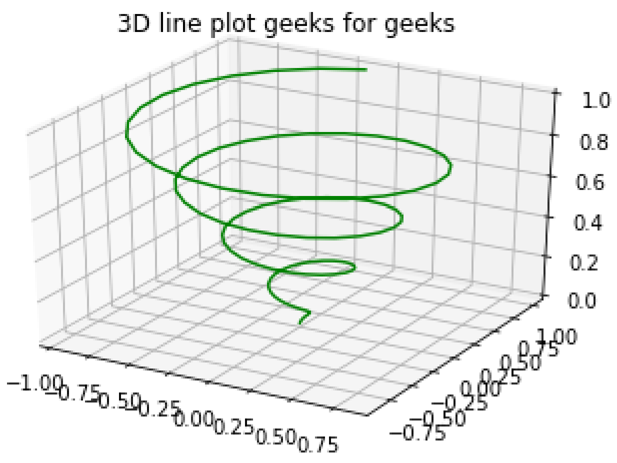
- Python3

```
# importing mplot3d toolkits, numpy and matplotlib
frommpl_toolkits importmplot3d
importnumpy as np
importmatplotlib.pyplot as plt
```

```
fig =plt.figure()

# syntax for 3-D projection
ax =plt.axes(projection ='3d')

# defining all 3 axis
z =np.linspace(0, 1, 100)
x =z *np.sin(25*z)
y =z *np.cos(25*z)

# plotting
ax.plot3D(x, y, z, 'green')
ax.set_title('3D line plot geeks for geeks')
plt.show()
```

Output:

```
Text(0.5, 0.92, '3D line plot geeks for geeks')
```



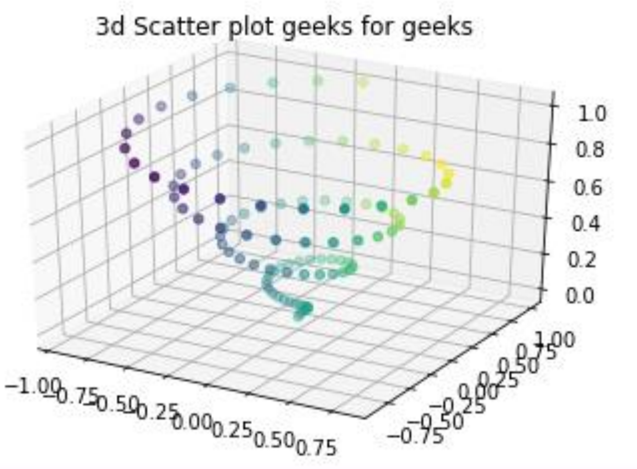
Example 2: 3 dimensional scattered graph

- Python3

```
# importing mplot3d toolkits
frommpl_toolkits importmplot3d
importnumpy as np
importmatplotlib.pyplot as plt
```

```
fig =plt.figure()

# syntax for 3-D projection
ax =plt.axes(projection ='3d')

# defining axes
z =np.linspace(0, 1, 100)
x =z *np.sin(25*z)
y =z *np.cos(25*z)
c =x +y
ax.scatter(x, y, z, c =c)

# syntax for plotting
ax.set_title('3d Scatter plot geeks for geeks')
plt.show()
```

Output:



Matplotlib - Contour Plot

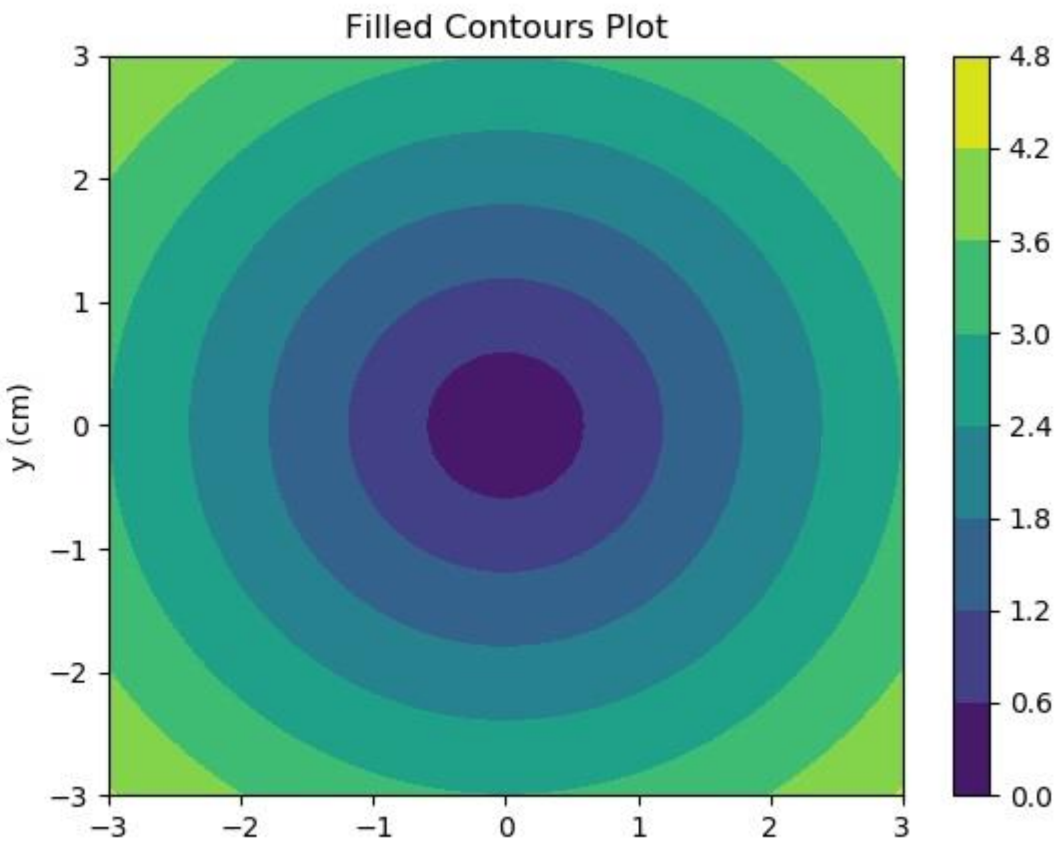
Contour plots (sometimes called Level Plots) are a way to show a three-dimensional surface on a two-dimensional plane. It graphs two predictor variables X Y on the y-axis and a response variable Z as contours. These contours are sometimes called the z-slices or the iso-response values.

A contour plot is appropriate if you want to see how alue Z changes as a function of two inputs X and Y, such that Z = f(X,Y). A contour line or isoline of a function of two variables is a curve along which the function has a constant value.

The independent variables x and y are usually restricted to a regular grid called meshgrid. The numpy.meshgrid creates a rectangular grid out of an array of x values and an array of y values.

Matplotlib API contains contour() and contourf() functions that draw contour lines and filled contours, respectively. Both functions need three parameters x,y and z.

```
import numpy as np
import matplotlib.pyplot as plt
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
fig,ax=plt.subplots(1,1)
cp = ax.contourf(X, Y, Z)
fig.colorbar(cp) # Add a colorbar to a plot
ax.set_title('Filled Contours Plot')
#ax.set_xlabel('x (cm)')
ax.set_ylabel('y (cm)')
plt.show()
```



Density Plots with Pandas in Python

Density Plot is a type of data visualization tool. It is a variation of the histogram that uses ‘kernel smoothing’ while plotting the values. It is a continuous and smooth version of a histogram inferred from a data.

Density plots uses Kernel Density Estimation (so they are also known as Kernel density estimation plots or KDE) which is a probability density function. The region of plot with a higher peak is the region with maximum data points residing between those values.

Density plots can be made using pandas, seaborn, etc. In this article, we will generate density plots using Pandas. We will be using two datasets of the Seaborn Library namely – ‘car_crashes’ and ‘tips’.

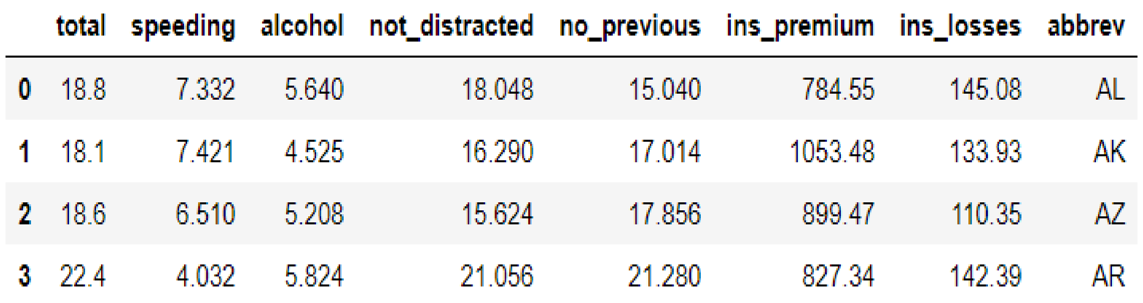
Syntax: ***pandas.DataFrame.plot.density | pandas.DataFrame.plot.kde***
where ***pandas*** *-> the dataset of the type ‘pandas dataframe’*
Dataframe *-> the column for which the density plot is to be drawn*
plot *-> keyword directing to draw a plot/graph for the given column*
density *-> for plotting a density graph*
kde *-> to plot a density graph using the Kernel Density Estimation function*
Example 1: Given the dataset ‘car_crashes’, let’s find out using the density plot which is the most common speed due to which most of the car crashes happened.

Python3

```
# importing the libraries

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt



# loading the dataset

# from seaborn library

data = sns.load_dataset('car_crashes')
```

```
# viewing the dataset

print(data.head(4))
```

Output:



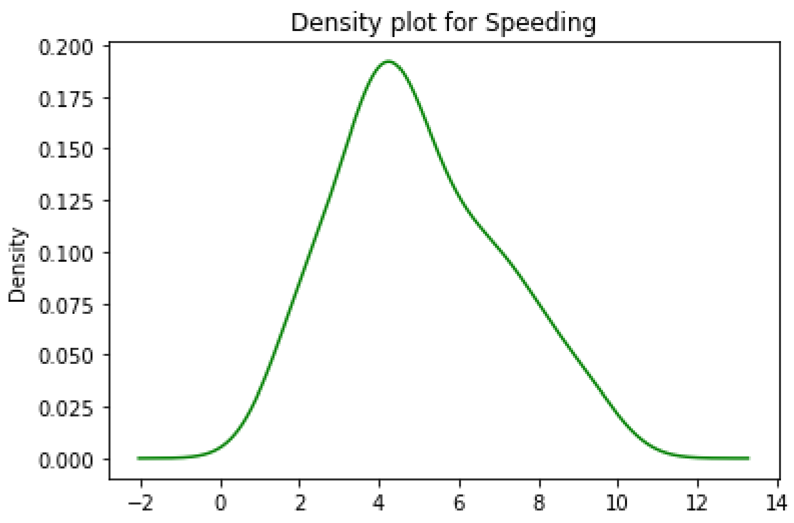
	total	speeding	alcohol	not_distracted	no_previous	ins_premium	ins_losses	abbrev
0	18.8	7.332	5.640	18.048	15.040	784.55	145.08	AL
1	18.1	7.421	4.525	16.290	17.014	1053.48	133.93	AK
2	18.6	6.510	5.208	15.624	17.856	899.47	110.35	AZ
3	22.4	4.032	5.824	21.056	21.280	827.34	142.39	AR

Plotting the graph:

Python3

```
# plotting the density plot

# for 'speeding' attribute

# using plot.density()

data.speeding.plot.density(color='green')

plt.title('Density plot for Speeding')

plt.show()
```

Output:



Using a density plot, we can figure out that the speed between 4-5 (kmph) was the most common for crash crashes in the dataset because of it being high density (high peak) region.

Seaborn

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It is built on the top of matplotlib library and also closely integrated into the data structures from pandas.

Installation

For python environment :

```
pip install seaborn
```

For conda environment :

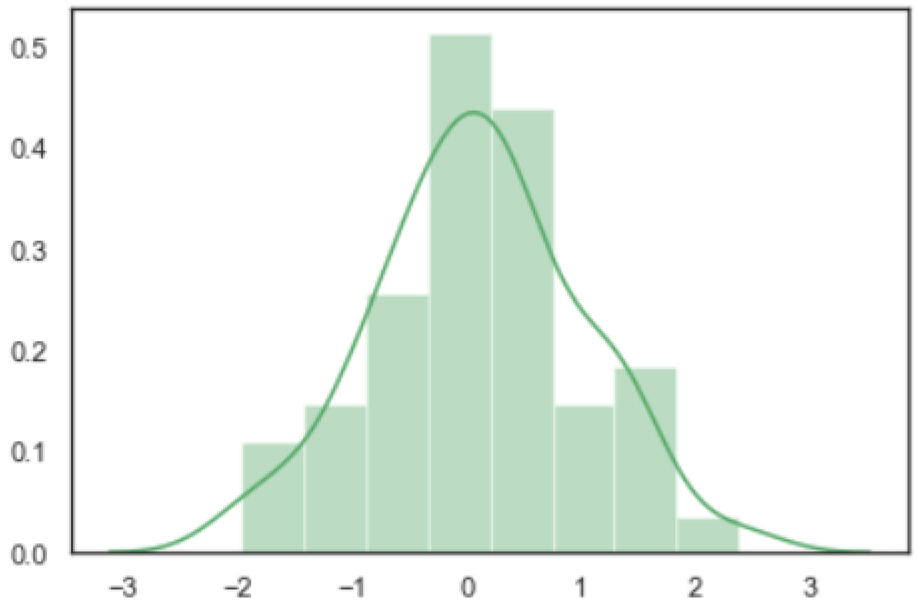
```
conda install seaborn
```

Let's create Some basic plots using seaborn:

- Python3

```
# Importing libraries

import numpy as np

import seaborn as sns




# Selecting style as white,
# dark, whitegrid, darkgrid

# or ticks

sns.set( style = "white" )



# Generate a random univariate

# dataset

rs = np.random.RandomState( 10 )

d = rs.normal( size = 50 )



# Plot a simple histogram and kde

# with binsize determined automatically

sns.distplot(d, kde = True, color = "g")
```

Output:



Seaborn: statistical data visualization

Seaborn helps to visualize the statistical relationships, To understand how variables in a dataset are related to one another and how that relationship is dependent on other variables, we perform statistical analysis. This Statistical analysis helps to visualize the trends and identify various patterns in the dataset.

These are the plot will help to visualize:

- Line Plot
- Scatter Plot
- Box plot
- Point plot
- Count plot
- Violin plot
- Swarm plot
- Bar plot
- KDE Plot

Line plot:

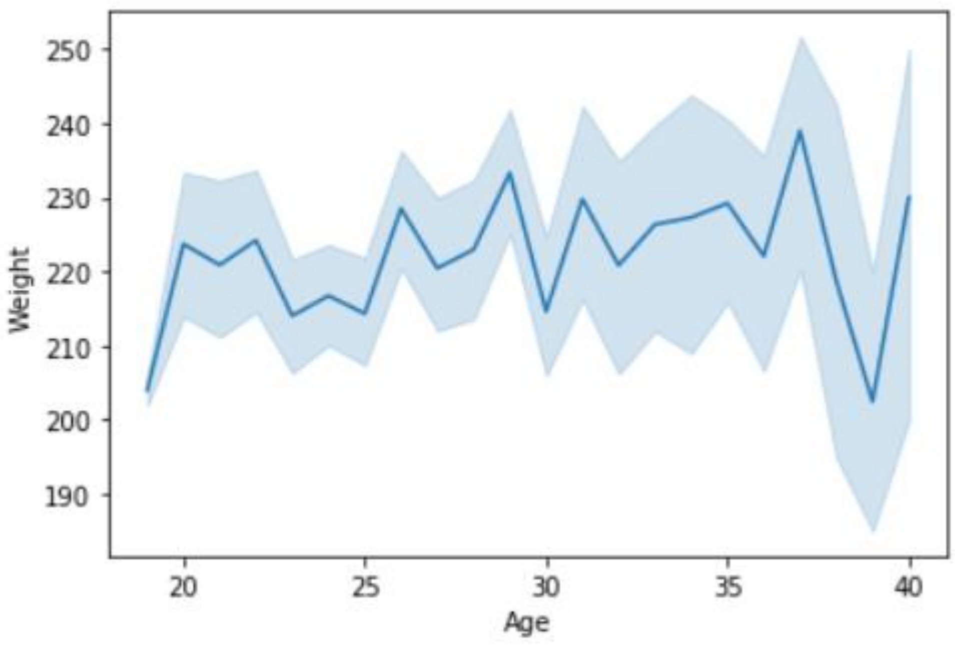
Lineplot Is the most popular plot to draw a relationship between x and y with the possibility of several semantic groupings.
Syntax : *sns.lineplot(x=None, y=None)*
Parameters:
x, y: *Input data variables; must be numeric. Can pass data directly or reference columns in data.*
Let's visualize the data with a line plot and pandas:
Example 1:

- Python3

```
# import module

import seaborn as sns

import pandas



# loading csv

data = pandas.read_csv("nba.csv")
```

```
# plotting lineplot

sns.lineplot( data['Age'], data['Weight'])
```

Output:



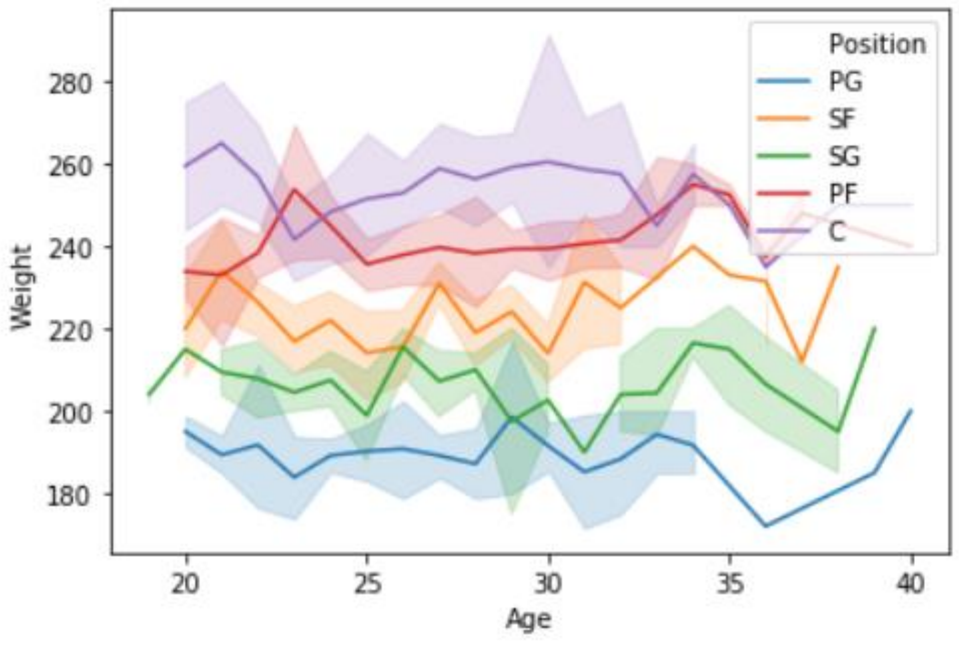
Example 2: Use the hue parameter for plotting the graph.

- Python3

```
# import module

import seaborn as sns

import pandas



# read the csv data

data = pandas.read_csv("nba.csv")



# plot
```

```
sns.lineplot(data['Age'],data['Weight'], hue =data["Position"])
```

Output:



Scatter Plot:

Scatterplot Can be used with several semantic groupings which can help to understand well in a graph against continuous/categorical data. It can draw a two-dimensional graph.

Syntax: *seaborn.scatterplot(x=None, y=None)*

Parameters:

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

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

Let's visualize the data with a scatter plot and pandas:

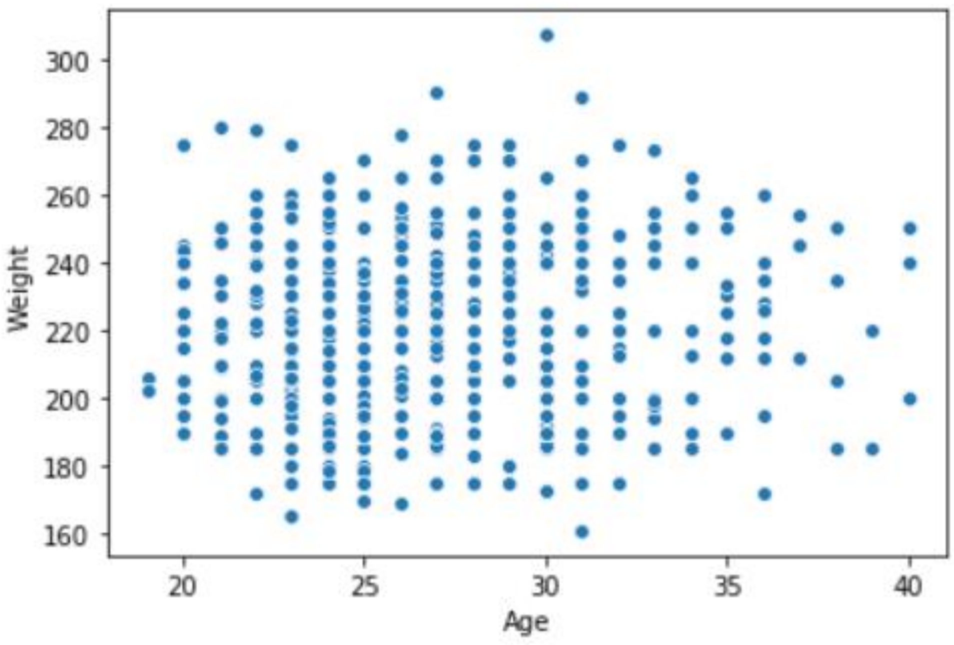
Example 1:

- Python3

```
# import module

import seaborn

import pandas
```

```
# load csv

data = pandas.read_csv("nba.csv")



# plotting

seaborn.scatterplot(data['Age'],data['Weight'])
```

Output:

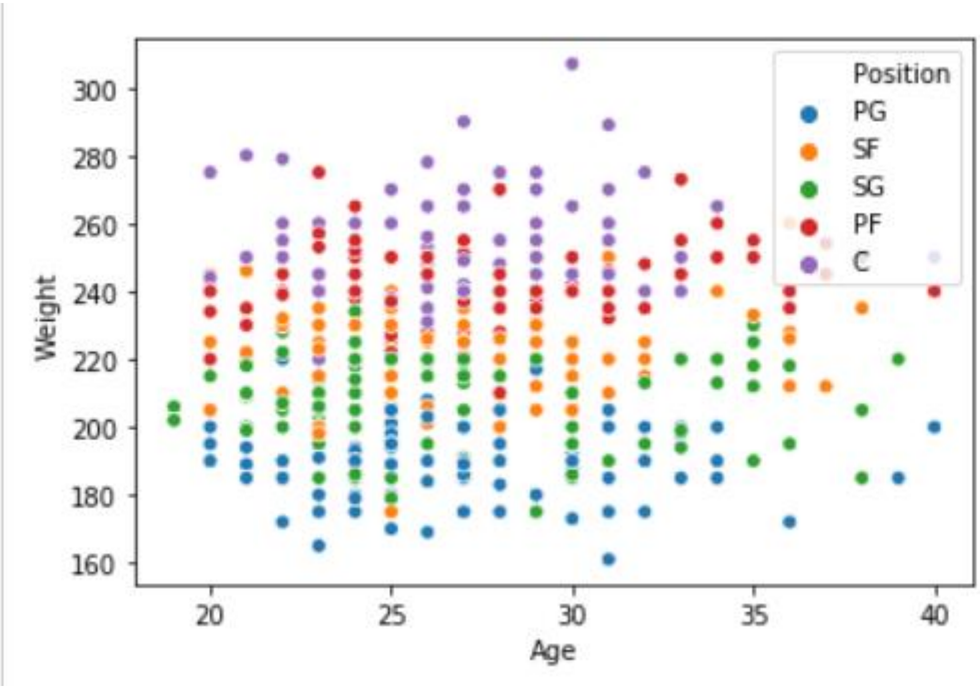


Example 2: Use the hue parameter for plotting the graph.

- Python3

```
import seaborn

import pandas

data = pandas.read_csv("nba.csv")



seaborn.scatterplot(     data['Age'],     data['Weight'],     hue
=data["Position"])
```

Output:



Box plot:

A box plot (or box-and-whisker plot) s is the visual representation of the depicting groups of numerical data through their quartiles against continuous/categorical data.
A box plot consists of 5 things.

- Minimum
- First Quartile or 25%
- Median (Second Quartile) or 50%
- Third Quartile or 75%
- Maximum

Syntax:
seaborn.boxplot(x=None, y=None, hue=None, data=None)

Parameters:

- ***x, y, hue:*** *Inputs for plotting long-form data.*
- ***data:*** *Dataset for plotting. If x and y are absent, this is interpreted as wide-form.*

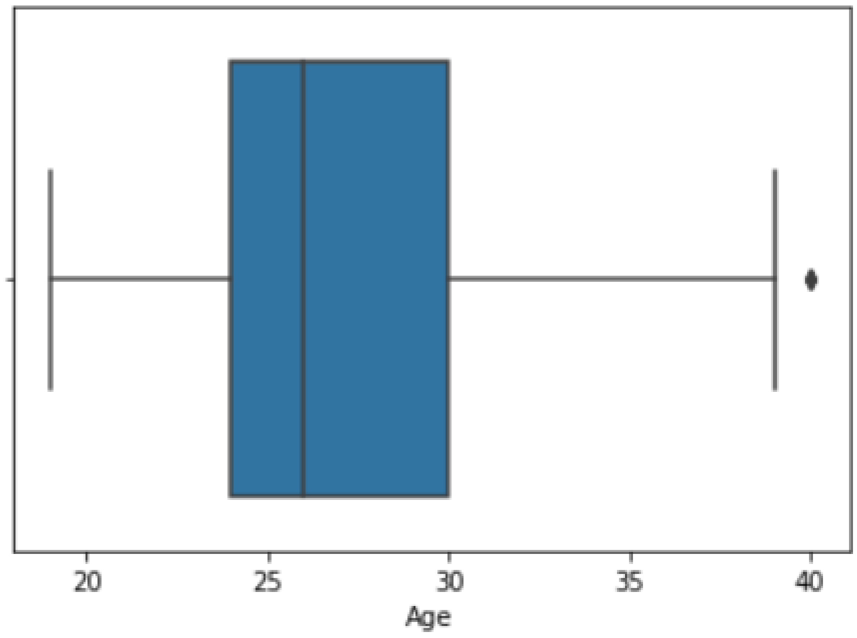
Returns: *It returns the Axes object with the plot drawn onto it.*
Draw the box plot with Pandas:
Example 1:

- Python3

```
# import module
```

```
import seaborn as sns

import pandas



# read csv and plotting

data = pandas.read_csv( "nba.csv" )

sns.boxplot( data['Age'] )
```

Output:



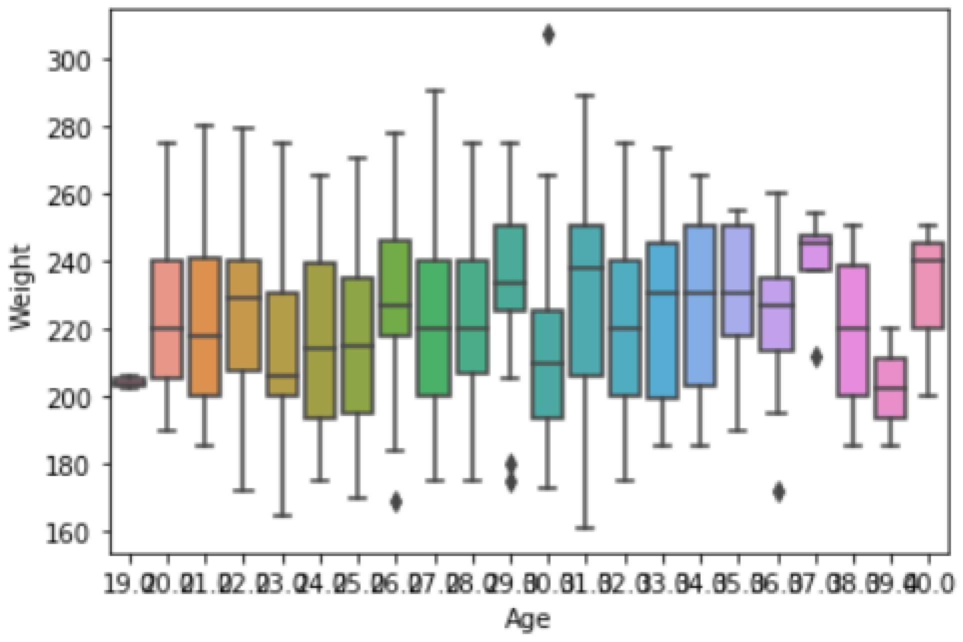
Example 2:

- Python3

```
# import module

import seaborn as sns

import pandas
```

```
# read csv and plotting

data = pandas.read_csv( "nba.csv" )

sns.boxplot( data['Age'], data['Weight'])
```

Output:



Violin Plot:

A violin plot is similar to a boxplot. It shows several quantitative data across one or more categorical variables such that those distributions can be compared.

Syntax: *seaborn.violinplot(x=None, y=None, hue=None, data=None)*

Parameters:

- ***x, y, hue:*** *Inputs for plotting long-form data.*
- ***data:*** *Dataset for plotting.*

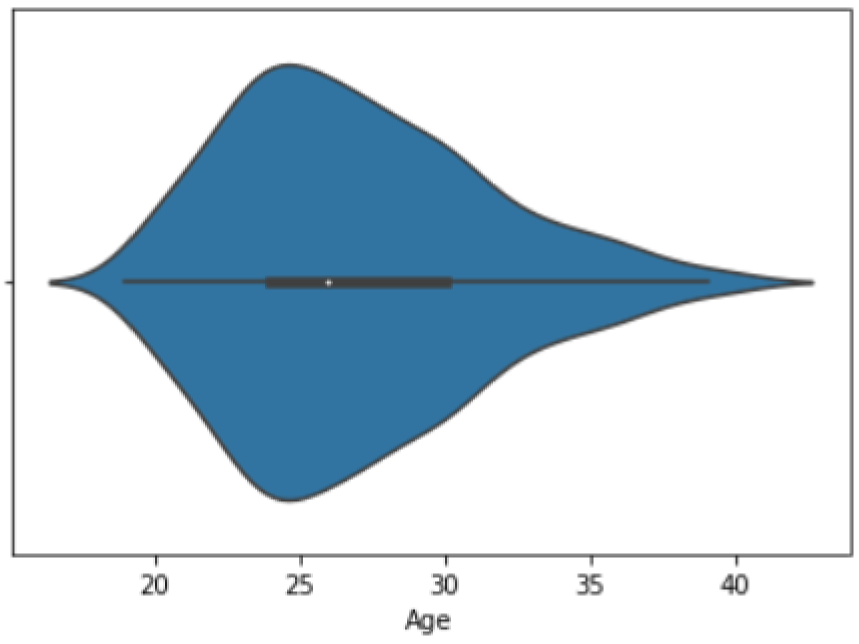
Draw the violin plot with Pandas:

Example 1:

- Python3

```
# import module

import seaborn as sns
import pandas



# read csv and plot

data = pandas.read_csv("nba.csv")

sns.violinplot(data['Age'])
```

Output:



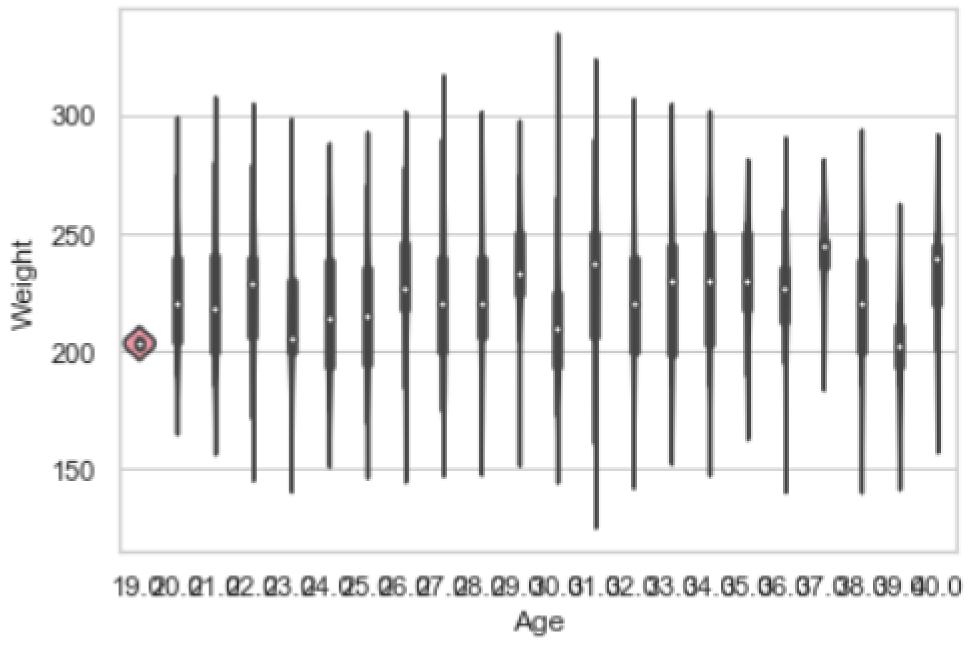
Example 2:

- Python3

```
# import module

import seaborn



seaborn.set(style = 'whitegrid')
```

```
# read csv and plot

data = pandas.read_csv("nba.csv")

seaborn.violinplot(x ="Age", y ="Weight",data = data)
```

Output:



Swarm plot:

A swarm plot is similar to a strip plot, We can draw a swarm plot with non-overlapping points against categorical data.

Syntax: *seaborn.swarmplot(x=None, y=None, hue=None, data=None)*

Parameters:

- ***x, y, hue:*** *Inputs for plotting long-form data.*
- ***data:*** *Dataset for plotting.*

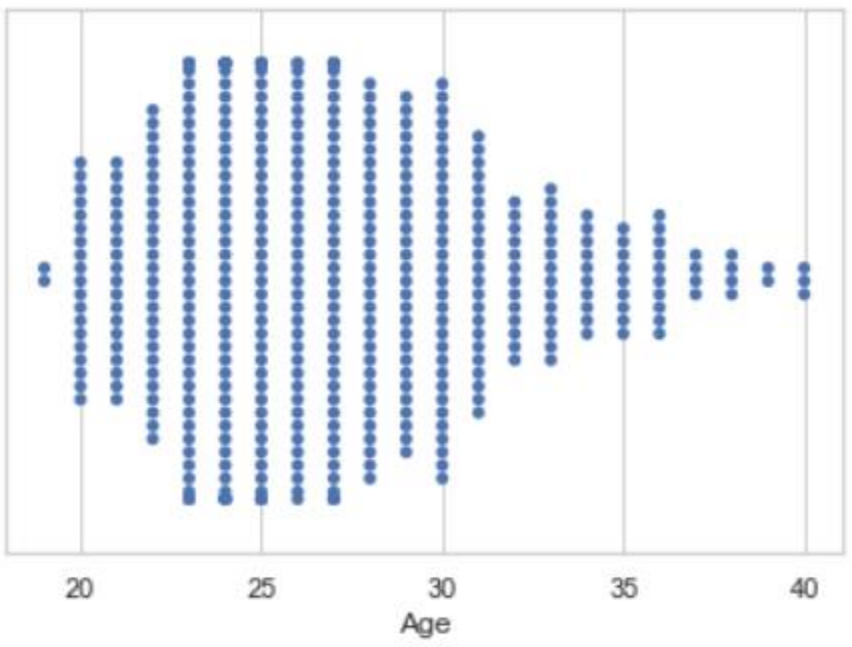
Draw the swarm plot with Pandas:
Example 1:

- Python3

```
# import module
```

```
import seaborn



seaborn.set(style = 'whitegrid')



# read csv and plot

data = pandas.read_csv( "nba.csv" )

seaborn.swarmplot(x = data["Age"])
```

Output:



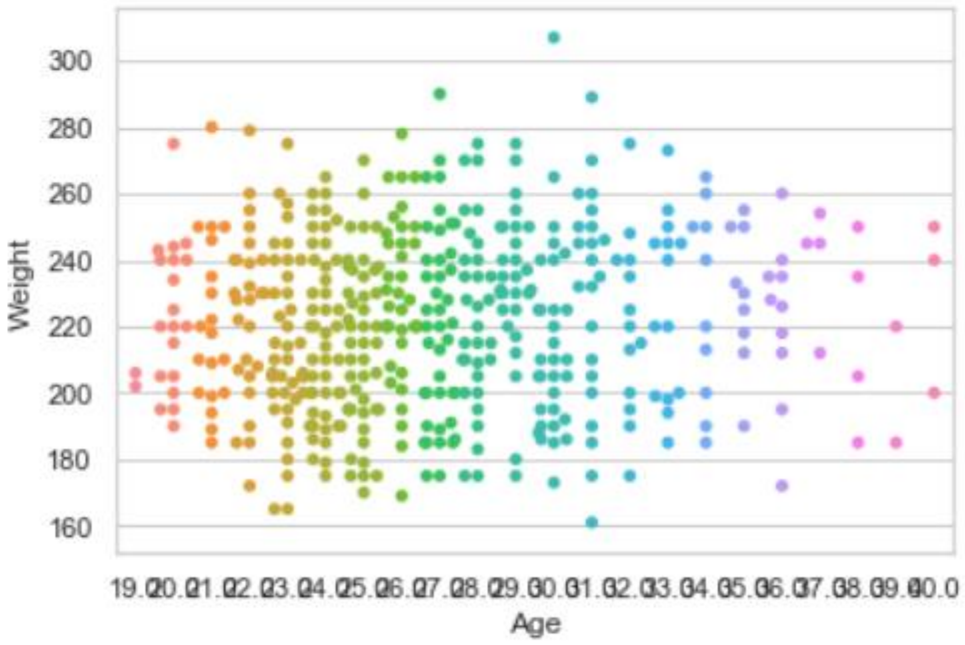
Example 2:

- Python3

```
# import module

import seaborn
```

```
seaborn.set(style = 'whitegrid')



# read csv and plot

data = pandas.read_csv("nba.csv")

seaborn.swarmplot(x ="Age", y ="Weight",data = data)
```

Output:



Bar plot:

Barplot 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.

Syntax : *seaborn.barplot(x=None, y=None, hue=None, data=None)*

Parameters :

- ***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.*

Returns : *Returns the Axes object with the plot drawn onto it.*

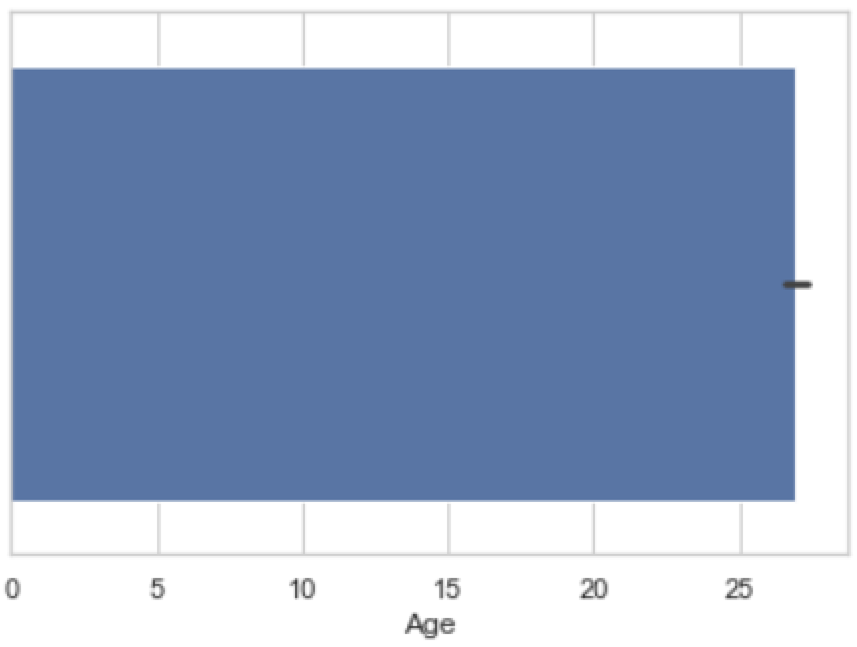
Draw the bar plot with Pandas:

Example 1:

- Python3

```
# import module
import seaborn



seaborn.set(style = 'whitegrid')



# read csv and plot
data = pandas.read_csv("nba.csv")
seaborn.barplot(x =data["Age"])
```

Output:

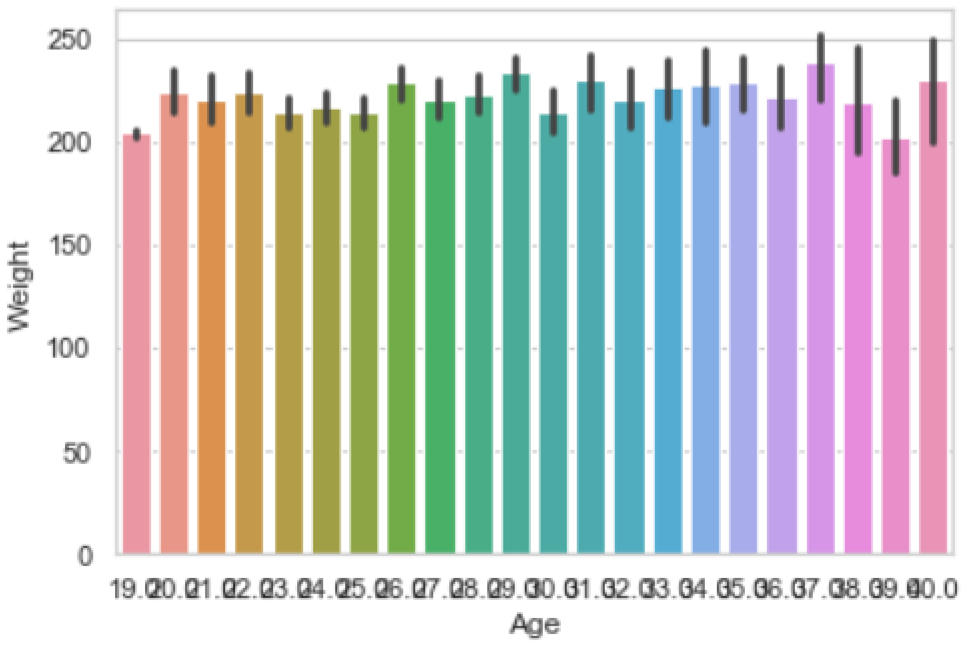


Example 2:

- Python3

```
# import module

import seaborn



seaborn.set(style = 'whitegrid')



# read csv and plot

data = pandas.read_csv("nba.csv")

seaborn.barplot(x ="Age", y ="Weight", data = data)
```

Output:



Point plot:

Point plot used to show point estimates and confidence intervals using scatter plot glyphs. A point plot represents an estimate of central tendency for a numeric variable by the position of scatter plot points and provides some indication of the uncertainty around that estimate using error bars.

Syntax: *seaborn.pointplot(x=None, y=None, hue=None, data=None)*

Parameters:

- ***x, y:*** *Inputs for plotting long-form data.*

- ***hue:*** *(optional) column name for color encoding.*
- ***data:*** *dataframe as a Dataset for plotting.*

Return: *The Axes object with the plot drawn onto it.*

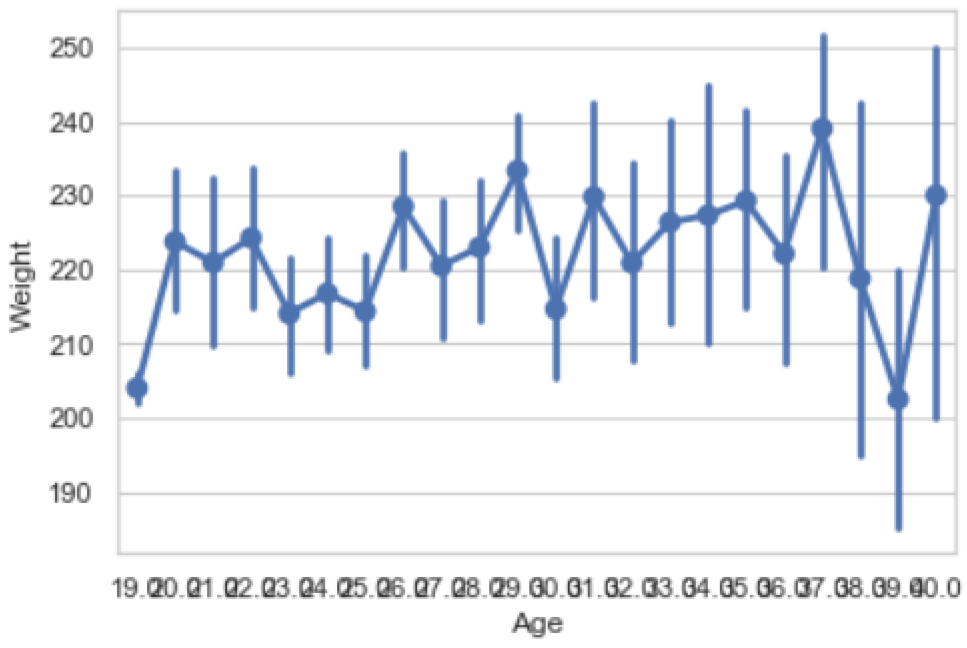
Draw the point plot with Pandas:

Example:

- Python3

```
# import module
import seaborn

seaborn.set(style = 'whitegrid')

# read csv and plot
data = pandas.read_csv("nba.csv")
seaborn.pointplot(x = "Age", y = "Weight", data = data)
```

Output:



Count plot:

Count plot used to Show the counts of observations in each categorical bin using bars.

Syntax : *seaborn.countplot(x=None, y=None, hue=None, data=None)*
Parameters :

- ***x, y:*** *This parameter take names of variables in data or vector data, optional, Inputs for plotting long-form data.*
- ***hue :*** *(optional) This parameter take column name for color 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.*

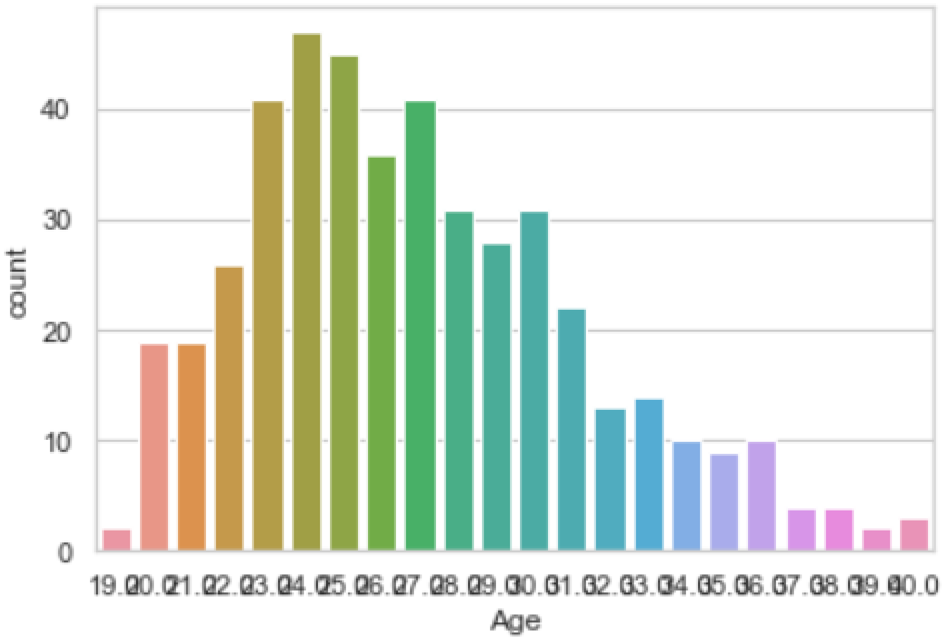
Returns: *Returns the Axes object with the plot drawn onto it.*

Draw the count plot with Pandas:
Example:

- Python3

```
# import module

import seaborn



seaborn.set(style = 'whitegrid')



# read csv and plot

data = pandas.read_csv("nba.csv")

seaborn.countplot(data["Age"])
```

Output:



KDE Plot:

KDE Plot described as **Kernel Density Estimate** is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization.

Syntax: *seaborn.kdeplot(x=None, *, y=None, vertical=False, palette=None, **kwargs)*

Parameters:

x, y : *vectors or keys in data*

vertical : *boolean (True or False)*

data : *pandas.DataFrame, numpy.ndarray, mapping, or sequence*

Draw the KDE plot with Pandas:

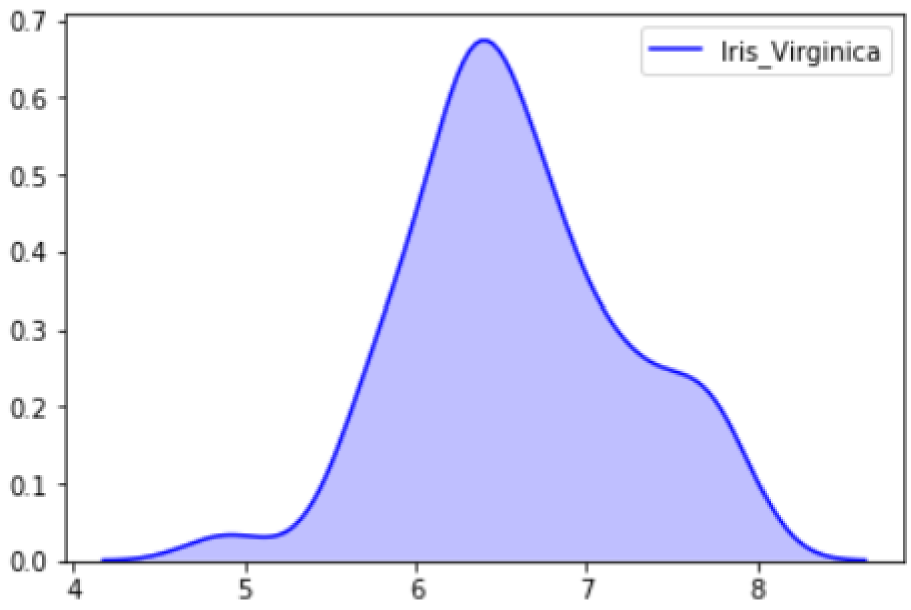
Example 1:

- Python3

```
# importing the required libraries

from sklearn import datasets

import pandas as pd

import seaborn as sns
```

```
# Setting up the Data Frame

iris = datasets.load_iris()



iris_df = pd.DataFrame(iris.data, columns=['Sepal_Length',

                      'Sepal_Width', 'Patal_Length', 'Petal_Width'])



iris_df['Target'] = iris.target



iris_df['Target'].replace([0], 'Iris_Setosa', inplace=True)

iris_df['Target'].replace([1], 'Iris_Vercicolor', inplace=True)

iris_df['Target'].replace([2], 'Iris_Virginica', inplace=True)



# Plotting the KDE Plot

sns.kdeplot(iris_df.loc[(iris_df['Target'] =='Iris_Virginica'),

            'Sepal_Length'],  color  =  'b',  shade  =  True,  Label
='Iris_Virginica')
```

Output:

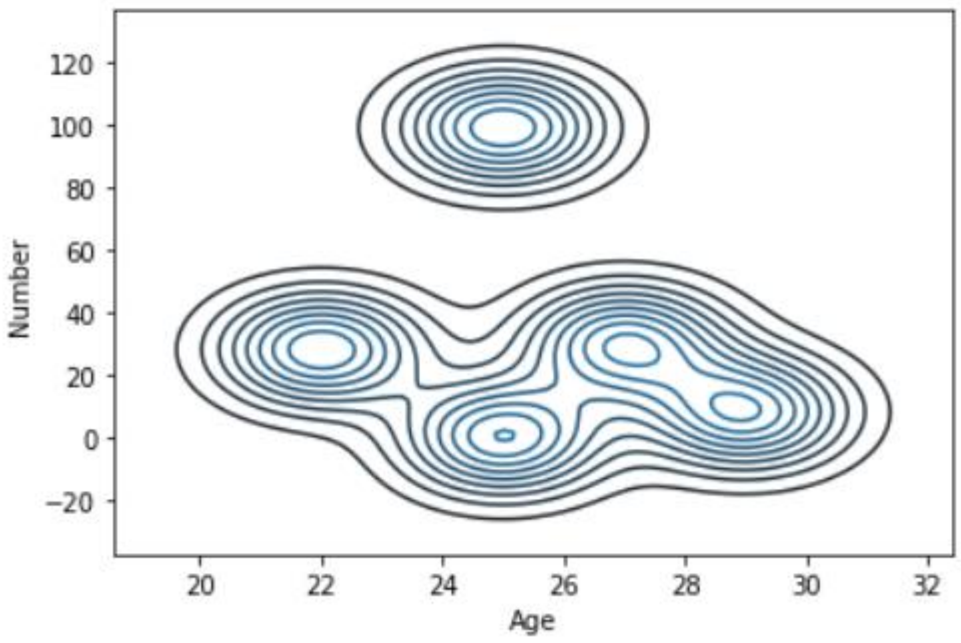


Example 2:

- Python3

```
# import module

import seaborn as sns

import pandas



# read top 5 column

data = pandas.read_csv("nba.csv").head()



sns.kdeplot( data['Age'], data['Number'])
```

Output:



Bivariate and Univariate data using seaborn and pandas:

Before starting let’s have a small intro of bivariate and univariate data:

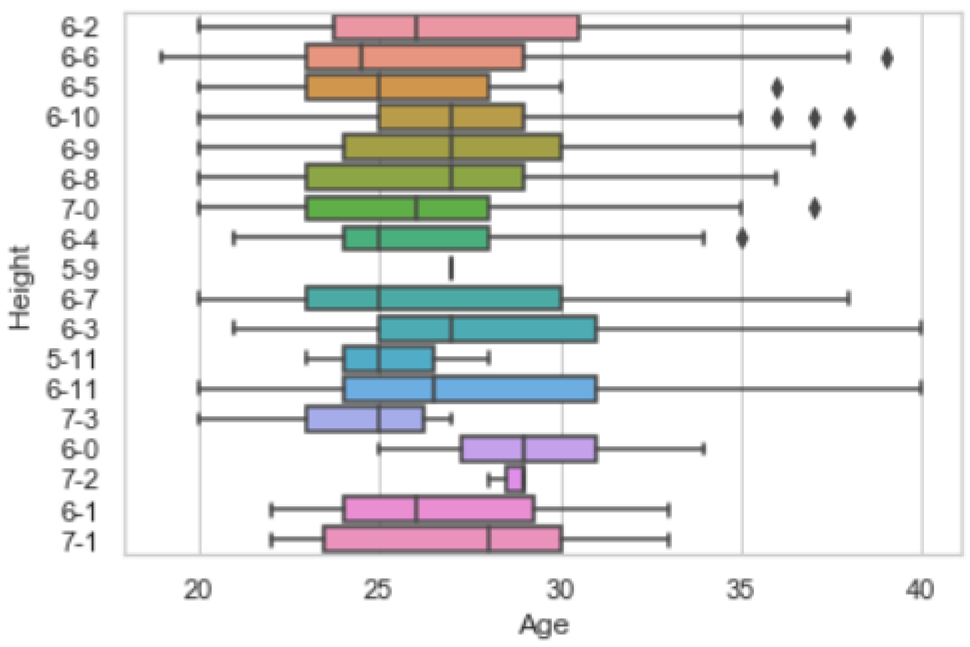
Bivariate data: This type of data involves **two different variables**. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship between the two variables.
Univariate data: This type of data consists of **only one variable**. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it.
Let’s see an example of Bivariate data :
Example 1: Using the box plot.

- Python3

```
# import module

import seaborn as sns

import pandas



# read csv and plotting
```

```
data = pandas.read_csv( "nba.csv" )

sns.boxplot( data['Age'], data['Height'])
```

Output:

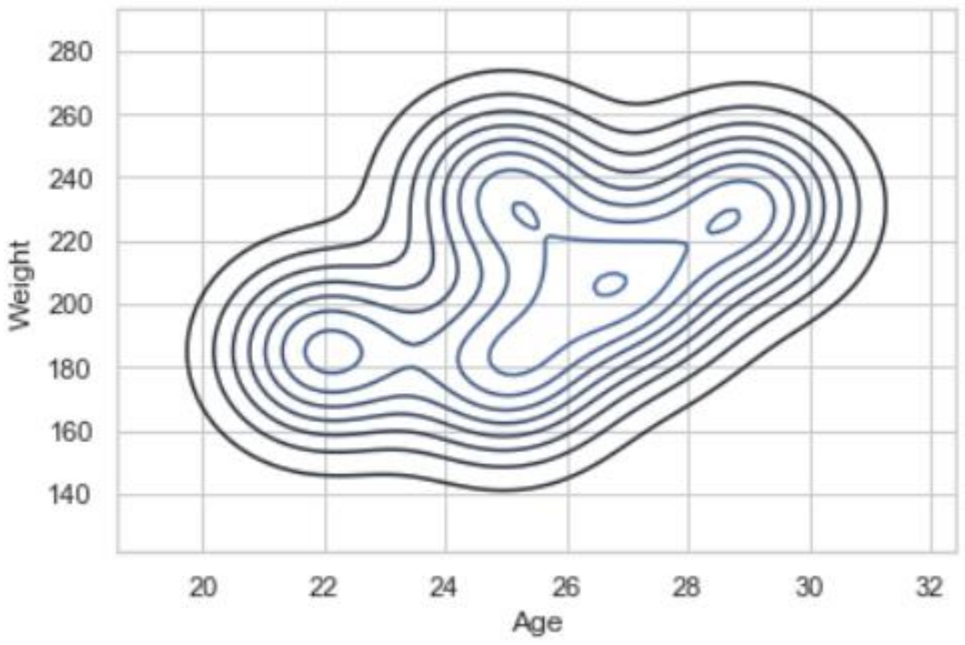


Example 2: using KDE plot.

- Python3

```
# import module

import seaborn as sns

import pandas



# read top 5 column

data = pandas.read_csv("nba.csv").head()



sns.kdeplot( data['Age'], data['Weight'])
```

Output:



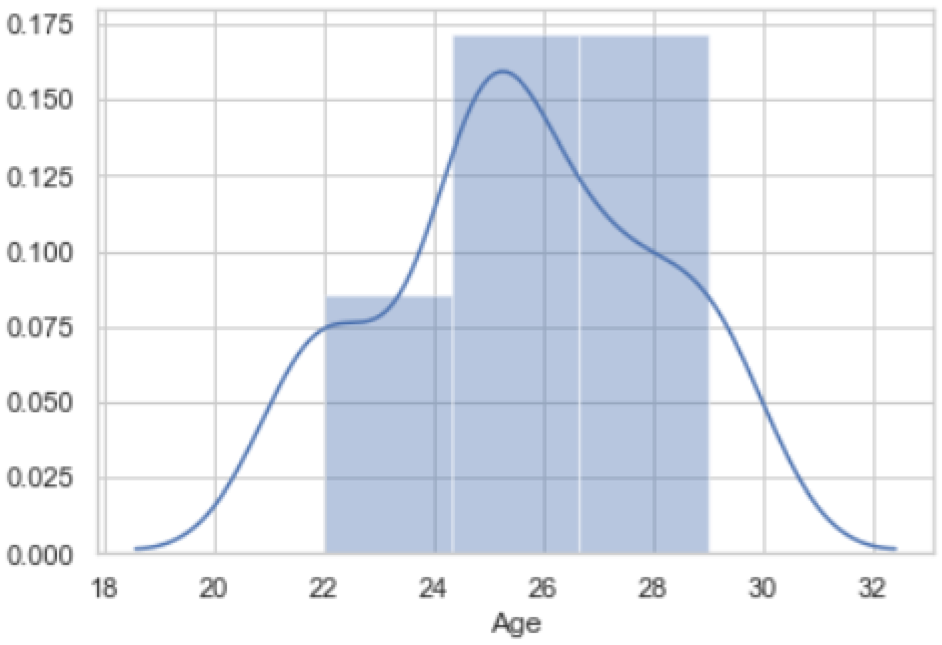
Let’s see an example of univariate data distribution:

Example: Using the dist plot

- Python3

```
# import module

import seaborn as sns

import pandas



# read top 5 column

data = pandas.read_csv("nba.csv").head()



sns.distplot( data['Age'])
```

Output:


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
18
20
22
24
26
28
30
32
Age