Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

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:

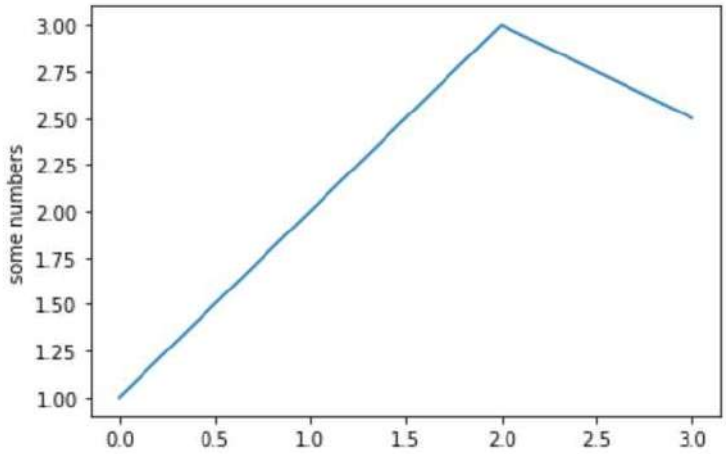
Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python interpreter and IPython shell, the jupyter notebook, web application servers, and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

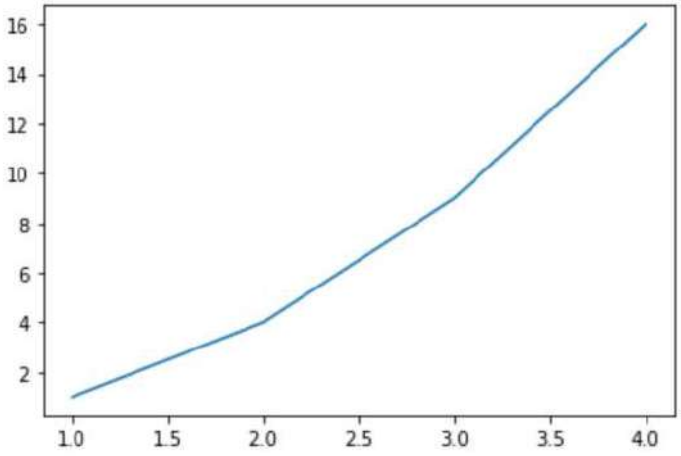
Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**



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:

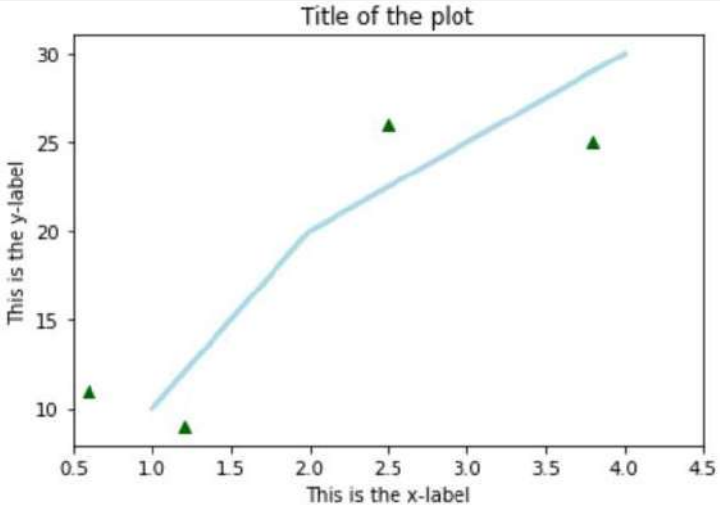
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16]);  # first parameter is x, second is y
```



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.

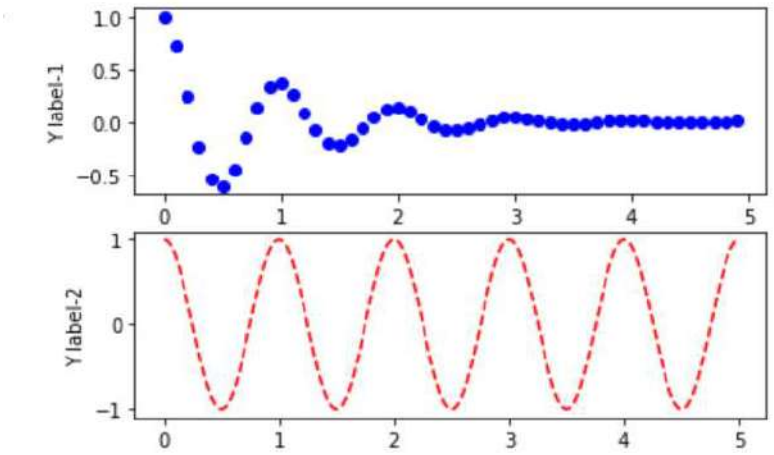
```
import numpy as np

# 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()
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

```
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)
```



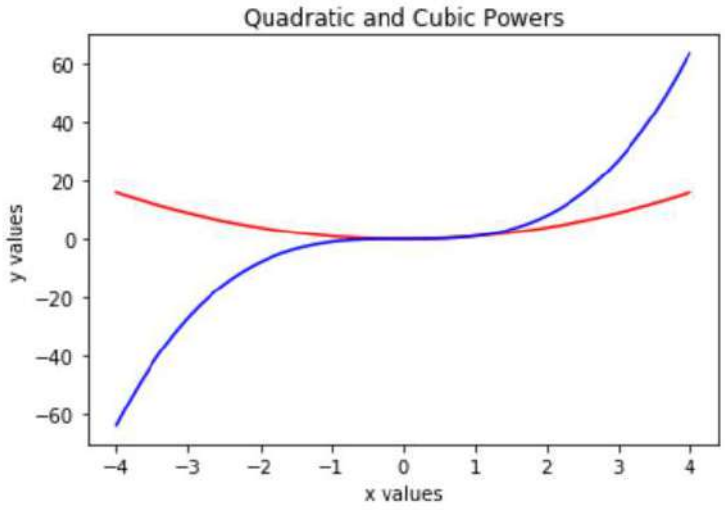
What you do when plotting with the object oriented interface, is to create the objects and then call methods for every object to make changes specifically to that object.
The plt.subplots function creates a figure object and an Subplot (Axes) object at the same time, then we can create a plot by calling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

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

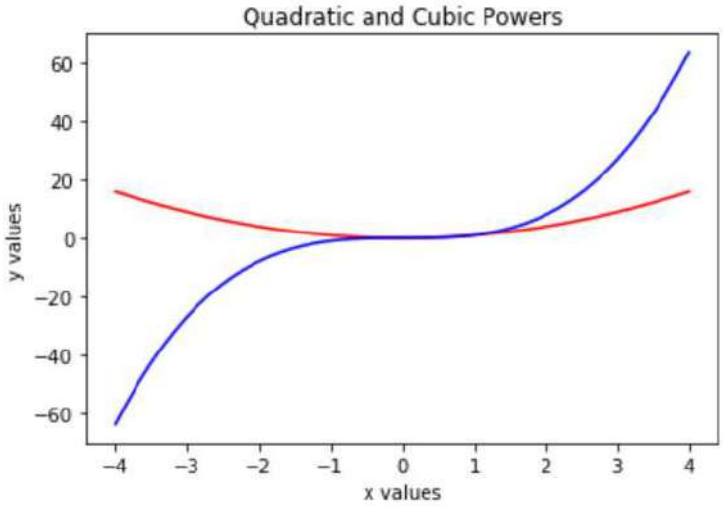
```
ax.plot(x, y2, 'blue')
ax.set_title('Quadratic and Cubic Powers')
ax.set_xlabel('x values')
ax.set_ylabel('y values'); # put a ; to remove the unnecessary Text(...) part
```



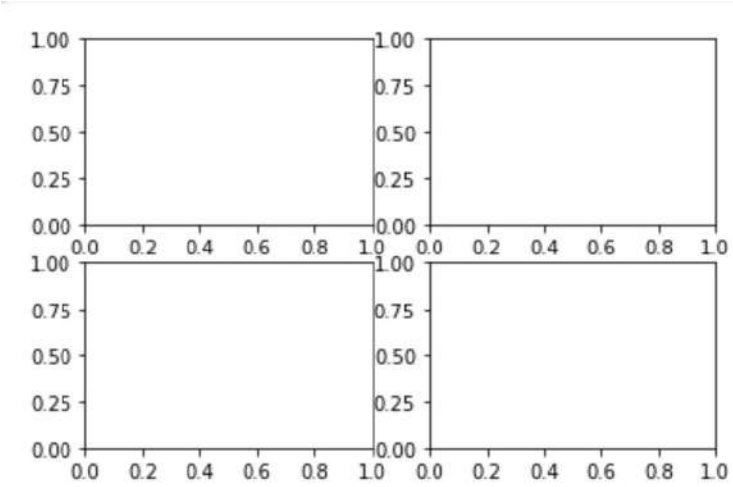

```
# pyplot
plt.plot(x, y1, 'red', x, y2, 'blue')
plt.title('Quadratic and Cubic Powers')
plt.xlabel('x values')
plt.ylabel('y values');
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**




```
# create a figure object with with 4 subplots as 2 by 2
fig,axes=plt.subplots(nrows=2,ncols=2)
```

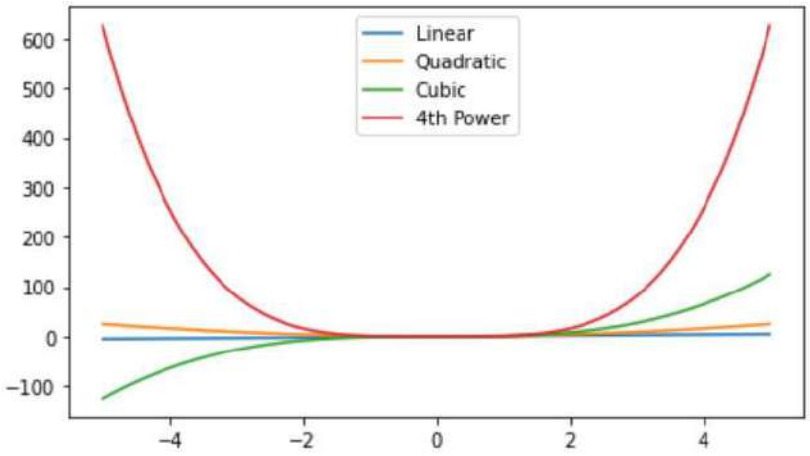


Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

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)

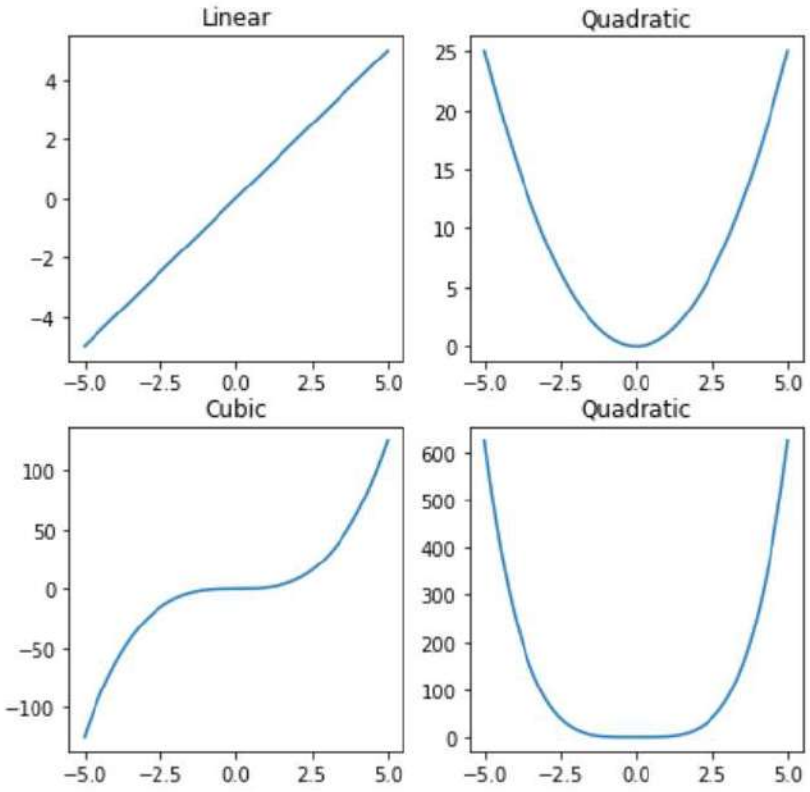
Code:

```
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
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

```
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)
```



Academic Year: 2022-23 **Semester: VI**

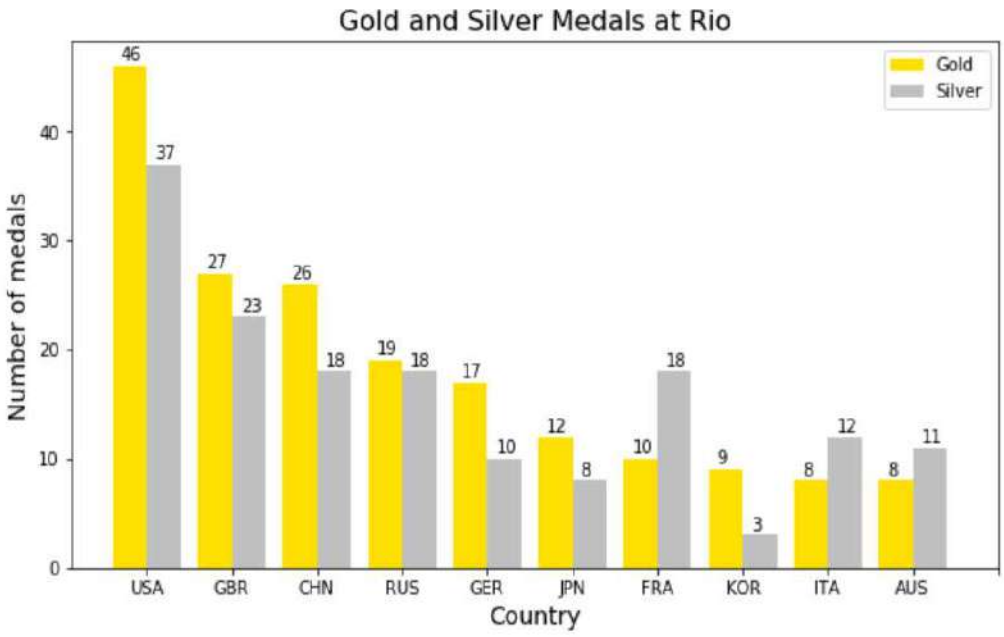
Class / Branch: TE IT A/B **Subject: DS using Python Lab**

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

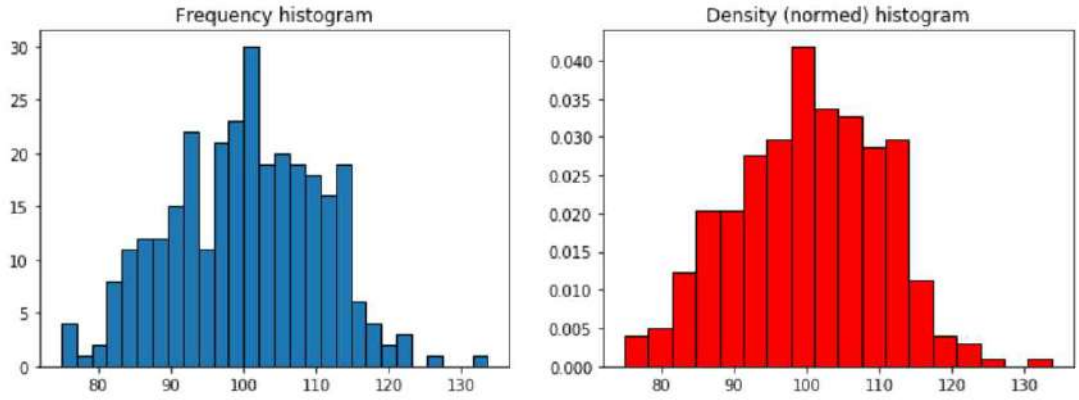
A histogram is a graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable). It is a kind of bar graph. To construct a histogram, the first step is to "bin" the range of values, that is, divide the 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');
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**



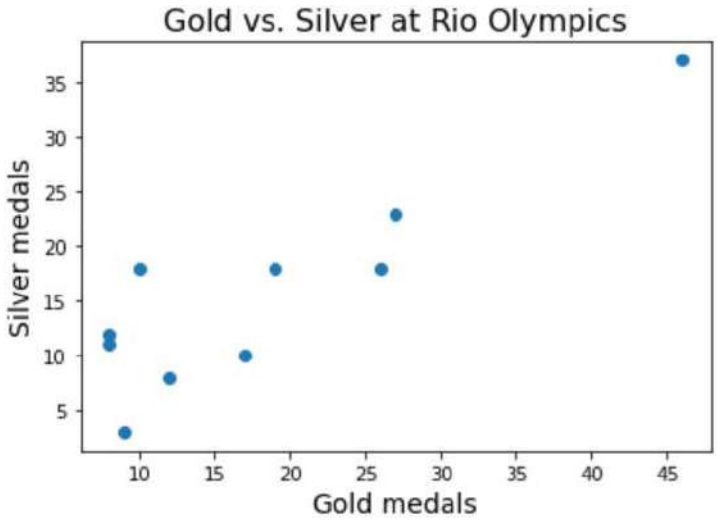
Scatter plot

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);
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**



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)
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

Species PetalWidthCm	Iris-setosa	Iris-versicolor	Iris-virginica
0.1	6	0	0
0.2	28	0	0
0.3	7	0	0
0.4	7	0	0
0.5	1	0	0
0.6	1	0	0
1.0	0	7	0
1.1	0	3	0
1.2	0	5	0
1.3	0	13	0
1.4	0	7	1
1.5	0	10	2
1.6	0	3	1
1.7	0	1	1
1.8	0	1	11
1.9	0	0	5
2.0	0	0	6
2.1	0	0	6
2.2	0	0	3
2.3	0	0	8
2.4	0	0	3
2.5	0	0	3

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.

Academic Year: 2022-23 **Semester: VI**

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()
```

	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

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

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

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()
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

	carat	cut	color	clarity	depth	table	price	x	y	z
36972	0.41	Ideal	H	VS1	61.8	56.0	962	4.78	4.73	2.94
26738	1.52	Good	F	VS1	63.4	60.0	16519	7.25	7.30	4.61
15499	1.10	Premium	D	VS2	60.0	59.0	6209	6.67	6.60	3.98
24993	1.50	Premium	F	VS2	62.0	53.0	13506	7.37	7.33	4.56
44694	0.56	Ideal	H	SI1	61.6	56.0	1612	5.29	5.33	3.27

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.

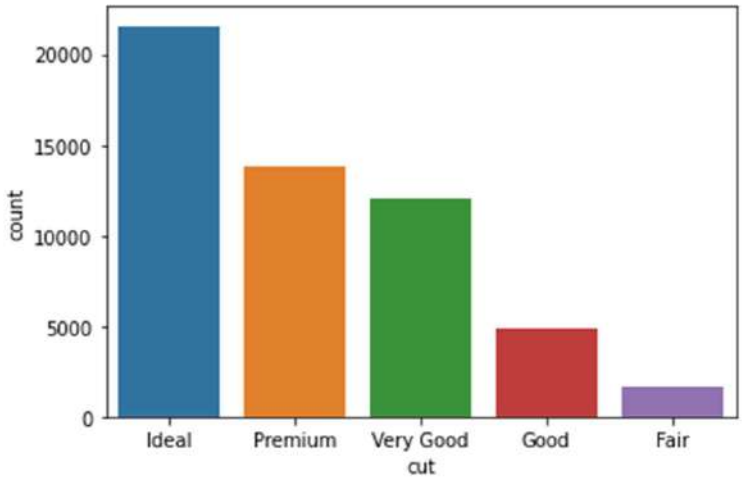
COUNTPLOT

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')
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**



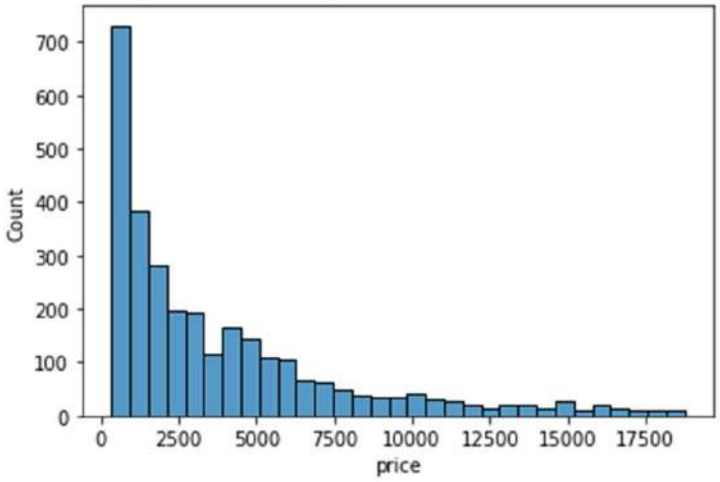
sns.countplot

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.



Academic Year: 2022-23 **Semester: VI**

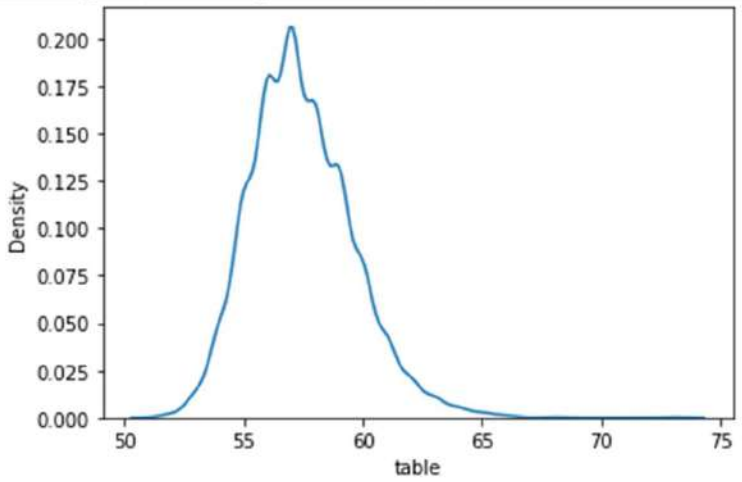
Class / Branch: TE IT A/B **Subject: DS using Python Lab**

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

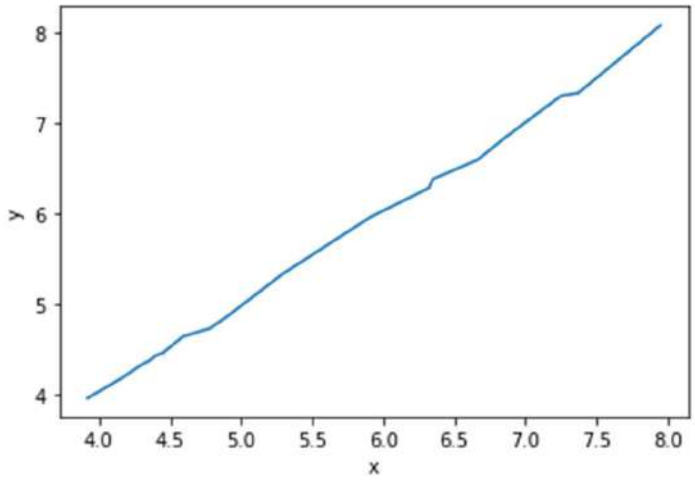
LINEPLOT

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')
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**



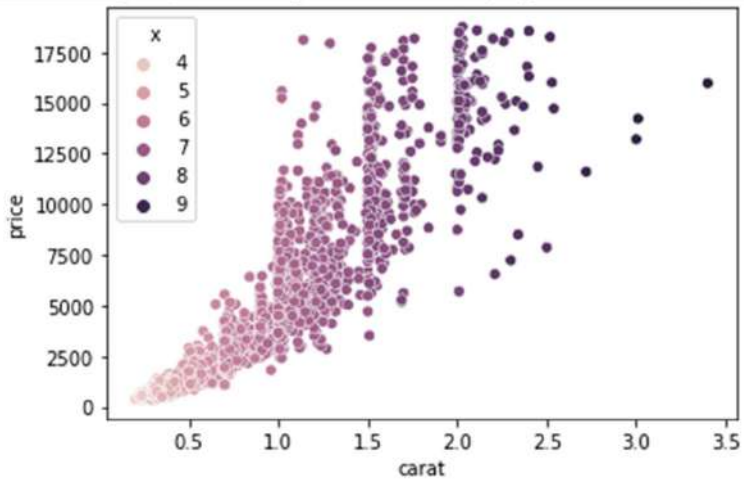
sns.lineplot

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

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

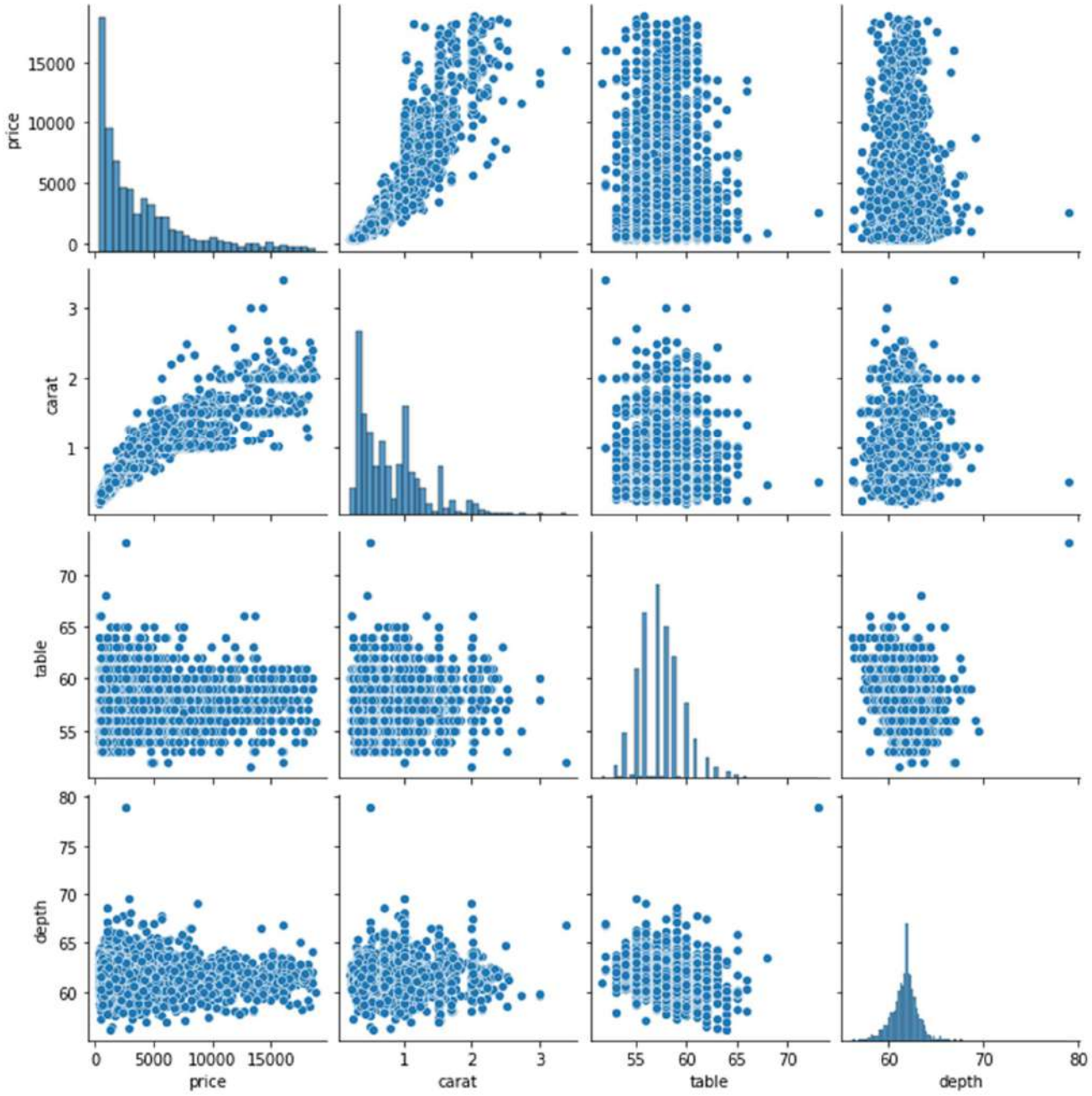
PAIRPLOT

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"]])
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**



sns.pairplot

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

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()
```

	carat	depth	table	price	x	y	z
carat	1.000000	0.028224	0.181618	0.921591	0.975094	0.951722	0.953387
depth	0.028224	1.000000	-0.295779	-0.010647	-0.025289	-0.029341	0.094924
table	0.181618	-0.295779	1.000000	0.127134	0.195344	0.183760	0.150929
price	0.921591	-0.010647	0.127134	1.000000	0.884435	0.865421	0.861249
x	0.975094	-0.025289	0.195344	0.884435	1.000000	0.974701	0.970772

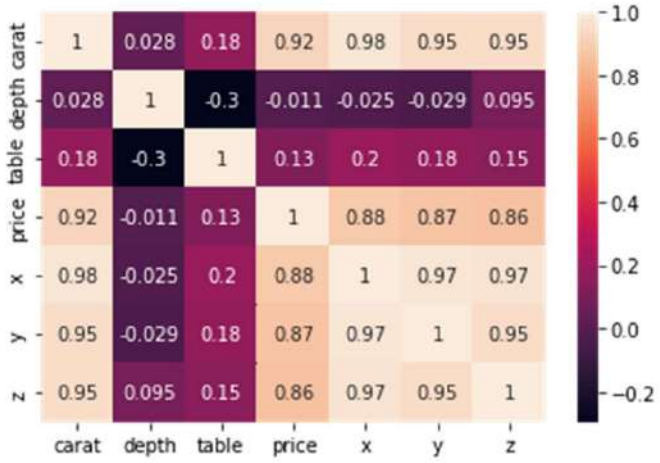
HEATMAP

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)
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**



sns.heatmap

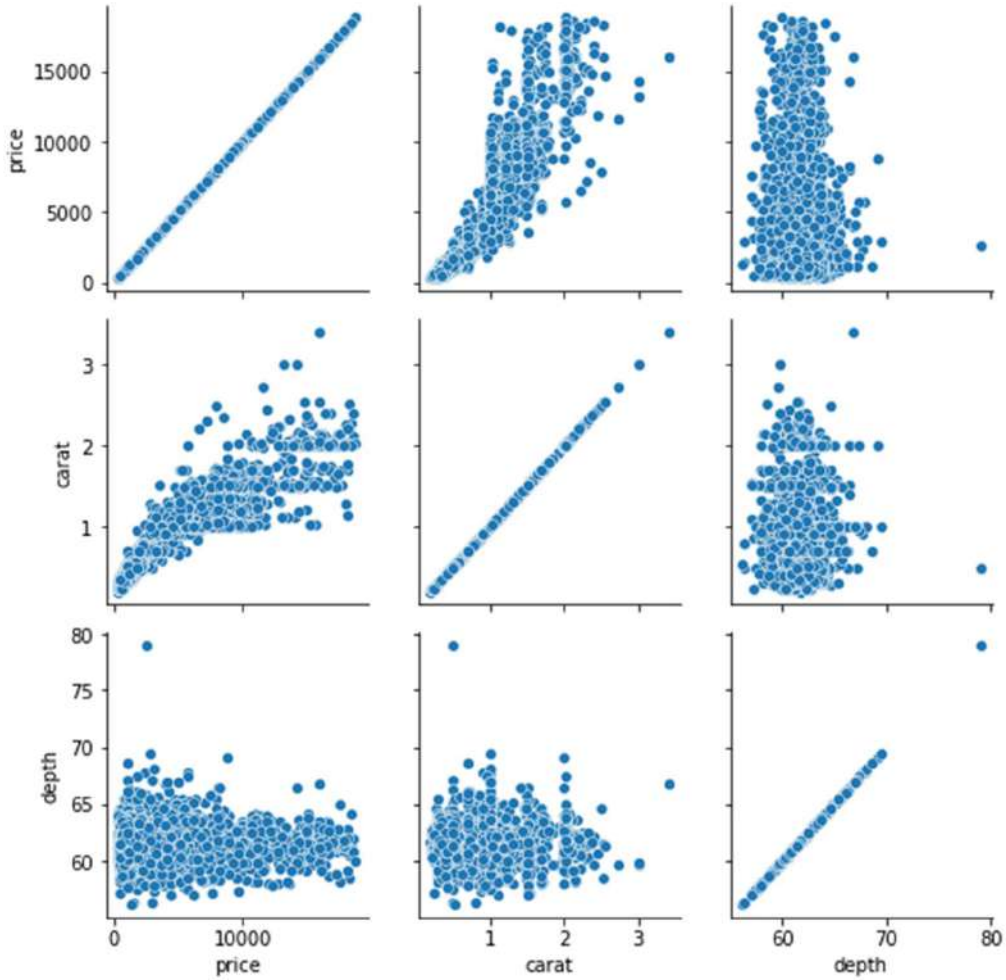
PAIRPLOT

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)
```

Academic Year: 2022-23 **Semester: VI**

Class / Branch: TE IT A/B **Subject: DS using Python Lab**

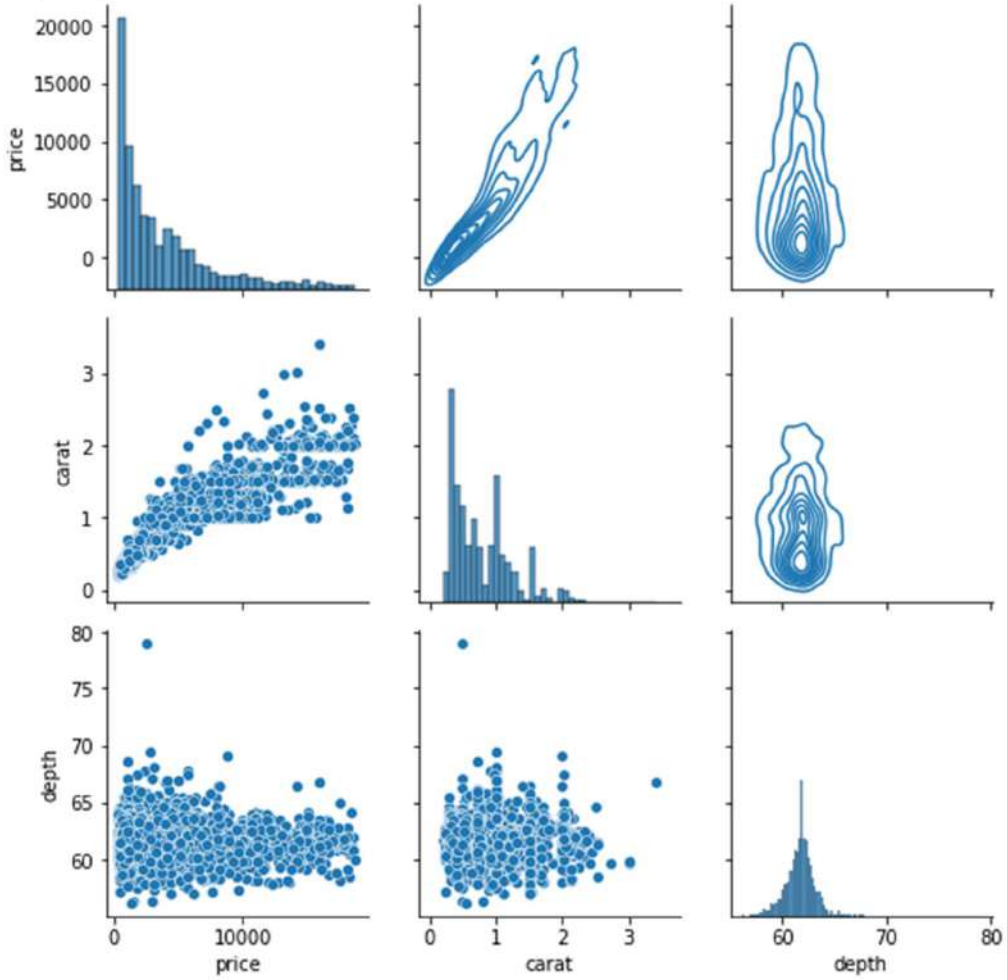


sns.PairGrid

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