**Experiment No.2**

**Aim:** Data Visualization / Exploratory Data Analysis for the selected data set using Matplotlib and Seaborn.

**Prerequisites**: python.

**Objectives:** - At the end of this experiment, you will be able to:

• Use Matplotlib for data visualization

**• Use Seaborn for data visualization**

**Theory:**

**Matplotlib** is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python interpreter and IPython shell, the jupyter notebook, web application servers, and graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. Although Matplotlib is written primarily in pure Python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays.

#%matplotlib notebook

#: will lead to interactive plots embedded within the notebook,

# you can zoom and resize the figure

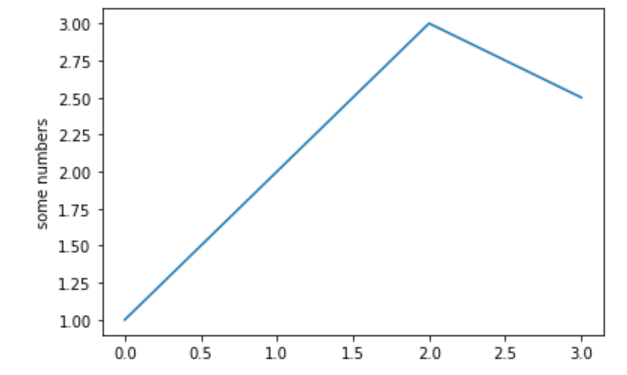
%matplotlib inline

#: only draw static images in the notebook

import matplotlib.pyplot as plt  # like we import numpy as np , similarly we import matplotlib.pyplot as plt

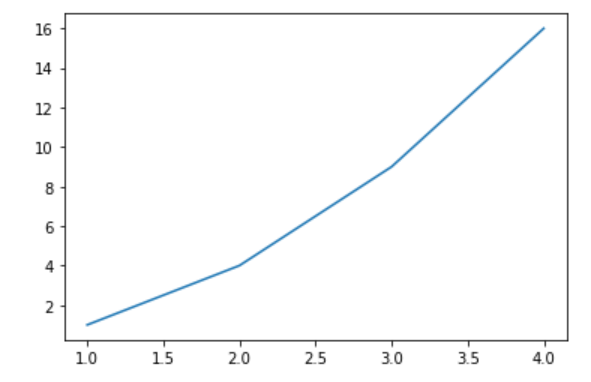
plt.plot([1,2,3,2.5])  # we are plotting some y-values for default x-values 1,2,3,.....

plt.ylabel('some numbers');  # Try putting a ; at the end of this last statement



plot() is a versatile command, and will take an arbitrary number of arguments. For example, to plot x versus y, you can issue the command:

plt.plot([1, 2, 3, 4], [1, 4, 9, 16]);  # first parameter is x, second is y



Using the pyplot interface, you build a graph by calling a sequence of functions and all of them are applied to the current subplot, like so:

plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3) # makes a line plot

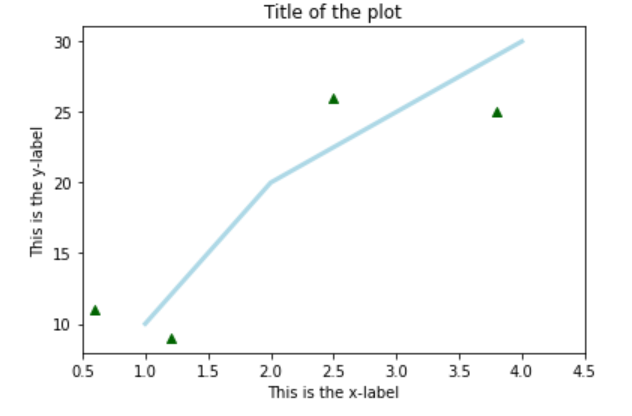
plt.scatter([0.6, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^')  # makes a scatter plot

plt.xlim(0.5, 4.5)  # limiting the range of the X-axis from min = 0.5 to max = 4.5

plt.title("Title of the plot")  # assigning a title to the plot.

plt.xlabel("This is the x-label")

plt.ylabel("This is the y-label");



When working with just one subplot in the figure, generally is OK to work with the pyplot interface, however, when doing more complicated plots, or working within larger scripts, you will want to explicitly pass around the Subplot (Axes) and/or Figure object to operate upon.

import numpy as np

# we are defining a function to produce some numbers, given a value of t

def f(t):

    return np.exp(-t) \* np.cos(2\*np.pi\*t)

# t1 is range of numbers from 0 to 5 , incrementing by 0.1

# code t1 here

t1=np.arange(0.0,5.0,0.1)

# t2 is range of numbers from 0 to 5 , incrementing by 0.02

# code t2 here

t2=np.arange(0.0,5.0,0.02)

plt.figure()

plt.subplot(2, 1, 1) # no of subplot,col,row

plt.plot(t1, f(t1), 'bo')  # Note : x values are t1 and y values are f(t1)

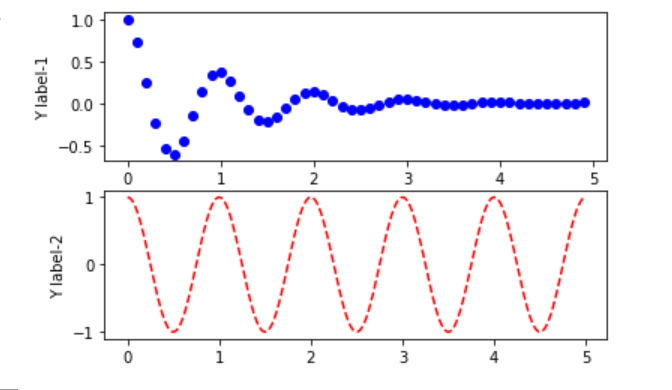
plt.ylabel("Y label-1");

plt.subplot(2, 1, 2)  # so 2,1,2 means 2 rows , 1 column and we want to plot in the 2nd cell.

plt.plot(t2, np.cos(2\*np.pi\*t2), 'r--')  # Note : x values are t2 and y values are np.cos(2\*np.pi\*t2)

plt.ylabel("Y label-2"); # Which subplot is modifying this function?

# Ans : the current one. i.e subplot (2, 1, 2)



What you do when plotting with the object oriented interface, is to create the objects and then call methods for every object to make changes specifically to that object. The plt.subplots function creates a figure object and an Subplot (Axes) object at the same time, then we can create a plot by calling methods on those objects.

x = np.linspace(-4,4)  # default num = 50 values

y1, y2 = x\*\*2, x\*\*3

# Object Oriented Interface

fig, ax = plt.subplots()

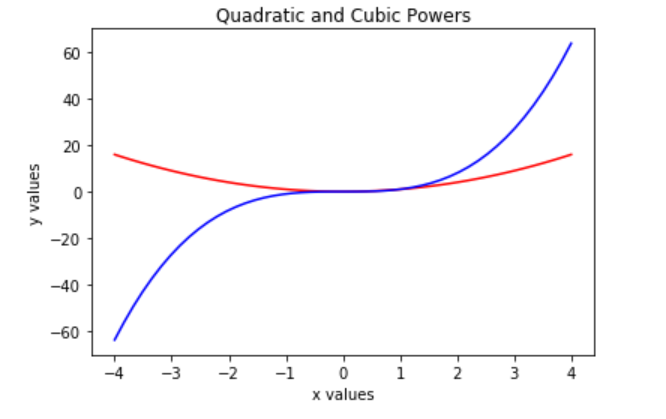
ax.plot(x, y1, 'red')

ax.plot(x, y2, 'blue')

ax.set\_title('Quadratic and Cubic Powers')

ax.set\_xlabel('x values')

ax.set\_ylabel('y values'); # put a ; to remove the unnecessary Text(...) part



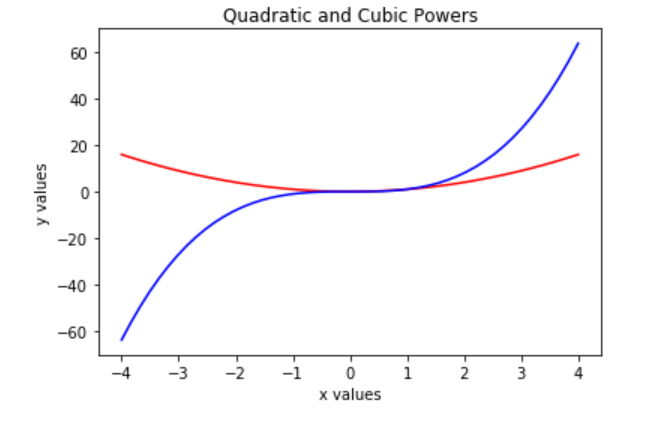
# pyplot

plt.plot(x, y1, 'red', x, y2, 'blue')

plt.title('Quadratic and Cubic Powers')

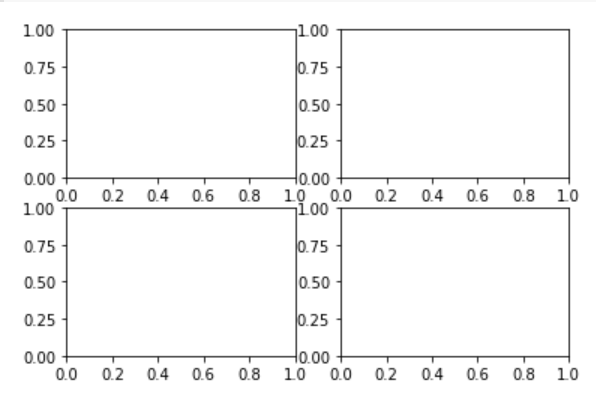
plt.xlabel('x values')

plt.ylabel('y values');



# create a figure object with with 4 subplots as 2 by 2

fig,axes=plt.subplots(nrows=2,ncols=2)



Example: Creating a figure showing the shape of the power functions.

x = np.linspace(start=-5, stop=5, num=150)

# All functions in one axes

fig, ax = plt.subplots(figsize = (7,4))

ax.plot(x, x, label='Linear')

ax.plot(x, x\*\*2,  label='Quadratic')

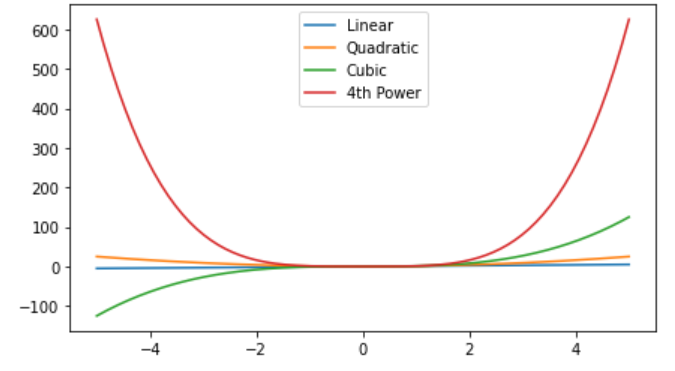
ax.plot(x, x\*\*3, label='Cubic')

ax.plot(x, x\*\*4, label='4th Power')

ax.legend();

# Notice : linear and Quadratic are hardly seen in their real shapes.

# becoz they are being overpowered by higher order equations



Show 4 Axes/Subplots: One function in one axes(for all 4 function)

Code:

fig, axes = plt.subplots(nrows=2, ncols=2, figsize = (7,7))

axes[0,0].set\_title('Linear')

axes[0,0].plot(x,x)

## more code here

axes[0,1].set\_title('Quadratic')

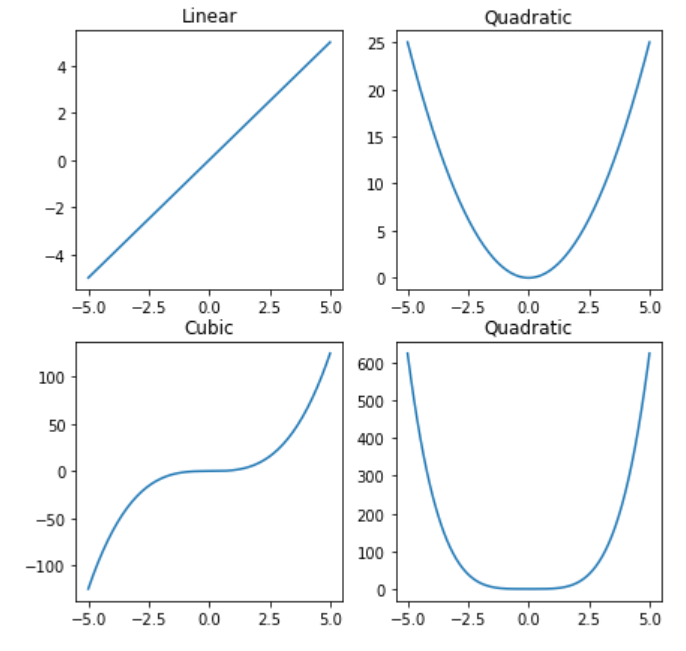
axes[0,1].plot(x,x\*\*2)

axes[1,0].set\_title('Cubic')

axes[1,0].plot(x,x\*\*3)

axes[1,1].set\_title('Quadratic')

axes[1,1].plot(x,x\*\*4)



## Bar plots

A bar chart or bar graph is a chart or graph that presents data with rectangular bars with lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

This type of graph can also be used to show comparisons among categories. One axis of the chart shows the specific categories being compared, and the other axis represents a discrete value. Some bar graphs present bars clustered in groups of more than one.

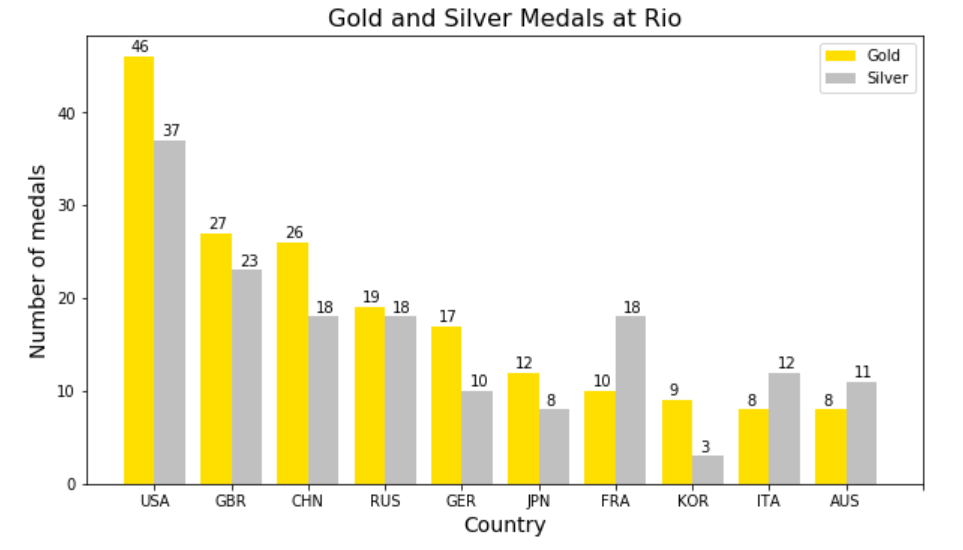
Top 10 countries in the Rio Olympics

countries = ['USA','GBR','CHN','RUS','GER','JPN','FRA','KOR','ITA','AUS']

gold = [46,27,26,19,17,12,10,9,8,8]

silver = [37,23,18,18,10,8,18,3,12,11]

bronze = [38,17,26,19,15,21,14,9,8,10]



fig, ax = plt.subplots(figsize=(10,5.5))

ax.bar(np.arange(10), gold, color="#FFDF00", width=0.4, label='Gold')

ax.bar(np.arange(10)+0.4, silver, color="#C0C0C0", width=0.4, label='Silver')

ax.set\_xticks(np.arange(0.2,10.4, 1))

ax.set\_xticklabels(countries);

for x,g,s in zip(np.arange(10), gold, silver):

    ax.text(x-0.1, g+0.5, g) # annotating the golds

    ax.text(x+0.31, s+0.5, s); # annotating the silvers

# set title , xlabel , ylabel and show legends. learner codes by self.

ax.set\_title('GGold and silver medial at Rio',size=15)

ax.set\_xlabel('country',size=14)

ax.set\_ylabel('number of medal',size=14)

ax.legend(loc='upper right');

## Histograms

A histogram is a graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable). It is a kind of bar graph. To construct a histogram, the first step is to "bin" the range of values, that is, divide the entire range of values into a series of intervals, and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent, and are often (but are not required to be) of equal size.

Histograms give a rough sense of the density of the underlying distribution of the data.

iqs = np.random.normal(loc=100, scale=10, size=300)

# np.random.normal produces a normal distribution around 100 of 300 points

# scale = 10 means standard deviation

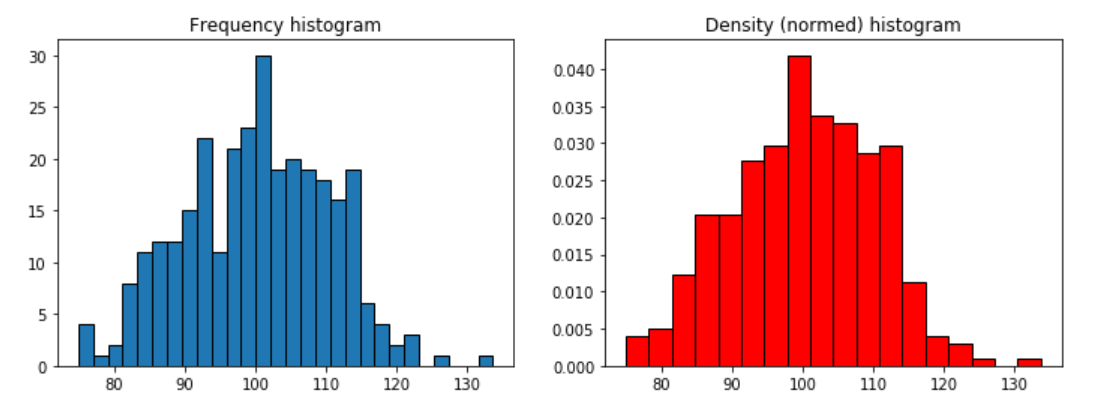
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12,4))

ax[0].hist(iqs, bins=28, edgecolor='k')

ax[0].set\_title('Frequency histogram')

ax[1].hist(iqs, bins=18, color = 'red', edgecolor='k', density=True)

ax[1].set\_title('Density (normed) histogram');



## Scatter plot

A scatter plot is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. If the points are color-coded, one additional variable can be displayed. The data is displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.

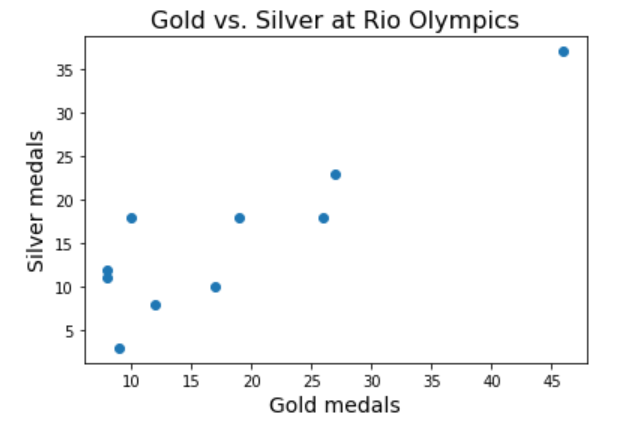
fig, ax = plt.subplots()

ax.scatter(gold, silver, marker='o')

ax.set\_title('Gold vs. Silver at Rio Olympics', size=16)

ax.set\_xlabel('Gold medals', size=14)

ax.set\_ylabel('Silver medals', size=14);



**Creating Contingency Table** Now we create a contingency table for the column showing petal width for each species. For this we use the crosstab function available in pandas and give these tow column’s names as inputs

we take the iris flower data set for analysis. This data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. We will create contingency model on these features which will be ultimately used in distinguishing the species from each other.

import numpy as np

import pandas as pd

datainput = pd.read\_csv("iris.csv")

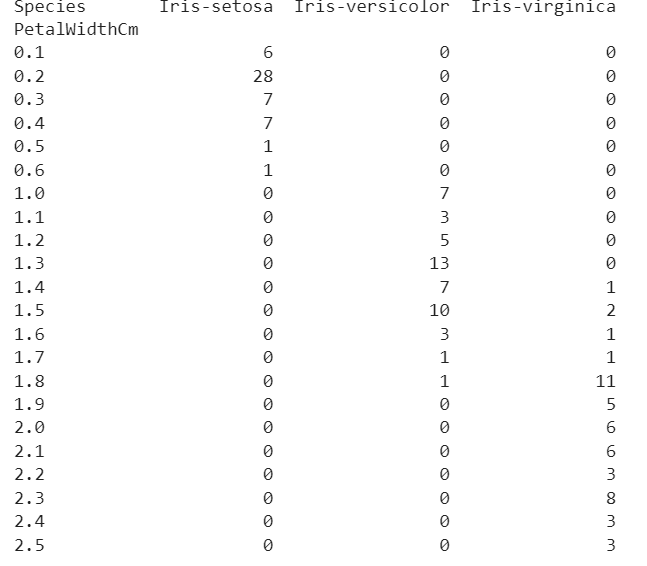
print (datainput.head(5))

datainput = pd.read\_csv("iris.csv")

print(datainput.dtypes)

width\_species = pd.crosstab(datainput['PetalWidthCm'],datainput['Species'],margins = False)

print(width\_species)



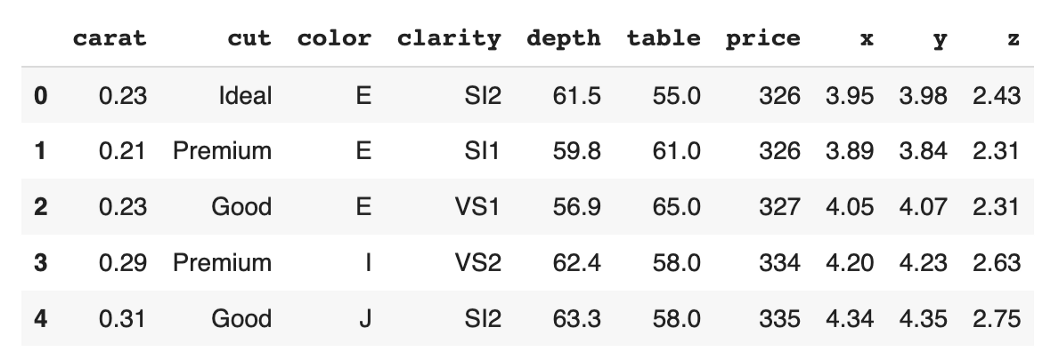
**SEABORN**

import matplotlib.pyplot as plt  
import numpy as np  
import seaborn as sns  
import pandas as pd

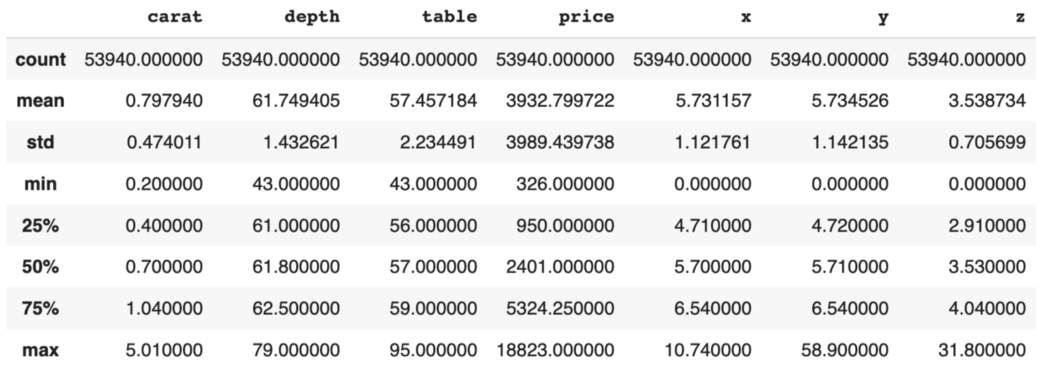
The above code imports Matplotlib, Numpy and seaborn libraries. Matplotlib and Seaborn are used to make graphs, charts and helps to visualise data. NumPy is used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. Pandas is used to analyse data and it has many functionalities to play with data. We’ll see more in my upcoming blogs. ‘diamonds’ is one among the inbuilt datasets and we will be using this dataset.

*print(sns.get\_dataset\_names()) #prints the inbuilt dataset names in Seaborn*

#loading dataset  
df = sns.load\_dataset("diamonds") #loads dataset into dataframe 'df'  
df.head()



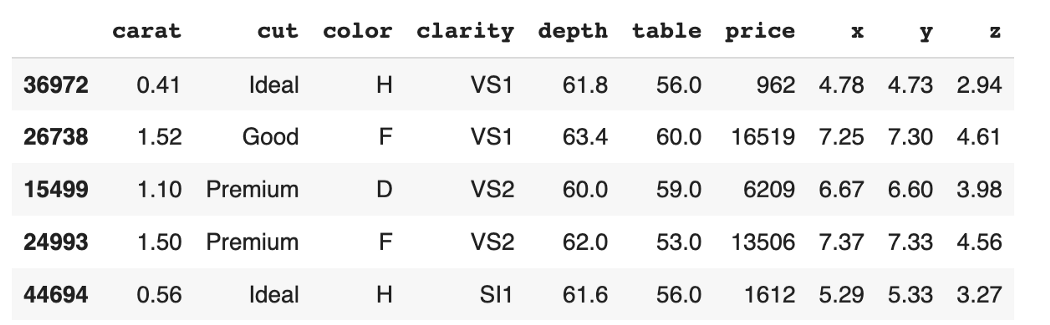
df.shape #returns no of rows and columns in form of tuple(53940, 10)  
df.describe()



df.describe()

As we have many records(rows), we will randomly take rows and analyse on it. Sample function will randomly selects records in the dataset. In below code ***.sample(3000)***means selecting 3000 rows at random in dataset.

sample = df.sample(3000)  
sample.head()



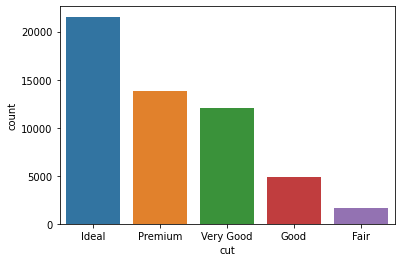
3000 samples are taken at random

In Seaborn library we have many different functions which are simple and returns data in visualised format. The below visualisations may differ if you try because, when you use sample function you may get different set of rows. Let’s see each of the functions and how it results.

*COUNTPLOT*

This function helps to count the no of type elements in a certain column. In this code we are using our main dataframe i.e, **df**. Let’s see a working example code below.

sns.countplot(data=df, x='cut')



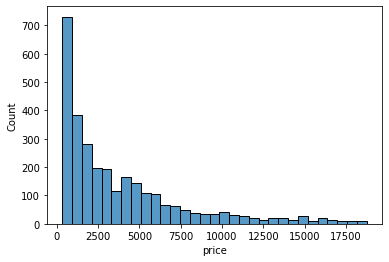
sns.countplot

*HISTPLOT*

Histplot function helps us to make histogram and the basic parameters to be passed are the dataframe and column for x-axis. Let’s see a working example code below.

sns.histplot(data=sample, x="price")

In this histogram, we can depict that most of our dataset has price in the range 0 to 2500.

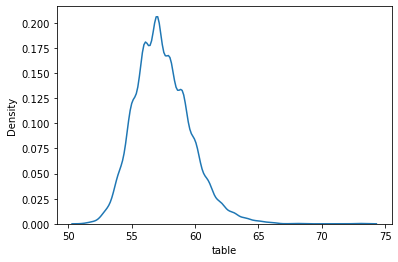


sns.histplot

*KDEPLOT*

A kernel density estimate (KDE) plot is a method for visualizing the distribution of observations in a dataset. Let’s see a working example code below.

sns.kdeplot(data=sample, x="table")

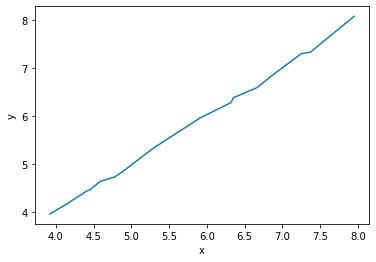


sns.kdeplot

*LINEPLOT*

This function helps to draw a line plot with possibility of several semantic groupings. The relationship between x and y can be shown for different subsets of the data using the hue, size, and style parameters. Let’s see a working example code below.

sns.lineplot(data=sample[:20], x='x',y='y')

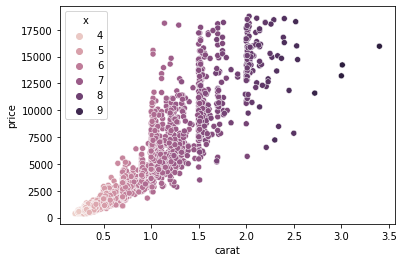


sns.lineplot

*SCATTERPLOT*

This function helps to make a diagram where each value in the data set is represented by a dot. Let’s see a working example code below.

sns.scatterplot(data=sample[:], x='carat', y='price', hue='x')

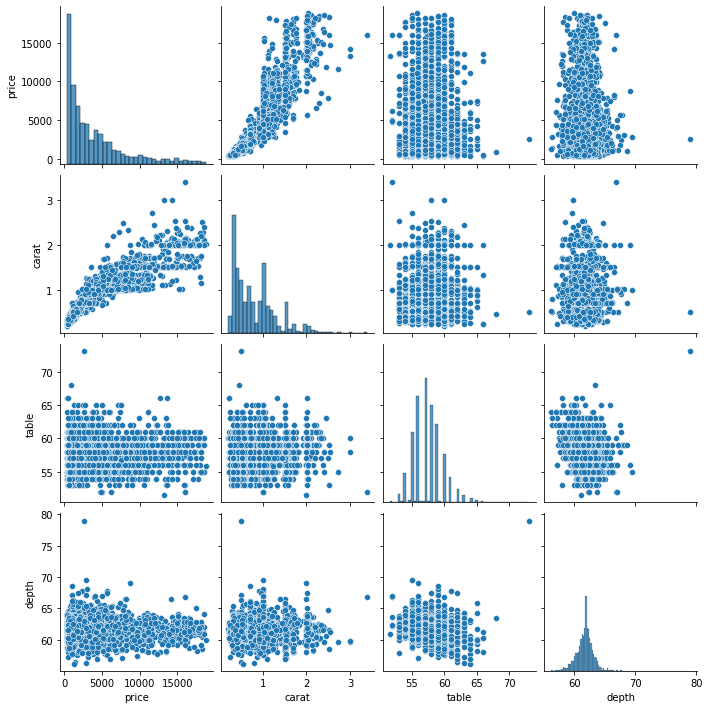


sns.scatterplot

*PAIRPLOT*

This function maps each of the column attribute and produces output which automatically takes required map style. Let’s see a working example code below.

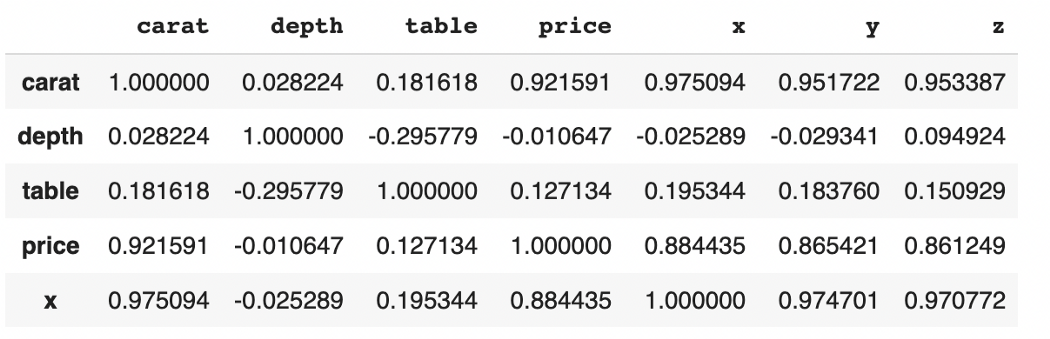
sns.pairplot(sample[["price", "carat", "table", "depth"]])



sns.pairplot

Correlation: *The statistical concept of correlation describes how closely two variables move in tandem with one another. Two variables are considered to have a positive correlation if they move in the same direction. They have a negative correlation if they travel in the opposite directions.*

#Absolute value of correlation  
#[0-0.3] as weak  
#[0.3-0.7] as moderate  
#[0.7-1.0] as strong correlationcorr\_matrix = df.corr()  
corr\_matrix.head()



*HEATMAP*

A representation of rectangular data presented as a matrix with colours is called a heatmap. It accepts a 2D dataset as a parameter. This kind of data visualisation is excellent as it can display the relationship between variables, including time. Let’s see a working example code below.

sns.heatmap(corr\_matrix, annot=True)

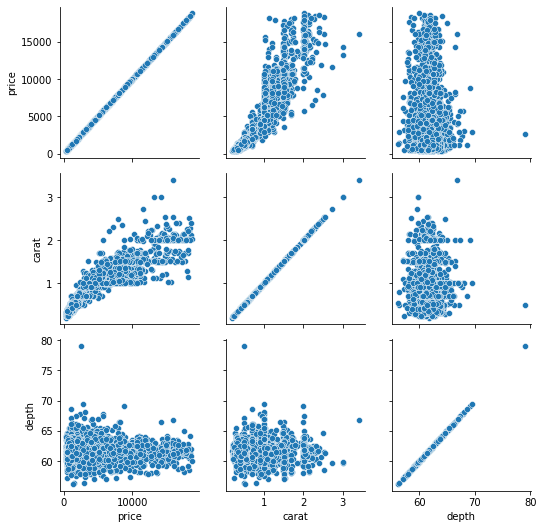


sns.heatmap

*PAIRPLOT*

In this function we can plot the data points by mentioning specific graph type and with multiple columns. Let’s see a working example code below.

g = sns.PairGrid(sample[["price", "carat", "depth"]])  
g.map(sns.scatterplot)

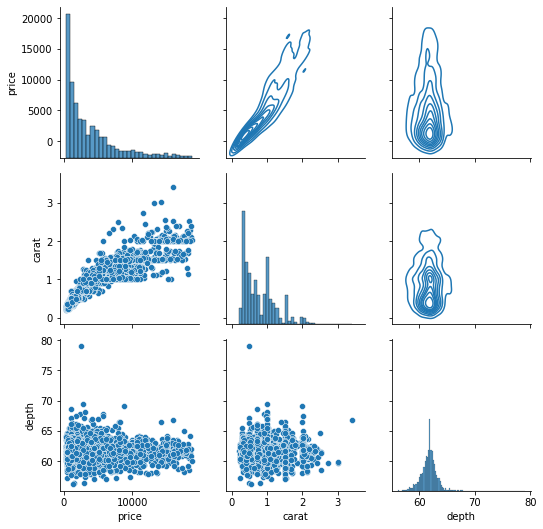


sns.PairGrid

We can also specify the position to make certain graph types to implement in those particular positions. Let’s see an example with same set of columns used as above.

g = sns.PairGrid(sample[["price", "carat", "depth"]])  
g.map\_lower(sns.scatterplot)  
g.map\_upper(sns.kdeplot)  
g.map\_diag(sns.histplot)

In the above code, map\_lower specifies lower triangular portion filled using scatterplot, map\_upper specifies upper triangular portion filled with kdeplot and map\_diag specifies diagonal parts filled with histplot in the chart.



sns.PairGrid