Ex No: 1

UNIVARIATE, BIVARIATE, MULTIVARIATE PROFILING – PYTHON

AIM:

To analyse the dataset, get statistical description and to visualize it in python.

Dataset Description:

The dataset contains information on over 2,000 mobile phones from different brands. It includes details such as the storage capacity, RAM, screen size, camera specifications, battery capacity, and price of each device.

The dataset is structured as a CSV file with 7 columns:

- Brand: The brand name of the mobile phone.
- Model: The model name of the mobile phone.
- Storage: The amount of storage space available on the mobile phone in GB.
- RAM: The amount of random access memory available on the mobile phone in GB.
- Screen Size: The size of the mobile phone's screen in inches.
- Camera: The quality of the mobile phone's cameras, measured in megapixels.
- Battery Capacity: The amount of battery life the mobile phone has in mAh.
- Price: The price of the mobile phone in USD.

Problem Statement

The mobile phone price prediction problem is to develop a model that can predict the price of a mobile phone given a set of features. The target variable is the price of the mobile phone in USD. The goal of the problem is to develop a model that can accurately predict the price of a mobile phone given its features. This model can be used by a variety of stakeholders, including:

Mobile phone manufacturers: Manufacturers can use the model to develop a pricing strategy for their products. They can also use the model to identify the features that are most important to consumers and to determine how much they should charge for their phones based on those features.

Retailers: Retailers can use the model to set prices for mobile phones in their store. They can also use the model to compare the prices of different phones from different manufacturers and to ensure that they are charging a competitive price.

Consumers: Consumers can use the model to make informed decisions about which mobile phone to buy. They can use the model to compare the prices of different phones with different features and to find the best value for their money.

PROGRAMS WITH OUTPUT:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from google.colab import drive

drive.mount('/content/drive')

p1 = '/content/drive/MyDrive/Colab Notebooks/MVT/Mobile.csv'

df = pd.read_csv(p1)
```

```
[ ] df.head()
```

	Brand	Model	Storage	RAM	Screen Size (inches)	Camera (MP)	Battery Capacity (mAh)	Price ($)
0	Apple	iPhone 13 Pro	128 GB	6 GB	6.1	12 + 12 + 12	3095	999
1	Samsung	Galaxy S21 Ultra	256 GB	12 GB	6.8	108 + 10 + 10 + 12	5000	1199
2	OnePlus	9 Pro	128 GB	8 GB	6.7	48 + 50 + 8 + 2	4500	899
3	Xiaomi	Redmi Note 10 Pro	128 GB	6 GB	6.67	64 + 8 + 5 + 2	5020	279
4	Google	Pixel 6	128 GB	8 GB	6.4	50 + 12.2	4614	799

```
df.describe()
```

	Battery Capacity (mAh)
count	407.000000
mean	4676.476658
std	797.193713
min	1821.000000
25%	4300.000000
50%	5000.000000
75%	5000.000000
max	7000.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 407 entries, 0 to 406
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Brand                   407 non-null    object
 1   Model                   407 non-null    object
 2   Storage                 407 non-null    object
 3   RAM                     407 non-null    object
 4   Screen Size (inches)    407 non-null    object
 5   Camera (MP)             407 non-null    object
 6   Battery Capacity (mAh)  407 non-null    int64 
 7   Price ($)               407 non-null    object
dtypes: int64(1), object(7)
memory usage: 25.6+ KB
```

```
df.isnull().sum()
```

```
Brand                     0
Model                     0
Storage                   0
RAM                       0
Screen Size (inches)      0
Camera (MP)               0
Battery Capacity (mAh)    0
Price ($)                 0
dtype: int64
```

```
[ ] df.columns
```

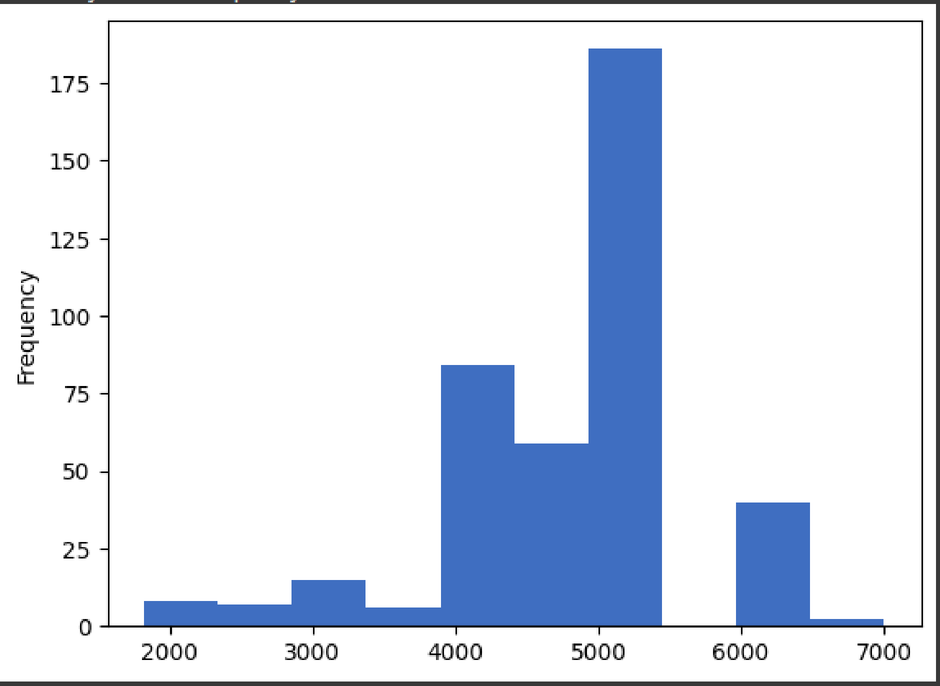
```
Index(['Brand', 'Model', 'Storage ', 'RAM ', 'Screen Size (inches)',
       'Camera (MP)', 'Battery Capacity (mAh)', 'Price ($)'],
      dtype='object')
```

Univariate Analysis

Univariate Analysis

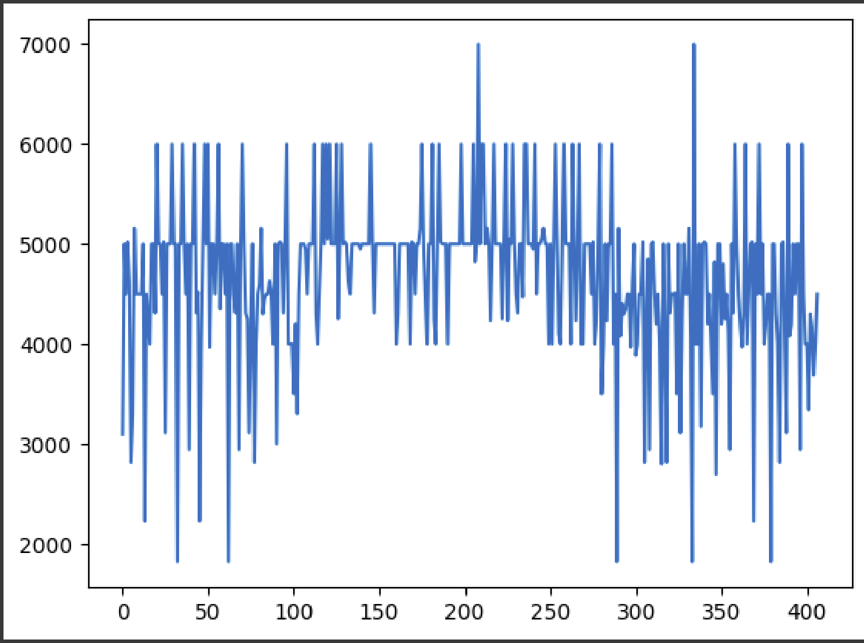
```
[ ] df['Battery Capacity (mAh)'].describe()

count     407.000000
mean     4676.476658
std       797.193713
min      1821.000000
25%      4300.000000
50%      5000.000000
75%      5000.000000
max      7000.000000
Name: Battery Capacity (mAh), dtype: float64
```

```
df['Battery Capacity (mAh)'].plot(kind="hist")

<Axes: ylabel='Frequency'>
```



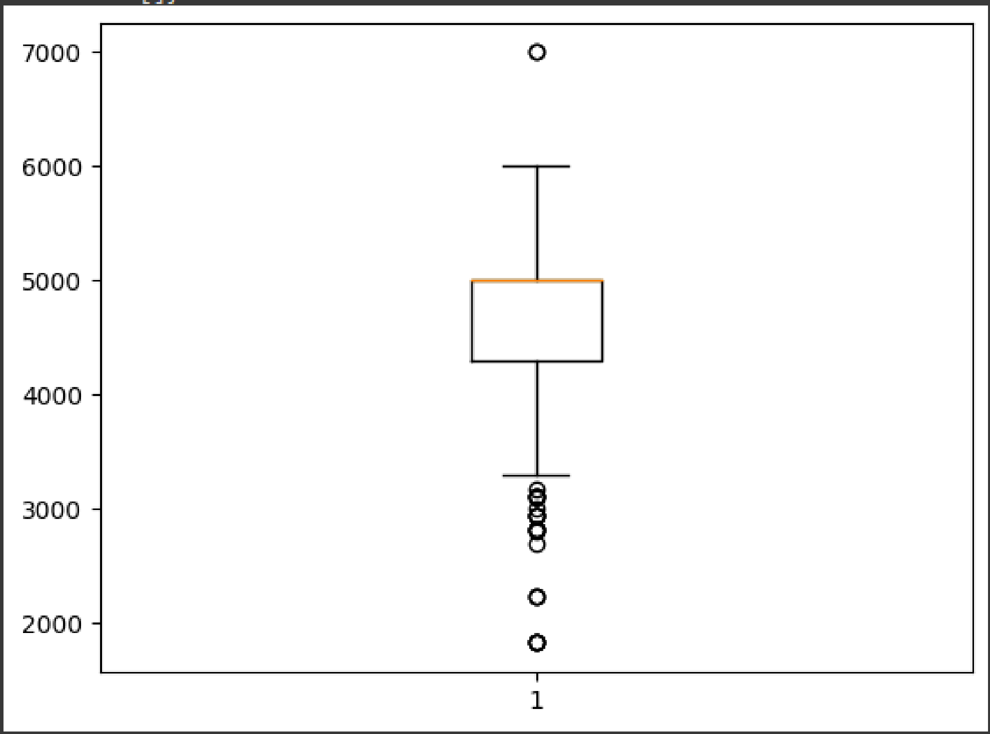
```
df['Battery Capacity (mAh)'].plot(kind="line")
```

```
<Axes: >
```



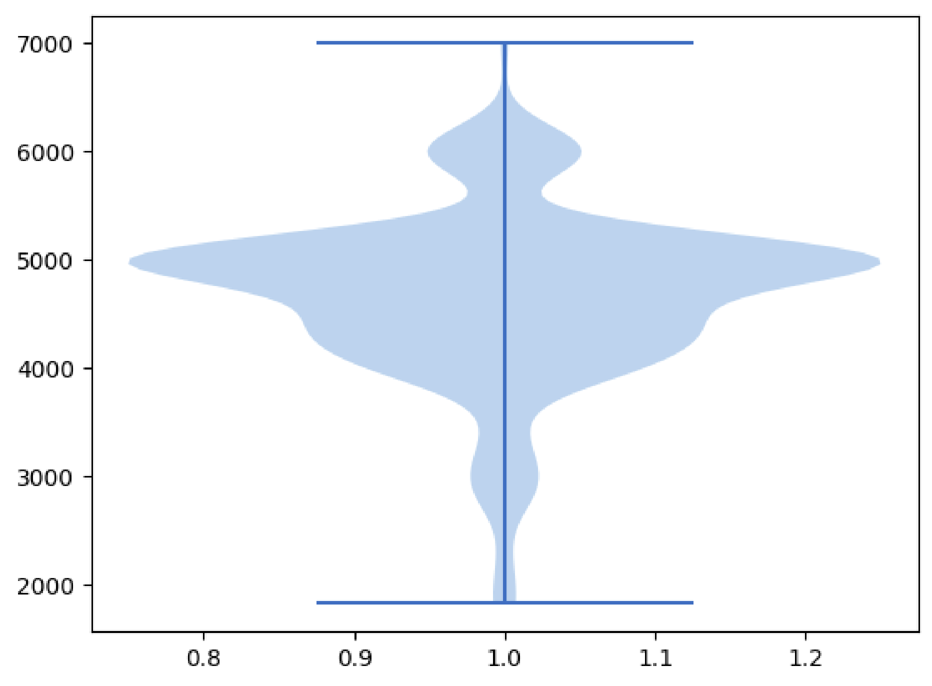

```
plt.boxplot(df['Battery Capacity (mAh)'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7c0629811480>,
  <matplotlib.lines.Line2D at 0x7c06298101f0>],
 'caps': [<matplotlib.lines.Line2D at 0x7c06298118a0>,
  <matplotlib.lines.Line2D at 0x7c0629811b40>],
 'boxes': [<matplotlib.lines.Line2D at 0x7c06298111e0>],
 'medians': [<matplotlib.lines.Line2D at 0x7c0629811de0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7c0629812080>],
 'means': []}
```



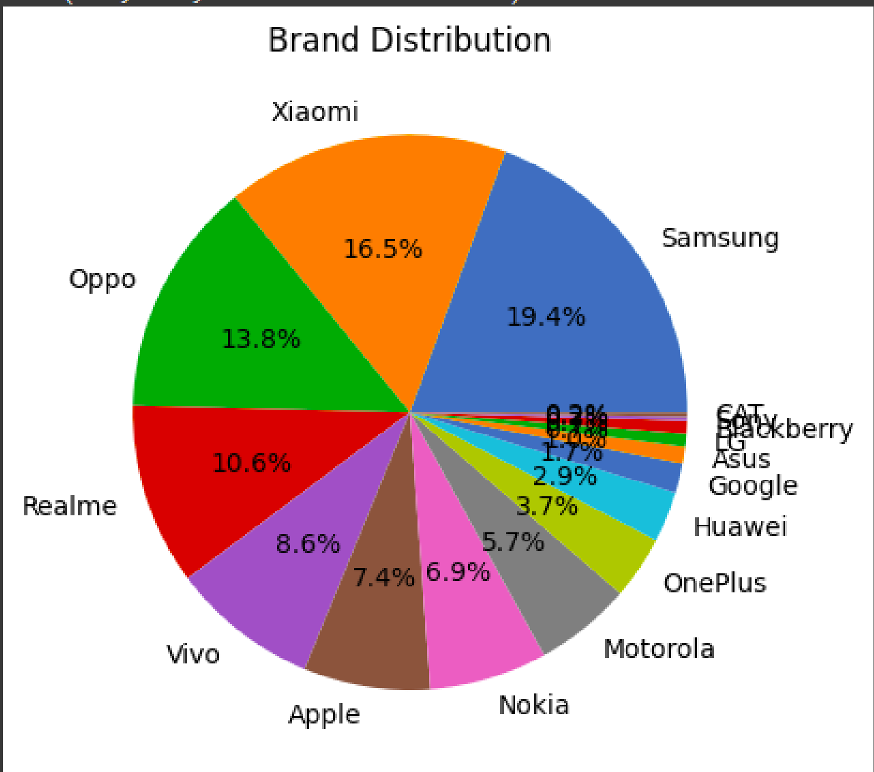
```python
plt.violinplot(df['Battery Capacity (mAh)'])
```

```
{'bodies': [<matplotlib.collections.PolyCollection at 0x7c062986fdf0>],
 'cmaxes': <matplotlib.collections.LineCollection at 0x7c062986fd90>,
 'cmins': <matplotlib.collections.LineCollection at 0x7c06286ac400>,
 'cbars': <matplotlib.collections.LineCollection at 0x7c06286ac760>}
```




```python
dependents = df['Brand'].value_counts()
plt.pie(dependents.values, labels=dependents.index, autopct='%1.1f%%')
plt.title("Brand Distribution")
```

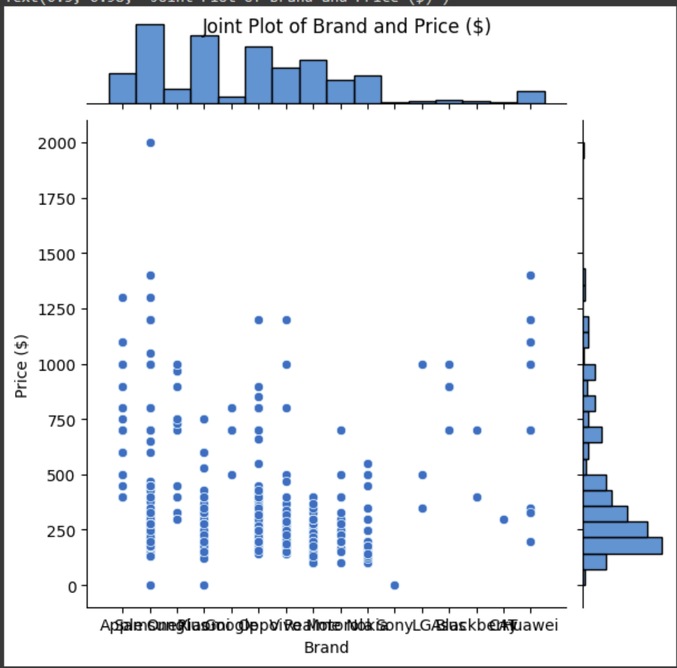
```
Text(0.5, 1.0, 'Brand Distribution')
```



BIVARIATE ANALYSIS

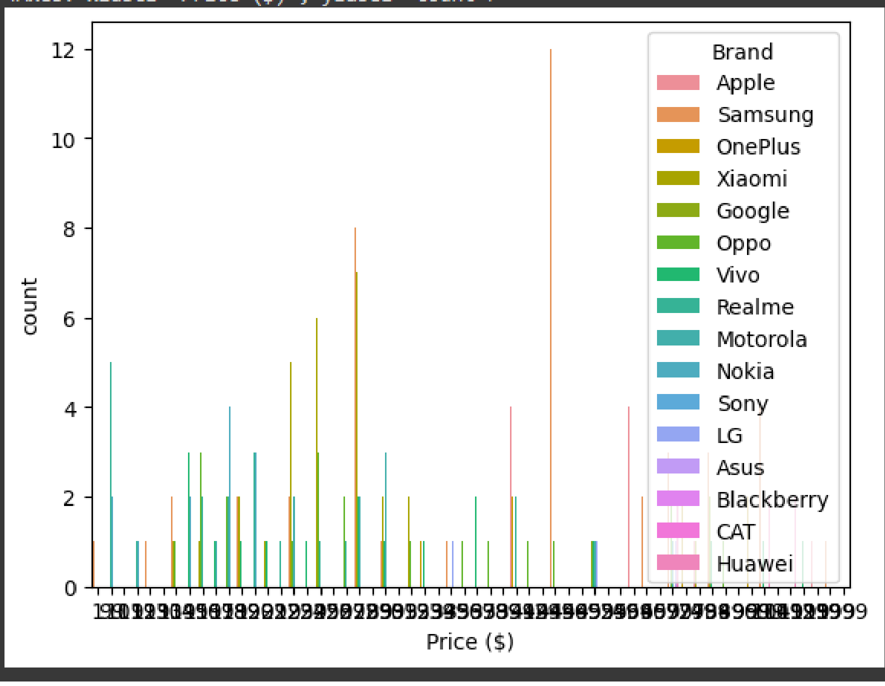
```
sns.jointplot(df,x="Brand",y="Price ($)",kind="scatter")
plt.suptitle("Joint Plot of Brand and Price ($)")
```

```
Text(0.5, 0.98, 'Joint Plot of Brand and Price ($)')
```



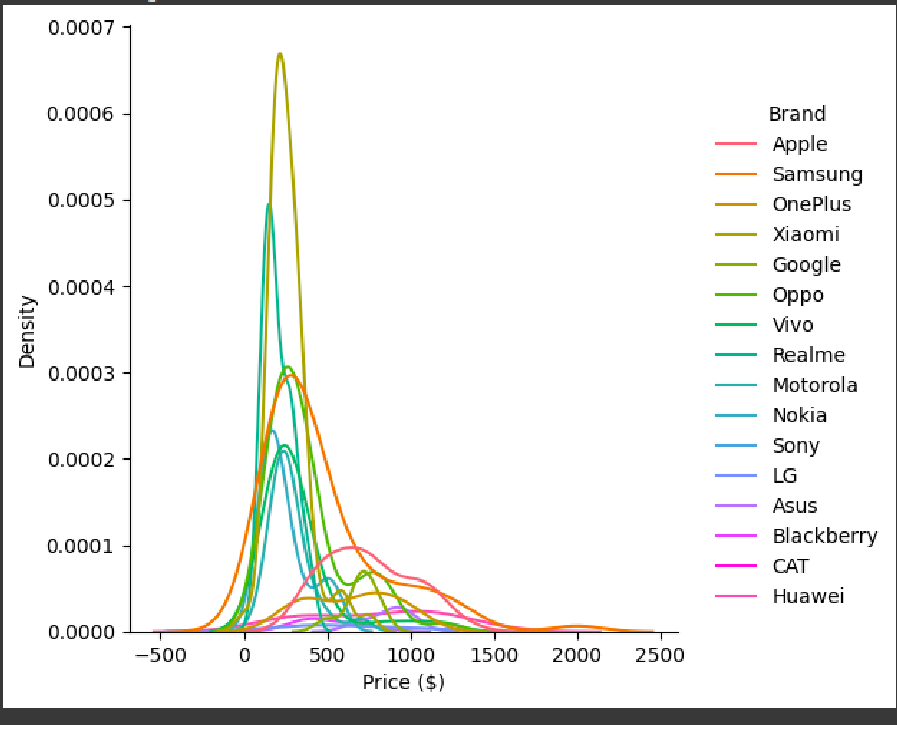

```
sns.countplot(df, x="Price ($)", hue="Brand")
```

```
<Axes: xlabel='Price ($)', ylabel='count'>
```

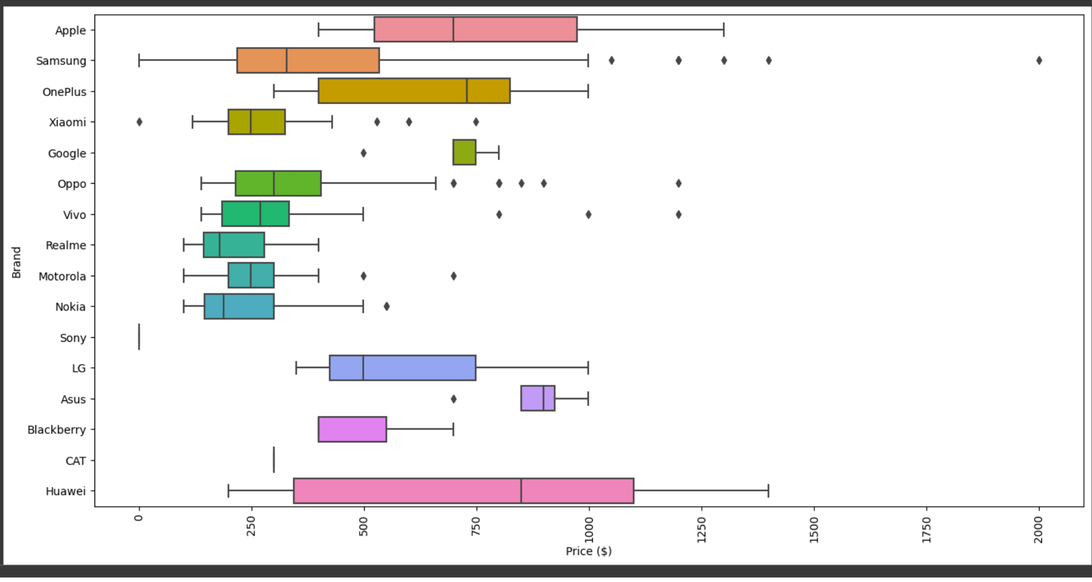


```
sns.displot(df, x= "Price ($)", hue="Brand", kind="kde")
```

```
<ipython-input-56-2ef00affcee>:1: UserWarning: Dataset has 0 variance; skipping density estimate. Pas
  sns.displot(df, x= "Price ($)", hue="Brand", kind="kde")
<ipython-input-56-2ef00affcee>:1: UserWarning: Dataset has 0 variance; skipping density estimate. Pas
  sns.displot(df, x= "Price ($)", hue="Brand", kind="kde")
<seaborn.axisgrid.FacetGrid at 0x7c061f275810>
```

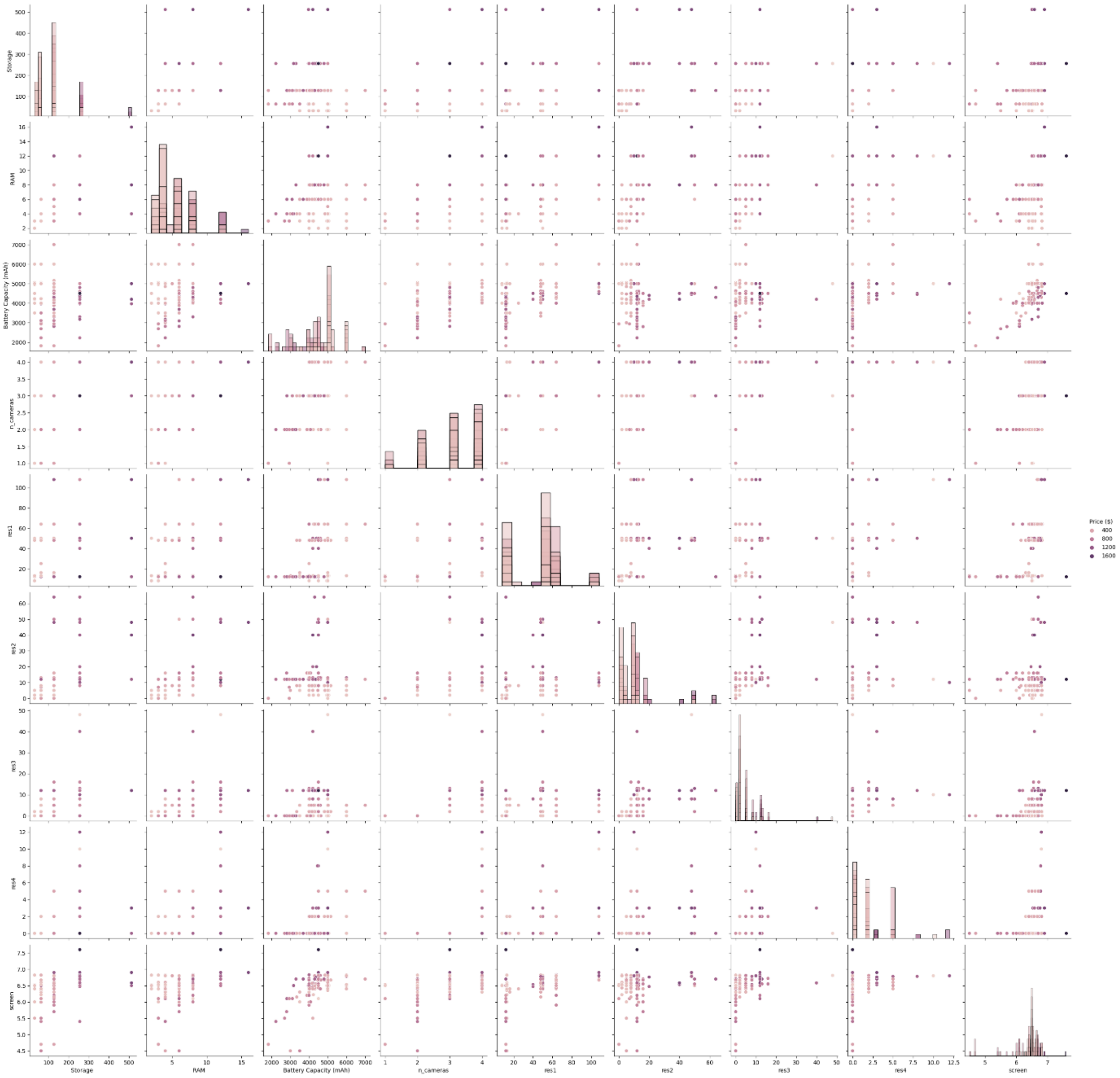



```
var = 'Price ($)'
data = pd.concat([df['Brand'], df[var]], axis=1)
f, ax = plt.subplots(figsize=(16, 8))
fig = sns.boxplot(x=var, y="Brand", data=data)
plt.xticks(rotation=90);
```

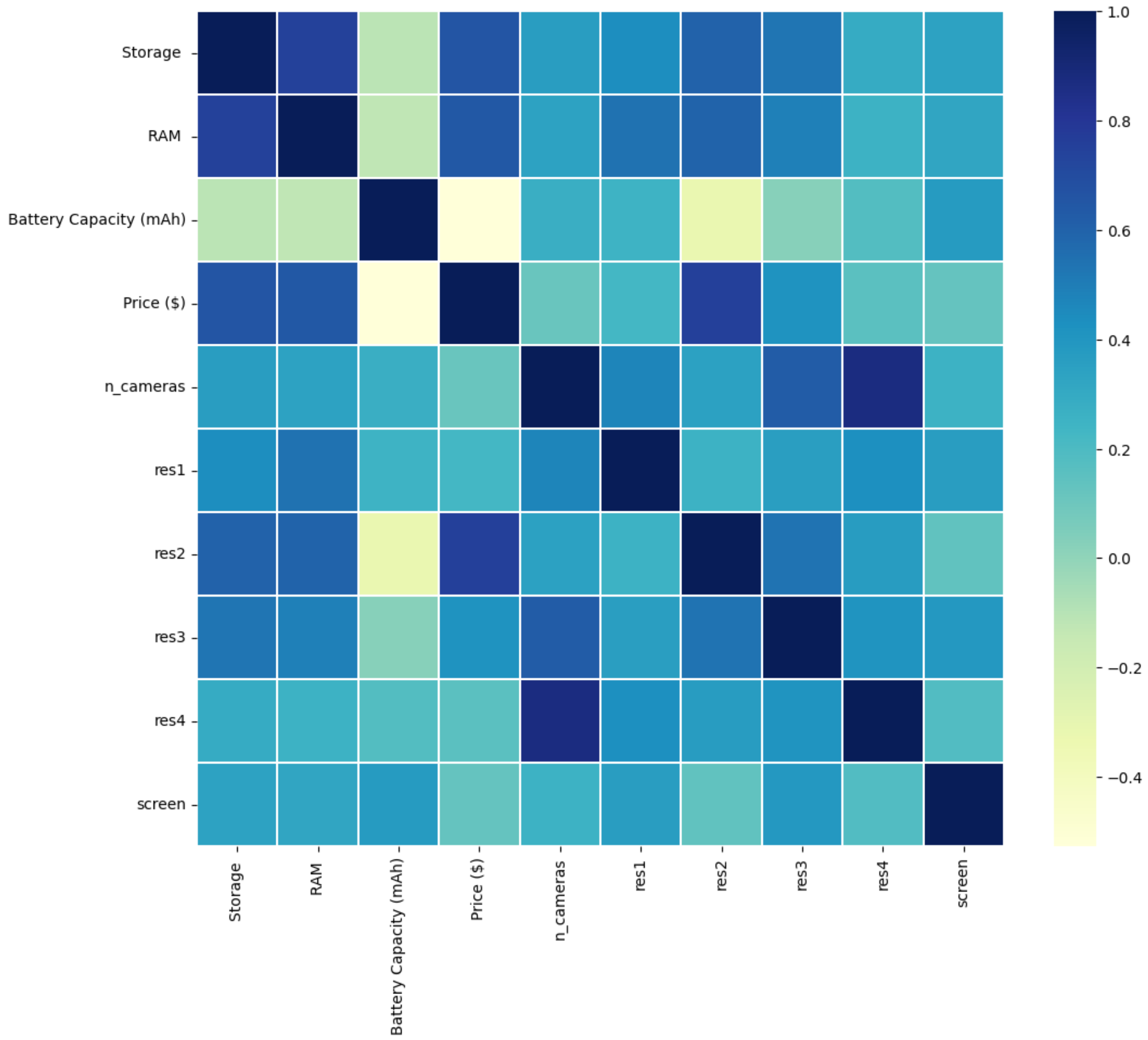


MULTI VARIATE ANALYSIS

```
s=sns.pairplot(df,hue='Price ($)',size=3,diag_kind='hist')
```



```
corrmat = df.corr(method='spearman')
f, ax = plt.subplots(figsize=(12, 10))
sns.heatmap(corrmat, ax=ax, cmap="YlGnBu", linewidths=0.1)
```



CONCLUSION:

The univariate, bivariate and multivariate data analysis has been done using the given dataset and the results have been analyzed using the above visualizations.

Ex No: 2

Exploratory Data Analysis (EDA)

AIM:

To perform exploratory data analysis for the dataset and obtain Measures of Central Tendency, Measure of Dispersion, Descriptive Statistics, Skewness and Kurtosis, and correlation using python.

Dataset Description:

The dataset contains information on over 2,000 mobile phones from different brands. It includes details such as the storage capacity, RAM, screen size, camera specifications, battery capacity, and price of each device.

The dataset is structured as a CSV file with 7 columns:

- Brand: The brand name of the mobile phone.
- Model: The model name of the mobile phone.
- Storage: The amount of storage space available on the mobile phone in GB.
- RAM: The amount of random access memory available on the mobile phone in GB.
- Screen Size: The size of the mobile phone's screen in inches.
- Camera: The quality of the mobile phone's cameras, measured in megapixels.
- Battery Capacity: The amount of battery life the mobile phone has in mAh.
- Price: The price of the mobile phone in USD.

Problem Statement

The mobile phone price prediction problem is to develop a model that can predict the price of a mobile phone given a set of features. The target variable is the price of the mobile phone in USD. The goal of the problem is to develop a model that can accurately predict the price of a mobile phone given its features. This model can be used by a variety of stakeholders, including:

Mobile phone manufacturers: Manufacturers can use the model to develop a pricing strategy for their products. They can also use the model to identify the features that are most important to consumers and to determine how much they should charge for their phones based on those features.

Retailers: Retailers can use the model to set prices for mobile phones in their store. They can also use the model to compare the prices of different phones from different manufacturers and to ensure that they are charging a competitive price.

Consumers: Consumers can use the model to make informed decisions about which mobile phone to buy. They can use the model to compare the prices of different phones with different features and to find the best value for their money.

PROGRAMS WITH OUTPUT:

df.head()

```
[ ] df.head()
```

	Brand	Model	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
0	Apple	iPhone 13 Pro	128	6	3095	999	3	12	12	12	0	6.10
1	Samsung	Galaxy S21 Ultra	256	12	5000	1199	4	108	10	10	12	6.80
2	OnePlus	9 Pro	128	8	4500	899	4	48	50	8	2	6.70
3	Xiaomi	Redmi Note 10 Pro	128	6	5020	279	4	64	8	5	2	6.67
4	Google	Pixel 6	128	8	4614	799	2	50	12	0	0	6.40

Measures of Central Tendency

```
df['Price ($)'].mean()

399.23832923832924

[ ] df['Price ($)'].mode()

0    199
Name: Price ($), dtype: int64

[ ] df['Price ($)'].mode()[0]

199

[ ] df['Price ($)'].median()

299.0
```

Measure of Dispersion

1. **Range**

Range

```
[ ] range=df['Price ($)'].max()-df['Price ($)'].min()
    range
```

```
1998
```

```
[ ] q1=df['Price ($)'].quantile(0.25)
    q2=df['Price ($)'].quantile(0.50)
    q3=df['Price ($)'].quantile(0.75)
```

```
[ ] IQR=q3-q1
    IQR
```

```
270.0
```

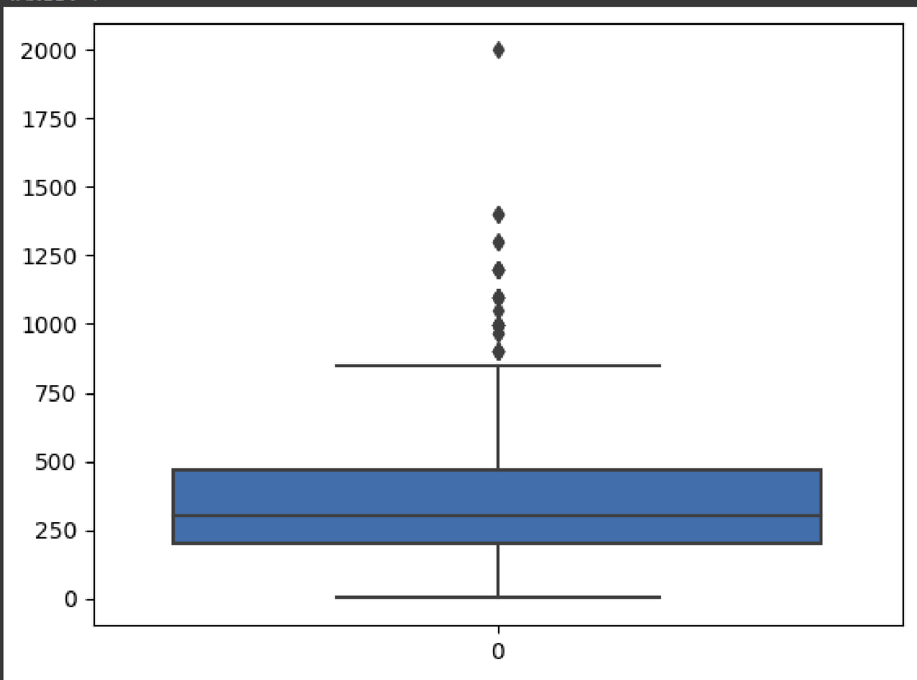
```
[ ] lb=q1-(1.5*IQR)
    ub=q3+(1.5*IQR)
```

```
lb,ub
```

```
(-206.0, 874.0)
```

```
[ ] sns.boxplot(df['Price ($)'])
```

<Axes: >



```
df_new=df[['Price ($)']]
df_new
```

	Price ($)
0	999
1	1199
2	899
3	279
4	799
...	...
402	1049
403	349
404	1099
405	429
406	649

407 rows × 1 columns

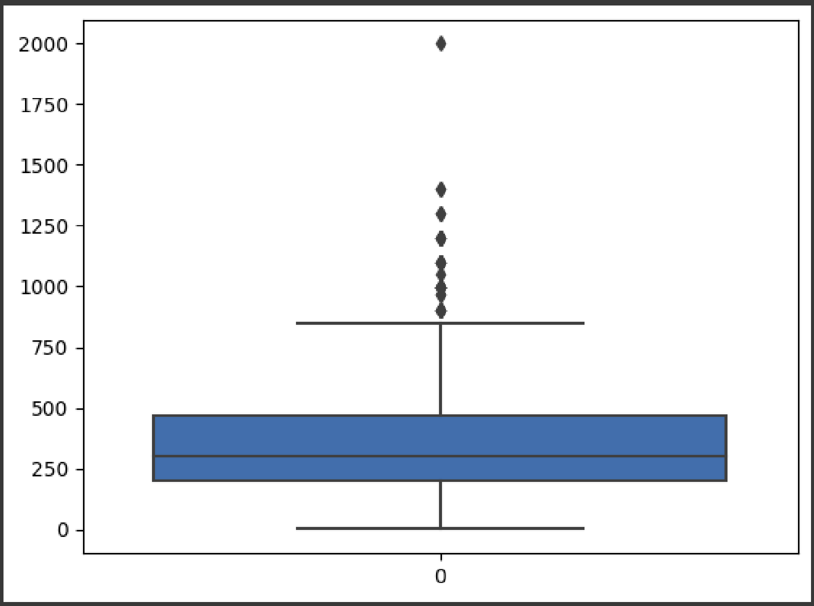
```
[ ] df_new = df_new[~((df_new < lb) & (df_new > ub)).any(axis=1)]
```

```
[ ] df_new.shape
```

```
(407, 1)
```

```
sns.boxplot(df_new['Price ($)'])
```

```
<Axes: >
```



2. Variance And Standard Deviation

Variance

- Variance measure the dispersion of the data from the mean
- It is an average of the sum of squares of difference between an observation and the mean. Thus the variance is always positive

```
[ ] df['Price ($)'].var()
```

```
85920.56621198
```

Standard Deviation

1. Standard deviation is the positive square root of the variance
2. It has the same unit as that of the observations

```
[ ] df['Price ($)'].std()
```

```
293.1221012001313
```

Coefficient of Variation

- The coefficient of variation is a statistical measure of dispersion of data points around the mean
- It is a unit free measure and is always expressed in percentage

```
[ ] (df['Price ($)'].std()/df['Price ($)'].mean())*100
```

```
73.42033059785429
```

```
df.cov()
```

```
<ipython-input-51-6f98a29763d5>:1: FutureWarning: The default value of numeric_only in DataFrame.cov is deprecated. In a future version, it will default to False. Select only valid columns or
  df.cov()
```

	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
Storage	4220.212101	109.224338	-3881.613440	13042.602147	15.695186	580.174822	348.129604	177.135014	32.528292	5.385899
RAM	109.224338	5.914525	-53.562908	421.110172	0.646718	30.709160	13.222341	5.833311	1.304886	0.229575
Battery Capacity (mAh)	-3881.613440	-53.562908	635517.816572	-96091.054768	243.334673	5704.681062	-940.136230	-179.190660	238.627480	155.804883
Price ($)	13042.602147	421.110172	-96091.054768	85920.566212	11.546647	780.372551	1783.753313	731.031517	84.557582	3.713550
n_cameras	15.695186	0.646718	243.334673	11.546647	0.616865	9.291379	1.772431	1.332918	0.880309	0.113428
res1	580.174822	30.709160	5704.681062	780.372551	9.291379	608.714564	38.115358	21.697722	17.969245	3.022312
res2	348.129604	13.222341	-940.136230	1783.753313	1.772431	38.115358	103.523668	22.614027	3.935029	0.504686
res3	177.135014	5.833311	-179.190660	731.031517	1.332918	21.697722	22.614027	24.038489	2.248635	0.374393
res4	32.528292	1.304886	238.627480	84.557582	0.880309	17.969245	3.935029	2.248635	3.001731	0.118412
screen	5.385899	0.229575	155.804883	3.713550	0.113428	3.022312	0.504686	0.374393	0.118412	0.102798

```
[ ] import scipy
    from scipy.stats import variation

    scipy.stats.variation(df['Price ($)'])
```

```
0.7333007815980785
```

Descriptive Statistics

```
[ ] df.describe(include='all')
```

	Brand	Model	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
count	407	407	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000
unique	16	239	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	Samsung	Poco X3 Pro	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	79	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	123.046683	5.837838	4676.476658	399.238329	3.144963	43.316953	9.125307	3.641278	1.027027	6.471327
std	NaN	NaN	64.963160	2.431980	797.193713	293.122101	0.785408	24.672141	10.174658	4.902906	1.732550	0.320622
min	NaN	NaN	32.000000	2.000000	1821.000000	1.000000	1.000000	8.000000	0.000000	0.000000	0.000000	4.500000
25%	NaN	NaN	64.000000	4.000000	4300.000000	199.000000	3.000000	13.000000	2.000000	2.000000	0.000000	6.440000
50%	NaN	NaN	128.000000	6.000000	5000.000000	299.000000	3.000000	48.000000	8.000000	2.000000	0.000000	6.500000
75%	NaN	NaN	128.000000	8.000000	5000.000000	469.000000	4.000000	64.000000	12.000000	5.000000	2.000000	6.590000
max	NaN	NaN	512.000000	16.000000	7000.000000	1999.000000	4.000000	108.000000	64.000000	48.000000	12.000000	7.600000

```
[ ] df.describe(include=np.number)
```

	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
count	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000	407.000000
mean	123.046683	5.837838	4676.476658	399.238329	3.144963	43.316953	9.125307	3.641278	1.027027	6.471327
std	64.963160	2.431980	797.193713	293.122101	0.785408	24.672141	10.174658	4.902906	1.732550	0.320622
min	32.000000	2.000000	1821.000000	1.000000	1.000000	8.000000	0.000000	0.000000	0.000000	4.500000
25%	64.000000	4.000000	4300.000000	199.000000	3.000000	13.000000	2.000000	2.000000	0.000000	6.440000
50%	128.000000	6.000000	5000.000000	299.000000	3.000000	48.000000	8.000000	2.000000	0.000000	6.500000
75%	128.000000	8.000000	5000.000000	469.000000	4.000000	64.000000	12.000000	5.000000	2.000000	6.590000
max	512.000000	16.000000	7000.000000	1999.000000	4.000000	108.000000	64.000000	48.000000	12.000000	7.600000

```
df.describe(include=np.object)
```

```
<ipython-input-55-f5c37ff7ec99>:1: DeprecationWarning: `np.object` is a deprecated alias for the builtin `object`. To silence this warning, use
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  df.describe(include=np.object)
```

	Brand	Model
count	407	407
unique	16	239
top	Samsung	Poco X3 Pro
freq	79	6

Shape of the data

1.Skewness

- Skewness helps us to study the shape of the data.
- It represents how much a distribution differs from a normal distribution, either to the left or to the right.
- The value of the skewness can be either positive, negative or zero.

Skewness

- Skewness helps us to study the shape of the data.
- It represents how much a distribution differs from a normal distribution, either to the left or to the right.
- The value of the skewness can be either positive, negative or zero.

```
df.skew()
```

```
<ipython-input-56-9e0b1e29546f>:1: FutureWarning: The default value of numeric_only in DataFrame.skew is deprecated. In a future version, it will defau
  df.skew()
Storage                  2.077271
RAM                      0.959169
Battery Capacity (mAh)  -0.818868
Price ($)                1.635034
n_cameras               -0.537236
res1                     0.395580
res2                     3.323150
res3                     4.370052
res4                     2.611113
screen                  -3.575040
dtype: float64
```

We find that Storage,RAM,Price,res2,res3,res4 are positively Skewed whereas screen size, no of cameras,Battery Capacity are neagtively skewed while res1 is symeteric

We find that Storage,RAM,Price,res2,res3,res4 are positively Skewed whereas screen size, no of cameras,Battery Capacity are neagtively skewed while res1 is symeteric

2.Kurtosis

- Kurtosis measures the peakedness of the distribution
- In other words, kurtosis is a statistical measure that defines how the tails of the distribution differ from the normal distribution
- Kurtosis identifies whether the tails of a given distribution contain extreme values

Kurtosis

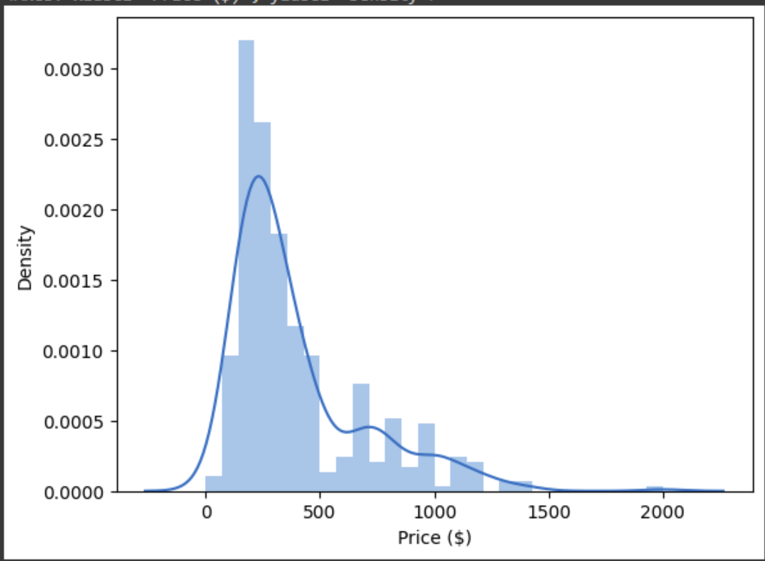
- Kurtosis measures the peakedness of the distribution
- In other words, kurtosis is a statistical measure that defines how the tails of the distribution differ from the normal distribution
- Kurtosis identifies whether the tails of a given distribution contain extreme values

```
[ ] df.kurt()
```

```
<ipython-input-57-8bd0d54cd88d>:1: FutureWarning: The default value of numeric_only in DataFrame.kurt is deprecated. In a future version, it will default to False. In additio
  df.kurt()
Storage                  8.809359
RAM                      0.915080
Battery Capacity (mAh)   2.113240
Price ($)                2.878530
n_cameras               -0.419465
res1                     0.241375
res2                    12.249120
res3                    30.069143
res4                     9.996952
screen                  17.879367
dtype: float64
```

```
sns.distplot(df['Price ($)'])
```

```
<ipython-input-58-576f466a8547>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Price ($)'])
<Axes: xlabel='Price ($)', ylabel='Density'>
```



The variable ‘prices’ is leptokurtic

The variable ‘prices’ is leptokurtic

Correlation

- It shows whether pairs of variables are related to each other
- If there is correlation, it shows how strong the correlation is
- Correlation takes values between -1 to +1, where values close to +1 represents strong positive correlation while values close to -1 represents strong negative correlation

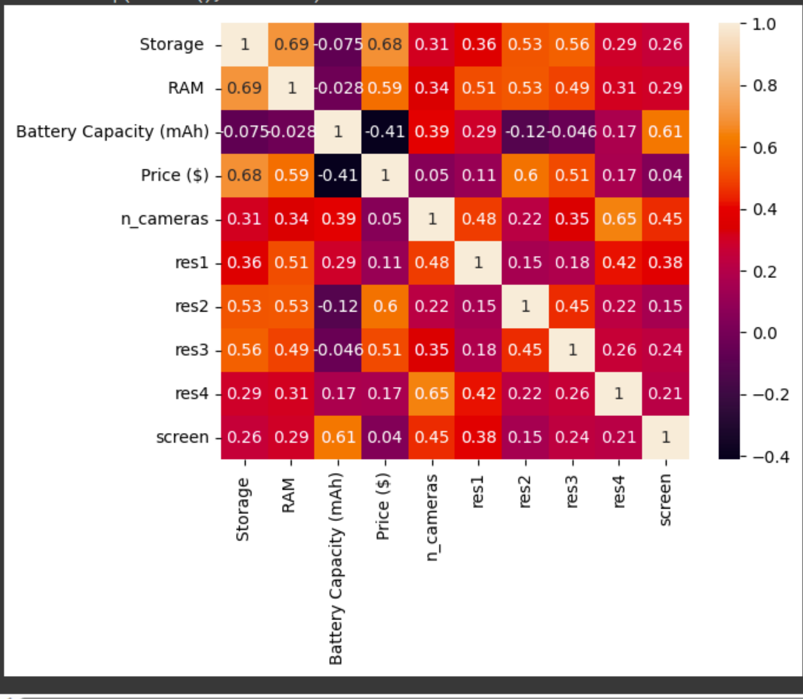
```
df.corr()
```

```
<ipython-input-59-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will defau
  df.corr()
```

	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
Storage	1.000000	0.691341	-0.074952	0.684934	0.307613	0.361980	0.526689	0.556139	0.289007	0.258582
RAM	0.691341	1.000000	-0.027627	0.590728	0.338579	0.511801	0.534353	0.489217	0.309690	0.294423
Battery Capacity (mAh)	-0.074952	-0.027627	1.000000	-0.411216	0.388638	0.290042	-0.115906	-0.045846	0.172771	0.609571
Price ($)	0.684934	0.590728	-0.411216	1.000000	0.050155	0.107906	0.598090	0.508667	0.166501	0.039514
n_cameras	0.307613	0.338579	0.388638	0.050155	1.000000	0.479489	0.221796	0.346142	0.646925	0.450435
res1	0.361980	0.511801	0.290042	0.107906	0.479489	1.000000	0.151836	0.179372	0.420375	0.382067
res2	0.526689	0.534353	-0.115906	0.598090	0.221796	0.151836	1.000000	0.453320	0.223225	0.154706
res3	0.556139	0.489217	-0.045846	0.508667	0.346142	0.179372	0.453320	1.000000	0.264716	0.238167
res4	0.289007	0.309690	0.172771	0.166501	0.646925	0.420375	0.223225	0.264716	1.000000	0.213166
screen	0.258582	0.294423	0.609571	0.039514	0.450435	0.382067	0.154706	0.238167	0.213166	1.000000

```
sns.heatmap(df.corr(),annot=True)
plt.show()
```

```
<ipython-input-60-f6412ee67fb3>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a fut
  sns.heatmap(df.corr(),annot=True)
```



We find screen size and Battery Capacity are strongly corelated

We find screen size and Battery Capacity are strongly corelated

Covariance

- It is the relationship between a pair of random variables where change in one variable causes change in another variable
- It can take any value between -infinity to +infinity, where the negative value represents the negative relationship whereas a positive value represents the positive relationship

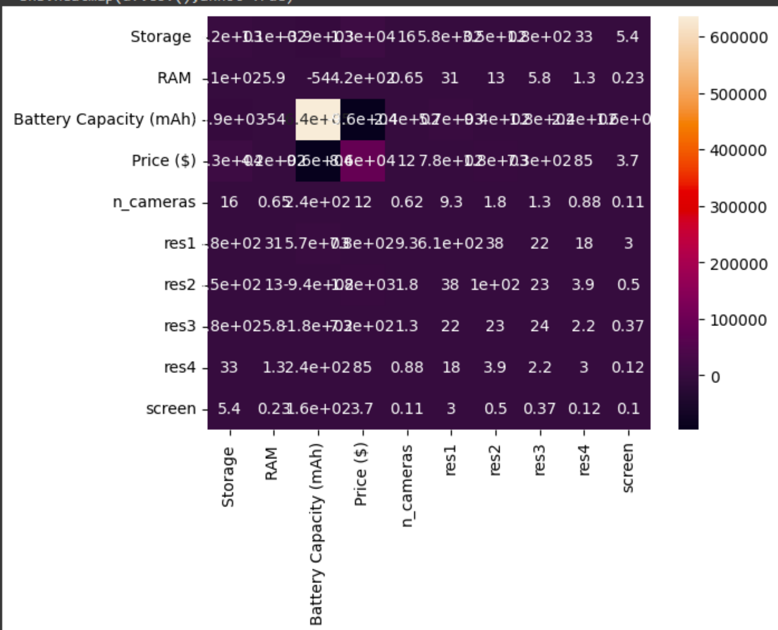
```
[ ] df.cov()
```

```
<ipython-input-61-6f98a29763d5>:1: FutureWarning: The default value of numeric_only in DataFrame.cov is deprecated. In a future version, it will default to False. Select
  df.cov()
```

	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
Storage	4220.212101	109.224338	-3881.613440	13042.602147	15.695186	580.174822	348.129604	177.135014	32.528292	5.385899
RAM	109.224338	5.914525	-53.562908	421.110172	0.646718	30.709160	13.222341	5.833311	1.304886	0.229575
Battery Capacity (mAh)	-3881.613440	-53.562908	635517.816572	-96091.054768	243.334673	5704.681062	-940.136230	-179.190660	238.627480	155.804883
Price ($)	13042.602147	421.110172	-96091.054768	85920.566212	11.546647	780.372551	1783.753313	731.031517	84.557582	3.713550
n_cameras	15.695186	0.646718	243.334673	11.546647	0.616865	9.291379	1.772431	1.332918	0.880309	0.113428
res1	580.174822	30.709160	5704.681062	780.372551	9.291379	608.714564	38.115358	21.697722	17.969245	3.022312
res2	348.129604	13.222341	-940.136230	1783.753313	1.772431	38.115358	103.523668	22.614027	3.935029	0.504686
res3	177.135014	5.833311	-179.190660	731.031517	1.332918	21.697722	22.614027	24.038489	2.248635	0.374393
res4	32.528292	1.304886	238.627480	84.557582	0.880309	17.969245	3.935029	2.248635	3.001731	0.118412
screen	5.385899	0.229575	155.804883	3.713550	0.113428	3.022312	0.504686	0.374393	0.118412	0.102798

```
sns.heatmap(df.cov(),annot=True)
plt.show()
```

```
<ipython-input-62-3653ad630287>:1: FutureWarning: The default value of numeric_only in DataFrame.cov is depre
  sns.heatmap(df.cov(),annot=True)
```



We find storage & Price and RAM & Price are strongly correlated

CONCLUSION:

The exploratory data analysis has been done using the given dataset and the results have been analysed using the above visualizations.

Ex No: 3

LINEAR REGRESSION

AIM:

To perform prediction with Linear regression using random linear regression dataset.

Dataset Description:

The dataset contains information on over 2,000 mobile phones from different brands. It includes details such as the storage capacity, RAM, screen size, camera specifications, battery capacity, and price of each device.

The dataset is structured as a CSV file with 7 columns:

- Brand: The brand name of the mobile phone.
- Model: The model name of the mobile phone.
- Storage: The amount of storage space available on the mobile phone in GB.
- RAM: The amount of random access memory available on the mobile phone in GB.
- Screen Size: The size of the mobile phone's screen in inches.
- Camera: The quality of the mobile phone's cameras, measured in megapixels.
- Battery Capacity: The amount of battery life the mobile phone has in mAh.
- Price: The price of the mobile phone in USD.

Problem Statement

The mobile phone price prediction problem is to develop a model that can predict the price of a mobile phone given a set of features. The target variable is the price of the mobile phone in USD. The goal of the problem is to develop a model that can accurately predict the price of a mobile phone given its features. This model can be used by a variety of stakeholders, including:

Mobile phone manufacturers: Manufacturers can use the model to develop a pricing strategy for their products. They can also use the model to identify the features that are most important to consumers and to determine how much they should charge for their phones based on those features.

Retailers: Retailers can use the model to set prices for mobile phones in their store. They can also use the model to compare the prices of different phones from different manufacturers and to ensure that they are charging a competitive price.

Consumers: Consumers can use the model to make informed decisions about which mobile phone to buy. They can use the model to compare the prices of different phones with different features and to find the best value for their money.

PROGRAMS WITH OUTPUT:

Linear Regression Using numpy

```
def mean(values):
    return sum(values)/float(len(values))



def variance(values,mean):
    return sum([(x-mean)**2 for x in values])



def covariance(x,mean_x,y,mean_y):
    covar=0.0
    for i in range(len(x)):
        covar+=(x[i]-mean_x) * (y[i]-mean_y)
    return covar



def coefficients(dataset):
    b1=covariance(x,mean_x,y,mean_y)/variance(x,mean_x)
    b0=mean_y-b1*mean_x
    return[b0,b1]



def simple_linear_regression(train,test):
    for row in test:
        ytest = b0 + b1 * row[0]
    return ytest
```

```
dataset=[[50,28],[60,40],[48,45],[70,50],[55,50],[60,38],[45,20]]
x=[row[0] for row in dataset]
y=[row[1] for row in dataset]
mean_x=mean(x)
mean_y=mean(y)


variance_x=variance(x,mean_x)
variance_y=variance(y,mean_y)
print('x stats:mean=%.3f variance=%.3f' % (mean_x,variance_x))
print('y stats:mean=%.3f variance=%.3f' % (mean_y,variance_y))


covar = covariance(x,mean_x,y,mean_y)
print('covariance: %.3f' % (covar))


b0,b1 = coefficients(dataset)
print('coefficients:b0=%.3f,b1=%.3f' % (b0,b1))
print('Regression equation of y on x : y=%.3f+%.3fx '% (b0, b1))


test=[[55]]
result=simple_linear_regression(dataset,test)
print('Value of y when x=55 is %.3f' % (result))
```

```
[33] def mean(values):
         return sum(values)/float(len(values))
```

```
[34] def variance(values,mean):
         return sum([(x-mean)**2 for x in values])
```

```
     def covariance(x,mean_x,y,mean_y):
         covar=0.0
         for i in range(len(x)):
             covar+=(x[i]-mean_x) * (y[i]-mean_y)
         return covar
```

```
[36] def coefficients(dataset):
         b1=covariance(x,mean_x,y,mean_y)/variance(x,mean_x)
         b0=mean_y-b1*mean_x
         return[b0,b1]
```

```
[37] def simple_linear_regression(train,test):
         for row in test:
             ytest = b0 + b1 * row[0]
         return ytest
```

```
[38] dataset=[[50,28],[60,40],[48,45],[70,50],[55,50],[60,38],[45,20]]
     x=[row[0] for row in dataset]
     y=[row[1] for row in dataset]
     mean_x=mean(x)
     mean_y=mean(y)
```


```
[38] dataset=[[50,28],[60,40],[48,45],[70,50],[55,50],[60,38],[45,20]]
     x=[row[0] for row in dataset]
     y=[row[1] for row in dataset]
     mean_x=mean(x)
     mean_y=mean(y)
```


```
[39] variance_x=variance(x,mean_x)
     variance_y=variance(y,mean_y)
     print('x stats:mean=%.3f variance=%.3f' % (mean_x,variance_x))
     print('y stats:mean=%.3f variance=%.3f' % (mean_y,variance_y))
```

```
x stats:mean=55.429 variance=447.714
y stats:mean=38.714 variance=761.429
```

```
[40] covar = covariance(x,mean_x,y,mean_y)
     print('covariance: %.3f' % (covar))
```

```
covariance: 368.857
```

```
[41] b0,b1 = coefficients(dataset)
     print('coefficients:b0=%.3f,b1=%.3f' % (b0,b1))
     print('Regression equation of y on x : y=%.3f+%.3fx '% (b0, b1))
```

```
coefficients:b0=-6.951,b1=0.824
Regression equation of y on x : y=-6.951+0.824x
```

```
[42] test=[[55]]
     result=simple_linear_regression(dataset,test)
     print('Value of y when x=55 is %.3f' % (result))
```

```
Value of y when x=55 is 38.361
```

Linear Regression Using Sklearn

```
from sklearn.model_selection import train_test_split
Xtrain,Xtest,ytrain,ytest=train_test_split(X,y,test_size=0.3,random_state=1)

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
Xtrain=scaler.fit_transform(Xtrain)
Xtest=scaler.transform(Xtest)

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(Xtrain, ytrain)
ypred = model.predict(Xtest)

r_sq = model.score(Xtrain, ytrain)
print('coefficient of determination:', r_sq)

print('intercept:', model.intercept_)
print('slope:', model.coef_)
```

```
[48] from sklearn.model_selection import train_test_split
     Xtrain,Xtest,ytrain,ytest=train_test_split(X,y,test_size=0.3,random_state=1)
```

```
[49] from sklearn.preprocessing import StandardScaler
     scaler=StandardScaler()
     Xtrain=scaler.fit_transform(Xtrain)
     Xtest=scaler.transform(Xtest)
```

```
[50] from sklearn.linear_model import LinearRegression
     model = LinearRegression()
     model.fit(Xtrain, ytrain)
     ypred = model.predict(Xtest)
```

```
r_sq = model.score(Xtrain, ytrain)
print('coefficient of determination:', r_sq)

coefficient of determination: 0.7474558066467858
```

```
[52] print('intercept:', model.intercept_)
     print('slope:', model.coef_)

intercept: 390.71478873239437
slope: [-114.38052398   79.67076712   63.48954106  -54.18321136  -13.71769667
    4.40063075   62.81454272   12.90156951   23.57564517   33.38604439]
```

CONCLUSION:

The predicted output is displayed using the linear regression model trained with the given dataset and results are verified.

Ex No: 4

PRINCIPAL COMPONENT ANALYSIS

AIM:

To perform Principal component analysis using Mobile dataset.

Dataset Description:

The dataset contains information on over 2,000 mobile phones from different brands. It includes details such as the storage capacity, RAM, screen size, camera specifications, battery capacity, and price of each device.

The dataset is structured as a CSV file with 7 columns:

- Brand: The brand name of the mobile phone.
- Model: The model name of the mobile phone.
- Storage: The amount of storage space available on the mobile phone in GB.
- RAM: The amount of random access memory available on the mobile phone in GB.
- Screen Size: The size of the mobile phone's screen in inches.
- Camera: The quality of the mobile phone's cameras, measured in megapixels.
- Battery Capacity: The amount of battery life the mobile phone has in mAh.
- Price: The price of the mobile phone in USD.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe.
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Load the training dataset and fit the data into the PCA model.
5. Display the heatmap for the correlation.
6. Load the test dataset and predict the value using the model
7. Display the results of the output predicted from the model.

PROGRAM AND OUTPUT:

Importing All required libraries and loading the dataset.

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
[2] from google.colab import drive
    drive.mount('/content/drive')
    p1 = '/content/drive/MyDrive/Colab Notebooks/MVT/Mobile.csv'
    df = pd.read_csv(p1)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dri
```

```
df.head()
```

	Brand	Model	Storage	RAM	Screen Size (inches)	Camera (MP)	Battery Capacity (mAh)	Price ($)
0	Apple	iPhone 13 Pro	128 GB	6 GB	6.1	12 + 12 + 12	3095	999
1	Samsung	Galaxy S21 Ultra	256 GB	12 GB	6.8	108 + 10 + 10 + 12	5000	1199
2	OnePlus	9 Pro	128 GB	8 GB	6.7	48 + 50 + 8 + 2	4500	899
3	Xiaomi	Redmi Note 10 Pro	128 GB	6 GB	6.67	64 + 8 + 5 + 2	5020	279
4	Google	Pixel 6	128 GB	8 GB	6.4	50 + 12.2	4614	799

Principal Component Analysis

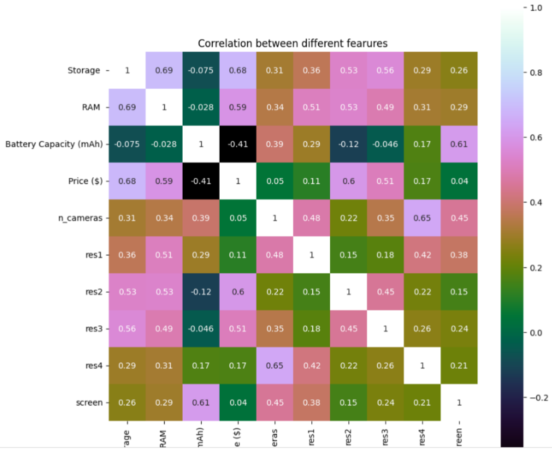
```
df.corr()
```

```
<ipython-input-35-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only
  df.corr()
```

	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
Storage	1.000000	0.691341	-0.074952	0.684934	0.307613	0.361980	0.526689	0.556139	0.289007	0.258582
RAM	0.691341	1.000000	-0.027627	0.590728	0.338579	0.511801	0.534353	0.489217	0.309690	0.294423
Battery Capacity (mAh)	-0.074952	-0.027627	1.000000	-0.411216	0.388638	0.290042	-0.115906	-0.045846	0.172771	0.609571
Price ($)	0.684934	0.590728	-0.411216	1.000000	0.050155	0.107906	0.598090	0.508667	0.166501	0.039514
n_cameras	0.307613	0.338579	0.388638	0.050155	1.000000	0.479489	0.221796	0.346142	0.646925	0.450435
res1	0.361980	0.511801	0.290042	0.107906	0.479489	1.000000	0.151836	0.179372	0.420375	0.382067
res2	0.526689	0.534353	-0.115906	0.598090	0.221796	0.151836	1.000000	0.453320	0.223225	0.154706
res3	0.556139	0.489217	-0.045846	0.508667	0.346142	0.179372	0.453320	1.000000	0.264716	0.238167
res4	0.289007	0.309690	0.172771	0.166501	0.646925	0.420375	0.223225	0.264716	1.000000	0.213166
screen	0.258582	0.294423	0.609571	0.039514	0.450435	0.382067	0.154706	0.238167	0.213166	1.000000

```python
correlation = df.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='cubehelix')

plt.title('Correlation between different fearures')
```

```
<ipython-input-36-effb445d3340>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  correlation = df.corr()
Text(0.5, 1.0, 'Correlation between different fearures')
```



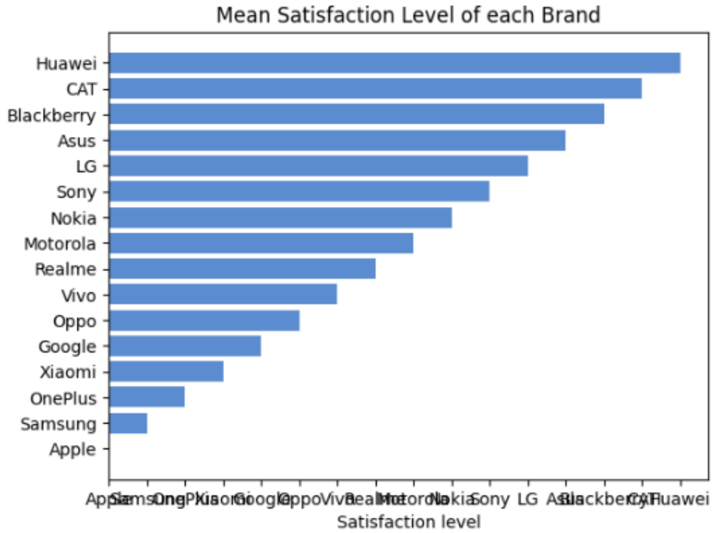

```python
groupby_brand=df.groupby('Brand').mean()
groupby_brand
```

```
<ipython-input-39-1ee09bfb17a7>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, nu
  groupby_brand=df.groupby('Brand').mean()
```

Brand	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
Apple	128.000000	3.966667	2863.900000	745.666667	2.066667	12.000000	9.033333	3.600000	0.000000	5.870000
Asus	160.000000	7.500000	5000.000000	874.000000	2.750000	64.000000	12.250000	6.250000	0.000000	6.505000
Blackberry	64.000000	5.333333	3500.000000	499.000000	2.000000	12.333333	10.000000	0.000000	0.000000	4.996667
CAT	32.000000	3.000000	4200.000000	299.000000	2.000000	13.000000	5.000000	0.000000	0.000000	5.500000
Google	128.000000	7.428571	4019.857143	699.000000	2.000000	22.857143	14.857143	0.000000	0.000000	6.100000
Huawei	218.666667	7.333333	4161.666667	783.166667	3.833333	48.833333	16.000000	12.166667	1.916667	6.523333
LG	170.666667	6.666667	4100.000000	615.666667	3.333333	58.666667	8.666667	6.333333	0.666667	6.776667
Motorola	105.739130	4.521739	5021.739130	278.130435	3.043478	55.521739	6.608696	2.782609	0.347826	6.591739
Nokia	74.285714	4.000000	4502.857143	244.714286	2.857143	31.428571	5.321429	1.357143	0.571429	6.533214
OnePlus	170.666667	8.666667	4415.000000	644.333333	3.533333	51.466667	26.266667	4.600000	1.266667	6.564000
Oppo	130.857143	6.928571	4631.339286	376.142857	3.375000	36.928571	6.428571	3.928571	0.803571	6.494107
Realme	96.744186	4.976744	5176.744186	206.906977	2.837209	37.139535	6.139535	1.534884	0.186047	6.510233
Samsung	130.835443	6.025316	4936.708861	465.240506	3.556962	51.367089	11.987342	5.063291	2.468354	6.558228
Sony	128.000000	8.000000	4500.000000	1.000000	3.000000	12.000000	12.000000	12.000000	0.000000	6.100000
Vivo	119.771429	6.828571	4750.000000	323.000000	2.628571	41.800000	6.657143	2.028571	0.457143	6.530286
Xiaomi	121.313433	5.626866	5101.791045	265.000000	3.567164	58.432836	7.850746	3.641791	1.283582	6.575224

```python
Brand_name=('Apple', 'Samsung', 'OnePlus', 'Xiaomi', 'Google', 'Oppo', 'Vivo',
        'Realme', 'Motorola', 'Nokia', 'Sony', 'LG', 'Asus', 'Blackberry',
        'CAT', 'Huawei')
Brand=('Apple', 'Samsung', 'OnePlus', 'Xiaomi', 'Google', 'Oppo', 'Vivo',
        'Realme', 'Motorola', 'Nokia', 'Sony', 'LG', 'Asus', 'Blackberry',
        'CAT', 'Huawei')
y_pos = np.arange(len(Brand))
x=np.arange(0,1,0.1)

plt.barh(y_pos, Brand, align='center', alpha=0.8)
plt.yticks(y_pos,Brand_name )
plt.xlabel('Satisfaction level')
plt.title('Mean Satisfaction Level of each Brand')
```

```
Text(0.5, 1.0, 'Mean Satisfaction Level of each Brand')
```



```
df.head()
```

	Brand	Model	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen
0	Apple	iPhone 13 Pro	128	6	3095	999	3	12	12	12	0	6.10
1	Samsung	Galaxy S21 Ultra	256	12	5000	1199	4	108	10	10	12	6.80
2	OnePlus	9 Pro	128	8	4500	899	4	48	50	8	2	6.70
3	Xiaomi	Redmi Note 10 Pro	128	6	5020	279	4	64	8	5	2	6.67
4	Google	Pixel 6	128	8	4614	799	2	50	12	0	0	6.40

```
[42] df_drop=df.copy()
```

```
[43] df_drop=df.drop(labels=['Brand','Model','res1','res2','res3','res4'],axis=1)
```

```
[44] cols = df_drop.columns.tolist()
     cols
```

```
['Storage ',
 'RAM ',
 'Battery Capacity (mAh)',
 'Price ($)',
 'n_cameras',
 'screen']
```

```
[45] df_drop.head()
```

	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	screen
0	128	6	3095	999	3	6.10
1	256	12	5000	1199	4	6.80
2	128	8	4500	899	4	6.70
3	128	6	5020	279	4	6.67
4	128	8	4614	799	2	6.40

```
[46] cols.insert(0, cols.pop(cols.index('screen')))
```

```
[47] cols
```

```
['screen',
 'Storage ',
 'RAM ',
 'Battery Capacity (mAh)',
 'Price ($)',
 'n_cameras']
```

```
[48] df_drop = df_drop.reindex(columns= cols)
```

```
[49] df_drop.head()
```

	screen	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras
0	6.10	128	6	3095	999	3
1	6.80	256	12	5000	1199	4
2	6.70	128	8	4500	899	4
3	6.67	128	6	5020	279	4
4	6.40	128	8	4614	799	2

+ Code + Text

```
[50] X = df_drop.iloc[:,1:6].values
     y = df_drop.iloc[:,0].values
     X
```

```
array([[ 128,    6, 3095,  999,    3],
       [ 256,   12, 5000, 1199,    4],
       [ 128,    8, 4500,  899,    4],
       ...,
       [ 128,    6, 3687, 1099,    3],
       [ 128,    8, 4025,  429,    4],
       [ 128,    6, 4500,  649,    3]])
```

```
[51] y
```

```
array([6.1 , 6.8 , 6.7 , 6.67, 6.4 , 6.1 , 6.7 , 6.67, 6.55, 6.78, 6.43,
       6.5 , 6.62, 5.4 , 6.7 , 6.55, 6.2 , 6.51, 6.5 , 6.43, 6.5 , 6.6 ,
       6.5 , 6.55, 6.67, 6.1 , 6.5 , 6.5 , 6.51, 6.5 , 6.7 , 6.5 , 4.7 ,
       6.5 , 6.58, 6.5 , 6.4 , 6.43, 6.5 , 6.1 , 6.52, 6.5 , 6.4 , 6.43,
       6.67, 5.4 , 6.44, 6.5 , 6.5 , 6.52, 6.53, 6.5 , 6.51, 6.5 , 6.5 ,
       6.51, 6.53, 6.7 , 6.58, 6.7 , 6.55, 6.5 , 4.7 , 6.58, 6.5 , 6.5 ,
       6.43, 6.43, 6.1 , 6.51, 6.5 , 6.6 , 6.43, 6.55, 6.1 , 6.56, 6.5 ,
       6.1 , 6.2 , 6.43, 6.4 , 6.67, 6.43, 6.78, 6.55, 6.7 , 6.67, 6.1 ,
       6.83, 6.67, 4.5 , 6.7 , 6.67, 6.5 , 6.43, 6.58, 6.8 , 6.39, 6.7 ,
       5.9 , 4.5 , 5.5 , 6.7 , 6.7 , 6.81, 6.52, 6.51, 6.5 , 6.5 , 6.5 ,
       6.5 , 6.5 , 6.78, 6.8 , 5.99, 6.55, 6.5 , 6.5 , 6.5 , 6.5 , 6.5 ,
       6.4 , 6.51, 6.67, 6.51, 6.4 , 6.55, 6.6 , 6.82, 6.4 , 6.67, 6.5 ,
       6.67, 6.5 , 6.58, 6.52, 6.8 , 6.5 , 6.62, 6.5 , 6.5 , 6.51, 6.43,
       6.5 , 6.5 , 6.5 , 6.5 , 6.4 , 6.8 , 6.58, 6.6 , 6.82, 6.53, 6.5 ,
       6.4 , 6.6 , 6.51, 6.6 , 6.6 , 6.52, 6.44, 6.43, 6.7 , 6.5 , 6.43,
       6.5 , 6.5 , 6.7 , 6.44, 6.67, 6.8 , 6.55, 6.5 , 6.4 , 6.67, 6.5 ,
       6.6 , 6.5 , 6.44, 6.52, 6.62, 6.4 , 6.55, 6.4 , 6.5 , 6.4 , 6.67,
       6.5 , 6.52, 6.7 , 6.44, 6.5 , 6.5 , 6.53, 6.5 , 6.5 , 6.6 , 6.58,
       6.5 , 6.5 , 6.4 , 6.43, 6.43, 6.4 , 6.58, 6.8 , 6.67, 6.5 , 6.7 ,
       6.5 , 6.5 , 6.5 , 6.5 , 6.67, 6.5 , 6.52, 6.52, 6.82, 6.53, 6.6 ,
       6.5 , 6.51, 6.55, 6.5 , 6.5 , 6.52, 6.52, 6.6 , 6.5 , 6.67, 6.5 ,
       6.43, 6.58, 6.5 , 6.67, 6.53, 6.5 , 6.5 , 6.51, 6.5 , 6.5 , 6.5 ,
       6.55, 6.58, 6.5 , 6.5 , 6.67, 6.5 , 6.52, 6.44, 6.5 , 6.39, 6.43,
       6.4 , 6.51, 6.44, 6.39, 6.67, 6.5 , 6.51, 6.51, 6.5 , 6.51, 6.53,
       6.5 , 6.52, 6.58, 6.5 , 6.55, 6.3 , 6.4 , 6.51, 6.51, 6.5 , 6.5 ,
       6.53, 6.4 , 6.22, 6.51, 6.5 , 6.  , 6.3 , 6.5 , 6.5 , 6.35, 6.5 ,
       6.53, 6.4 , 6.9 , 4.7 , 6.67, 6.  , 6.76, 6.49, 6.55, 6.7 , 6.81,
       6.5 , 6.81, 6.6 , 6.2 , 6.72, 6.5 , 6.55, 6.53, 6.1 , 6.52, 6.82,
       6.1 , 6.8 , 6.67, 6.55, 6.58, 6.7 , 6.4 , 5.7 , 6.5 , 6.39, 6.1 ,
       6.5 , 6.43, 6.72, 6.7 , 6.78, 6.3 , 6.67, 6.1 , 6.43, 6.6 , 6.7 ,
       6.53, 6.67, 6.  , 4.7 , 6.7 , 6.55, 6.39, 6.81, 6.5 , 6.5 , 6.67,
       6.5 , 6.47, 6.5 , 6.67, 6.3 , 6.67, 5.5 , 6.5 , 6.4 , 6.4 , 6.7 ,
       6.55, 6.81, 6.44, 6.1 , 6.5 , 6.43, 6.53, 6.5 , 6.5 , 6.58, 6.5 ,
       6.7 , 6.53, 6.39, 6.55, 6.9 , 6.67, 5.4 , 6.5 , 6.55, 6.8 , 7.6 ,
       6.67, 6.5 , 6.58, 6.5 , 6.5 , 4.7 , 6.53, 6.5 , 6.49, 6.39, 6.1 ,
       6.6 , 6.67, 6.2 , 6.1 , 6.53, 6.  , 6.4 , 6.9 , 6.55, 6.4 , 6.5 ,
       6.1 , 6.53, 6.5 , 6.43, 6.4 , 6.15, 6.7 , 6.57, 6.7 , 6.4 , 6.7 ])
```

```
[52] np.shape(X)
```

```
(407, 5)
```

```
[53] np.shape(y)
```

```
(407,)
```

```
[54] from sklearn.preprocessing import StandardScaler
     X_std = StandardScaler().fit_transform(X)
```

```
[55] mean_vec = np.mean(X_std, axis=0)
     cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
     print('Covariance matrix \n%s' %cov_mat)
```

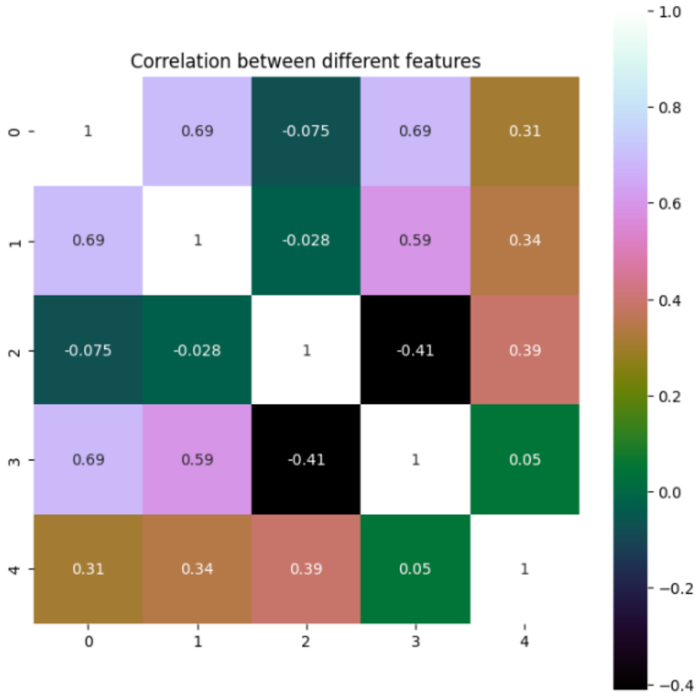
```
Covariance matrix
[[ 1.00246305  0.69304384 -0.07513627  0.68662074  0.30837039]
 [ 0.69304384  1.00246305 -0.02769547  0.59218253  0.33941296]
 [-0.07513627 -0.02769547  1.00246305 -0.41222935  0.38959507]
 [ 0.68662074  0.59218253 -0.41222935  1.00246305  0.05027831]
 [ 0.30837039  0.33941296  0.38959507  0.05027831  1.00246305]]
```

```
[56] print('NumPy covariance matrix: \n%s' %np.cov(X_std.T))
```

```
NumPy covariance matrix:
[[ 1.00246305  0.69304384 -0.07513627  0.68662074  0.30837039]
 [ 0.69304384  1.00246305 -0.02769547  0.59218253  0.33941296]
 [-0.07513627 -0.02769547  1.00246305 -0.41222935  0.38959507]
 [ 0.68662074  0.59218253 -0.41222935  1.00246305  0.05027831]
 [ 0.30837039  0.33941296  0.38959507  0.05027831  1.00246305]]
```

```python
plt.figure(figsize=(8,8))
sns.heatmap(cov_mat, vmax=1, square=True,annot=True,cmap='cubehelix')

plt.title('Correlation between different features')
```

```
Text(0.5, 1.0, 'Correlation between different features')
```



[58]
```
eig_vals, eig_vecs = np.linalg.eig(cov_mat)

print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.57496752  0.05693812 -0.20774105 -0.64233945  0.45870916]
 [ 0.55165468  0.12399195 -0.23998498  0.01903495 -0.78889035]
 [-0.13089013  0.70600629 -0.59845001  0.28781303  0.20843259]
 [ 0.5413362  -0.28958709  0.00208102  0.70778343  0.34947491]
 [ 0.23433385  0.63172659  0.7355978   0.05695589  0.04075612]]

Eigenvalues
[2.45664912 1.50923153 0.48782891 0.23167254 0.32693317]
```

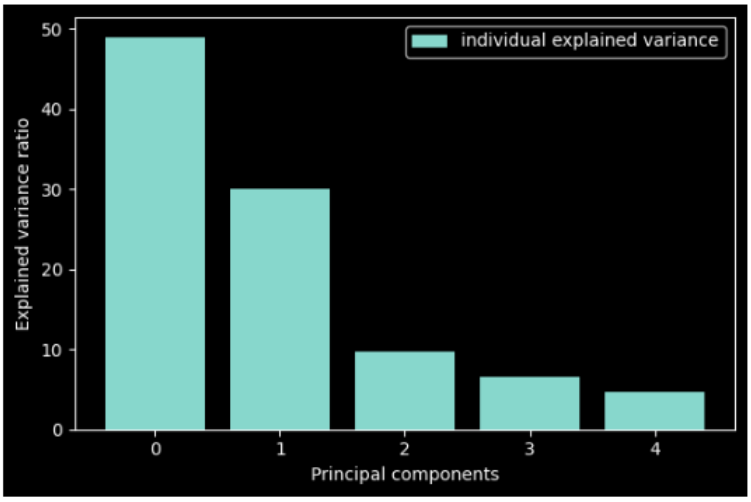
Selecting Principal Components

[59]
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])
```

```
Eigenvalues in descending order:
2.456649116958094
1.5092315336324813
0.4878289096527927
0.32693316904923 04
0.23167254164336587
```

[60]
```
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
```

```
with plt.style.context('dark_background'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(5), var_exp, align='center',
            label='individual explained variance')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.legend(loc='best')
    plt.tight_layout()
```



```
matrix_w = np.hstack((eig_pairs[0][1].reshape(5,1),
                      eig_pairs[1][1].reshape(5,1)
                    ))
print('Matrix W:\n', matrix_w)
```

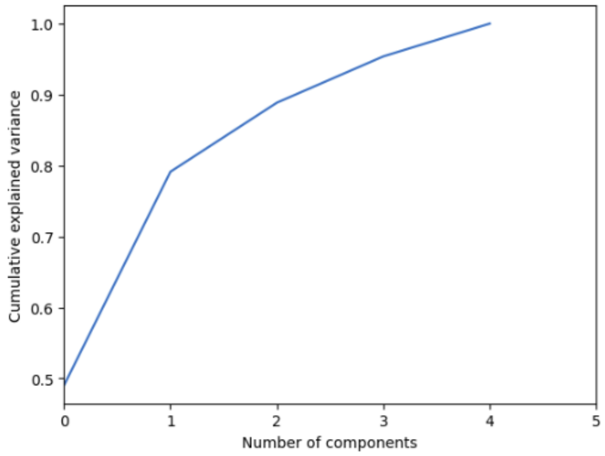
```
Matrix W:
 [[ 0.57496752  0.05693812]
 [ 0.55165468  0.12399195]
 [-0.13089013  0.70600629]
 [ 0.5413362  -0.28958709]
 [ 0.23433385  0.63172659]]
```

```
[63] Y = X_std.dot(matrix_w)
     Y
```

```
       [-4.90947615e-01, -7.06893985e-01],
       [ 7.86353554e-01, -2.96874361e+00],
       [-5.78499582e-01,  1.10382885e+00],
       [-9.81248964e-01,  1.60961416e-01],
       [ 3.29662161e-01, -3.05753645e+00],
       [-4.86046223e-01,  1.05437104e+00],
       [ 8.50175841e-01,  4.78575259e-01],
       [ 2.50794191e+00,  4.62628234e-01],
       [ 5.28256086e-01, -5.07652973e-01],
       [ 3.96946923e+00,  3.78935278e-01],
       [ 1.37926623e-01, -1.09761061e+00],
       [ 1.94211306e+00,  1.99576422e-01],
       [-1.72153099e-01, -1.94788636e+00],
       [ 8.87157185e-01,  4.58792137e-01],
       [-1.63449787e+00,  1.71516397e-01],
       [ 3.64716077e-01,  5.44955694e-01],
       [ 3.24756879e+00,  6.69658031e-02],
       [-2.11418576e-02,  1.27855405e+00],
       [ 8.45252280e-01, -1.63275288e+00],
       [-1.33953999e+00, -4.45578758e+00],
       [ 4.61927043e-02,  2.71225903e+00],
       [ 3.38476826e+00, -6.80772427e-01],
       [-1.27997575e+00, -6.44358231e-01],
```

```
from sklearn.decomposition import PCA
pca = PCA().fit(X_std)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,5,1)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

```
<ipython-input-64-225f5179fe1c>:4: MatplotlibDeprecationWarning: Passing the emit parameter of set_xlim() positionally is deprecated since
  plt.xlim(0,5,1)
Text(0, 0.5, 'Cumulative explained variance')
```



The above plot shows almost 90% variance by the first 4 components. Therfore we can drop 5th component

```
[65] from sklearn.decomposition import PCA
     sklearn_pca = PCA(n_components=4)
     Y_sklearn = sklearn_pca.fit_transform(X_std)
```

```
print(Y_sklearn)
```

```
[[ 1.40639852  2.09967736 -1.02511451  0.27676754]
 [ 4.25873123 -0.61558925  0.47019344  0.02239855]
 [ 1.7434752  -0.15247107 -0.70851106 -0.07241063]
 ...
 [ 1.49398595  1.67366393 -0.58086642  0.55111288]
 [ 0.9524992  -0.19618981 -1.06218959 -0.75780329]
 [ 0.52825609  0.50765297  0.03339915  0.2267668 ]]
```

\+ Code   + Text

```
[67] Y_sklearn.shape
```

```
(407, 4)
```

Thus Principal Component Analysis is used to remove the redundant features from the datasets without losing much information.These features are low dimensional in nature.The first component has the highest variance followed by second, third and so on.PCA works best on data set having 3 or higher dimensions. Because, with higher dimensions, it becomes increasingly difficult to make interpretations from the resultant cloud of data.

CONCLUSION:

The predicted output is displayed using the Principal Component Analysis model trained with the given dataset and results are verified. Thus the PCA is used to reduce the dimension of the dataset.

Ex No: 5

FACTOR ANALYSIS

AIM:

To perform Factor analysis using Mobile dataset.

Dataset Description:

The dataset contains information on over 2,000 mobile phones from different brands. It includes details such as the storage capacity, RAM, screen size, camera specifications, battery capacity, and price of each device.

The dataset is structured as a CSV file with 7 columns:

- Brand: The brand name of the mobile phone.
- Model: The model name of the mobile phone.
- Storage: The amount of storage space available on the mobile phone in GB.
- RAM: The amount of random access memory available on the mobile phone in GB.
- Screen Size: The size of the mobile phone's screen in inches.
- Camera: The quality of the mobile phone's cameras, measured in megapixels.
- Battery Capacity: The amount of battery life the mobile phone has in mAh.
- Price: The price of the mobile phone in USD.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe. (Dataset used : Personality data)
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Load the training dataset and fit the data into the Factor analysis model.
5. Display the scatterplot for the eigenvalues.
6. Load the test dataset and predict the value using the model
7. Display the results of the output predicted from the model.

PROGRAM AND OUTPUT:

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from factor_analyzer import FactorAnalyzer
```

```
from google.colab import drive
drive.mount('/content/drive')
p1 = '/content/drive/MyDrive/Colab Notebooks/MVT/Mobile.csv'
df = pd.read_csv(p1)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[3] df.head()
```

	Brand	Model	Storage	RAM	Screen Size (inches)	Camera (MP)	Battery Capacity (mAh)	Price ($)
0	Apple	iPhone 13 Pro	128 GB	6 GB	6.1	12 + 12 + 12	3095	999
1	Samsung	Galaxy S21 Ultra	256 GB	12 GB	6.8	108 + 10 + 10 + 12	5000	1199
2	OnePlus	9 Pro	128 GB	8 GB	6.7	48 + 50 + 8 + 2	4500	899
3	Xiaomi	Redmi Note 10 Pro	128 GB	6 GB	6.67	64 + 8 + 5 + 2	5020	279
4	Google	Pixel 6	128 GB	8 GB	6.4	50 + 12.2	4614	799

Factor Analysis

```
[35] df_drop=df.drop(labels=['Brand','Model','res1','res2','res3','res4'],axis=1)
```

```
[36] fa = FactorAnalyzer()
     fa.fit(df_drop)
     ev,v = fa.get_eigenvalues()
     ev
```

```
array([2.57543304, 1.97237333, 0.58540269, 0.35670897, 0.30352582,
       0.20655616])
```

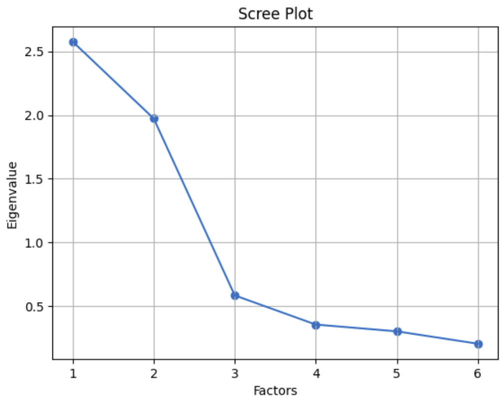
```
[37] v
```

```
array([ 2.27492014,  1.73706513,  0.16401668,  0.07545653, -0.01989282,
       -0.09327496])
```

```
[38] df_drop.shape[1]+1
```

```
7
```

```
plt.scatter(range(1,df_drop.shape[1]+1),ev)
plt.plot(range(1,df_drop.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```



```
fa = FactorAnalyzer(6,rotation='varimax')
fa.fit(df_drop)
fa.loadings_
```

```
array([[ 0.85813132,  0.12426218,  0.16653399, -0.09250024, -0.10376259,
         0.        ],
       [ 0.75189334,  0.16380394,  0.25533506, -0.00969152,  0.17015059,
         0.        ],
       [-0.24880011,  0.84109939,  0.08725445, -0.19440368, -0.01466264,
         0.        ],
       [ 0.86747679, -0.17121564, -0.11701422,  0.2124522 , -0.00905005,
         0.        ],
       [ 0.21038093,  0.47893856,  0.39703335, -0.01819104,  0.00229296,
         0.        ],
       [ 0.18625102,  0.79835789,  0.0777666 ,  0.11293926,  0.02848282,
         0.        ]])
```

```
[42] index_values = df_drop.columns[0:6]
```

```
[43] df_drop.shape
```

```
(407, 6)
```

```
[45] col_names=['Factor1','Factor2','Factor3','Factor4','Factor5','Factor6']
     df1=pd.DataFrame(fa.loadings_,index_values,col_names)
     df1.head()
```

	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
Storage	0.858131	0.124262	0.166534	-0.092500	-0.103763	0.0
RAM	0.751893	0.163804	0.255335	-0.009692	0.170151	0.0
Battery Capacity (mAh)	-0.248800	0.841099	0.087254	-0.194404	-0.014663	0.0
Price ($)	0.867477	-0.171216	-0.117014	0.212452	-0.009050	0.0
n_cameras	0.210381	0.478939	0.397033	-0.018191	0.002293	0.0

Factor1 has high factor loading for Storage, RAM, Price

Factor2 has high factor loading for n_camera i.e number of cameras

CONCLUSION:

The predicted output is displayed using the Factor Analysis model trained with the given dataset and results are verified. Thus the Factor Analysis is used to reduce the dimension of the dataset by selecting the most important factors in the dataset based on the latent features.

Ex No: 6

LINEAR PROGRAMMING

AIM:

To perform Linear programming in python for the given equations with the constraints and get the optimized values.

PROCEDURE:

1. Import the necessary library functions.
2. If pulp is not available use pip install method and install pulp library and import the entire package
3. Give the required constraints and maximization function to the model
4. View the model constraints and verify it.
5. Solve the equations using PULP_CBC_CMD ()
6. View the status of the model
7. Print the results which are calculated by the model
8. Get the optimized values of the given equation and constraints.

PROGRAM:

```
!pip install pulp

from pulp import *

import pandas as pd

import numpy as np

# Create a LP Maximization problem

# LpProblem - Function

# LpMaximize - Objective function is to Maximize

model = LpProblem("Problem", LpMaximize)



# Create problem Variables

x = pulp.LpVariable("x", lowBound = 0)   # Create a variable x >= 0

y = pulp.LpVariable("y", lowBound = 0)   # Create a variable y >= 0
```

```
# Objective Function

model += 2 * x + y


# Constraints:

model += (3 * x + 2 * y <= 12,"Constraint 1")

model += (x + 2.3 * y <= 6.9,"Constraint 2")

model += (x + 1.4 * y <= 4.9,"Constraint 3")

# Display the problem

print(model)

# Model.solve()

model.solve(PULP_CBC_CMD())

status =  LpStatus[model.status]

print(status)

print('Optimal value of x : ',pulp.value(x))

print('Optimal value of y : ',pulp.value(y))

print('Optimised Objective Function Value : ',pulp.value(model.objective))
```

OUTPUT

```
from pulp import *
import pandas as pd
import numpy as np
# Create a LP Maximization problem
# LpProblem - Function
# LpMaximize - Objective function is to Maximize
model = LpProblem("Problem", LpMaximize)

# Create problem Variables
x = pulp.LpVariable("x", lowBound = 0)   # Create a variable x >= 0
y = pulp.LpVariable("y", lowBound = 0)   # Create a variable y >= 0

# Objective Function
model += 2 * x + y

# Constraints:
model += (3 * x + 2 * y <= 12,"Constraint 1")
model += (x + 2.3 * y <= 6.9,"Constraint 2")
model += (x + 1.4 * y <= 4.9,"Constraint 3")
# Display the problem
print(model)
```

```
Problem:
MAXIMIZE
2*x + 1*y + 0
SUBJECT TO
Constraint_1: 3 x + 2 y <= 12

Constraint_2: x + 2.3 y <= 6.9

Constraint_3: x + 1.4 y <= 4.9

VARIABLES
x Continuous
y Continuous
```

```
[6] # Model.solve()
    model.solve(PULP_CBC_CMD())
    status =  LpStatus[model.status]
    print(status)
```

```
Optimal
```

```
[7] print('Optimal value of x : ',pulp.value(x))
    print('Optimal value of y : ',pulp.value(y))
    print('Optimised Objective Function Value : ',pulp.value(model.objective))
```

```
Optimal value of x :  4.0
Optimal value of y :  0.0
Optimised Objective Function Value :  8.0
```

```
[ ]
```

CONCLUSION:

Thus the Linear programming method using python was implemented and the results of various equations and optimized values was verified successfully.

Ex No: 07

TRANSPORTATION PROBLEM

AIM:

To perform Transportation problem in python for the given equations with the constraints and get the optimized values.

PROCEDURE:

1. Import the necessary library functions.
2. If pulp is not available use pip install method and install pulp library and import the entire package
3. Give the required constraints and minimization function to the model
4. View the model constraints and verify it.
5. Solve the equations using PULP_CBC_CMD()
6. View the status of the model
7. Print the results which are calculated by the model
8. Get the optimized values of the given equation and constraints.

PROGRAM:

```
!pip install pulp

from pulp import *

# Creates a list of all the supply nodes

supply_nodes = ["S1","S2","S3"]


# Creates a dictionary for the number of units of supply for each supply node

supply = {"S1": 11,

    "S2": 13,

    "S3": 19}


# Creates a list of all demand nodes

demand_nodes = ["D1","D2","D3" ,"D4"]


# Creates a dictionary for the number of units of demand for each demand node

demand = {"D1": 6,

    "D2": 10,
```

```
    "D3": 12,
    "D4":15}
# Creates a list of costs of each transportation path
costs = [# Demand
    #D1 D2 D3 D4
    [21,16,25,13], #S1
    [17,18,14,23], #S2  Supply
    [32,27,18,41]  #S3
    ]

costs = makeDict((supply_nodes, demand_nodes),costs)
print(costs)
# Creates the prob variable to contain the problem data
prob = LpProblem("Product Distribution Problem",LpMinimize)
# Creates a list of tuples containing all the possible routes for transport
Routes = [(s,d) for s in supply_nodes for d in demand_nodes]
# A dictionary called route_vars is created to contain the referenced variables (the routes)
route_vars = LpVariable.dicts("Route",(supply_nodes,demand_nodes),0,None,LpInteger)
# The objective function is added to prob first
prob += lpSum([route_vars[s][d]*costs[s][d] for (s,d) in Routes]), "Sum of Transporting Costs"
# The supply maximum constraints are added to prob for each supply node (warehouse)
for s in supply_nodes:
   prob += lpSum([route_vars[s][d] for d in demand_nodes]) <= supply[s], "Sum of Products out of supply %s"%s

# The demand minimum constraints are added to prob for each demand node (bar)
for d in demand_nodes:
```

```
    prob += lpSum([route_vars[s][d] for s in supply_nodes]) >= demand[d], "Sum of Products into demand %s"%d

prob.solve()

print("Status:",LpStatus[prob.status] )

total=0

for v in prob.variables():

    if(v.varValue != None):

        if (v.varValue > 0):

            print(v.name, "=", v.varValue)

            total+=v.varValue


total=0

for v in prob.variables():

    if(v.varValue != None):

        if (v.varValue > 0):

            print(v.name, "=", v.varValue)

            total+=v.varValue


print("Optimal Cost is = ",(13*11)+(17*6)+(18*3)+(23*4)+(27*7)+(18*12))
```

OUTPUT:

```python
from pulp import *
# Creates a list of all the supply nodes
supply_nodes = ["S1","S2","S3"]

# Creates a dictionary for the number of units of supply for each supply node
supply = {"S1": 11,
      "S2": 13,
      "S3": 19}

# Creates a list of all demand nodes
demand_nodes = ["D1","D2","D3" ,"D4"]

# Creates a dictionary for the number of units of demand for each demand node
demand = {"D1": 6,
      "D2": 10,
      "D3": 12,
      "D4":15}
# Creates a list of costs of each transportation path
costs = [# Demand
     #D1 D2 D3 D4
     [21,16,25,13], #S1
     [17,18,14,23], #S2  Supply
     [32,27,18,41]  #S3
     ]
costs = makeDict((supply_nodes, demand_nodes),costs)
print(costs)
# Creates the prob variable to contain the problem data
prob = LpProblem("Product Distribution Problem",LpMinimize)
# Creates a list of tuples containing all the possible routes for transport
Routes = [(s,d) for s in supply_nodes for d in demand_nodes]
# A dictionary called route_vars is created to contain the referenced variables (the routes)
route_vars = LpVariable.dicts("Route",(supply_nodes,demand_nodes),0,None,LpInteger)
# The objective function is added to prob first
prob += lpSum([route_vars[s][d]*costs[s][d] for (s,d) in Routes]), "Sum of Transporting Costs"
# The supply maximum constraints are added to prob for each supply node (warehouse)
for s in supply_nodes:
    prob += lpSum([route_vars[s][d] for d in demand_nodes]) <= supply[s], "Sum of Products out of supply %s"%s

# The demand minimum constraints are added to prob for each demand node (bar)
for d in demand_nodes:
    prob += lpSum([route_vars[s][d] for s in supply_nodes]) >= demand[d], "Sum of Products into demand %s"%d
prob.solve()
```

```
{'S1': {'D1': 21, 'D2': 16, 'D3': 25, 'D4': 13}, 'S2': {'D1': 17, 'D2': 18, 'D3': 14, 'D4': 23}, 'S3': {'D1': 32, 'D2': 27, 'D3': 18, 'D4': 41}}
/usr/local/lib/python3.10/dist-packages/pulp/pulp.py:1352: UserWarning: Spaces are not permitted in the name. Converted to '_'
  warnings.warn("Spaces are not permitted in the name. Converted to '_'")
1
```

```python
print("Status:",LpStatus[prob.status] )
```

```
Status: Optimal
```

```python
total=0
for v in prob.variables():
    if(v.varValue != None):
        if (v.varValue > 0):
            print(v.name, "=", v.varValue)
            total+=v.varValue
```

```
Route_S1_D4 = 11.0
Route_S2_D1 = 6.0
Route_S2_D2 = 3.0
Route_S2_D4 = 4.0
Route_S3_D2 = 7.0
Route_S3_D3 = 12.0
```

```python
print("Optimal Cost is = ",(13*11)+(17*6)+(18*3)+(23*4)+(27*7)+(18*12))
```

```
Optimal Cost is =  796
```

CONCLUSION:

Thus the transportation problem method using python was implemented and the results of various equations and optimized values was verified successfully.

Ex No: 08

ASSIGNMENT PROBLEM

AIM:

To perform assignment problem in python for the given cost matrix and get the optimized cost.

PROCEDURE:

9. Import the necessary library functions.
10. If scipy is not available use pip install method and install scipy library and import the entire package
11. Create a cost matrix and pass it to function to solve it
12. Solve the equations using linear_sum_assignment ()
13. Extract the optimal assignment values
14. Print the optimal assignment values
15. Calculate the total cost

PROGRAM:

```
import numpy as np

from scipy.optimize import linear_sum_assignment


# Create a cost matrix

cost_matrix = np.array([

    [10, 11,  4,  2,  8],

    [ 7, 11, 10, 14, 12],

    [ 5,  6,  9, 12, 14],

    [ 13, 15, 11, 10, 7]

])


# Solve the assignment problem

row_indices, col_indices = linear_sum_assignment(cost_matrix)


# Extract the optimal assignment

assignment = [(row, col) for row, col in zip(row_indices, col_indices)]
```

```
print("Optimal Assignment:")

for row, col in assignment:

   print(f"Resource {row} assigned to Task {col}")


print("Optimal Cost is = ",2+7+6+7)
```

OUTPUT:

```python
import numpy as np
from scipy.optimize import linear_sum_assignment

# Create a cost matrix
cost_matrix = np.array([
    [10, 11,  4,  2,  8],
    [ 7, 11, 10, 14, 12],
    [ 5,  6,  9, 12, 14],
    [ 13, 15, 11, 10, 7]
])

# Solve the assignment problem
row_indices, col_indices = linear_sum_assignment(cost_matrix)

# Extract the optimal assignment
assignment = [(row, col) for row, col in zip(row_indices, col_indices)]

print("Optimal Assignment:")
for row, col in assignment:
    print(f"Resource {row} assigned to Task {col}")
```

```
Optimal Assignment:
Resource 0 assigned to Task 3
Resource 1 assigned to Task 0
Resource 2 assigned to Task 1
Resource 3 assigned to Task 4
```

```
[ ] print("Optimal Cost is = ",2+7+6+7)
```

```
Optimal Cost is =  22
```

CONCLUSION:

Thus the assignment problem method using python was implemented and the results of cost matrix was verified successfully.

Ex No: 9

HIERARCHICAL CLUSTERING

AIM:

To perform Hierarchical Clustering using agglomerative clustering with four types of linkage methods ward ,single ,average ,complete using sklearn make blobs dataset.

Dataset Description:

The dataset contains information on over 2,000 mobile phones from different brands. It includes details such as the storage capacity, RAM, screen size, camera specifications, battery capacity, and price of each device.

The dataset is structured as a CSV file with 7 columns:

- Brand: The brand name of the mobile phone.
- Model: The model name of the mobile phone.
- Storage: The amount of storage space available on the mobile phone in GB.
- RAM: The amount of random access memory available on the mobile phone in GB.
- Screen Size: The size of the mobile phone's screen in inches.
- Camera: The quality of the mobile phone's cameras, measured in megapixels.
- Battery Capacity: The amount of battery life the mobile phone has in mAh.
- Price: The price of the mobile phone in USD.

PROCEDURE:

1. Import the necessary library functions.
2. Load the required dataset into the dataframe. (Dataset used : sklearn inbuilt makeblob dataset)
3. Load the training dataset and fit the data into the hierarchical clustering ,agglomerative clustering model.
4. Display the scatterplot for the two columns.
5. Display the dendogram using agglomerative model.
6. Fit and predict the point in agglomerative clustering model.
7. Display the scatter plot of the clusters.

PROGRAM AND OUTPUT;

```
[2] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from scipy.cluster.hierarchy import dendrogram, linkage
    import scipy.cluster.hierarchy as sch
    from sklearn.cluster import AgglomerativeClustering
```

```
[3] from google.colab import drive
    drive.mount('/content/drive')
    p1 = '/content/drive/MyDrive/Colab Notebooks/MVT/Mobile.csv'
    df = pd.read_csv(p1)
```

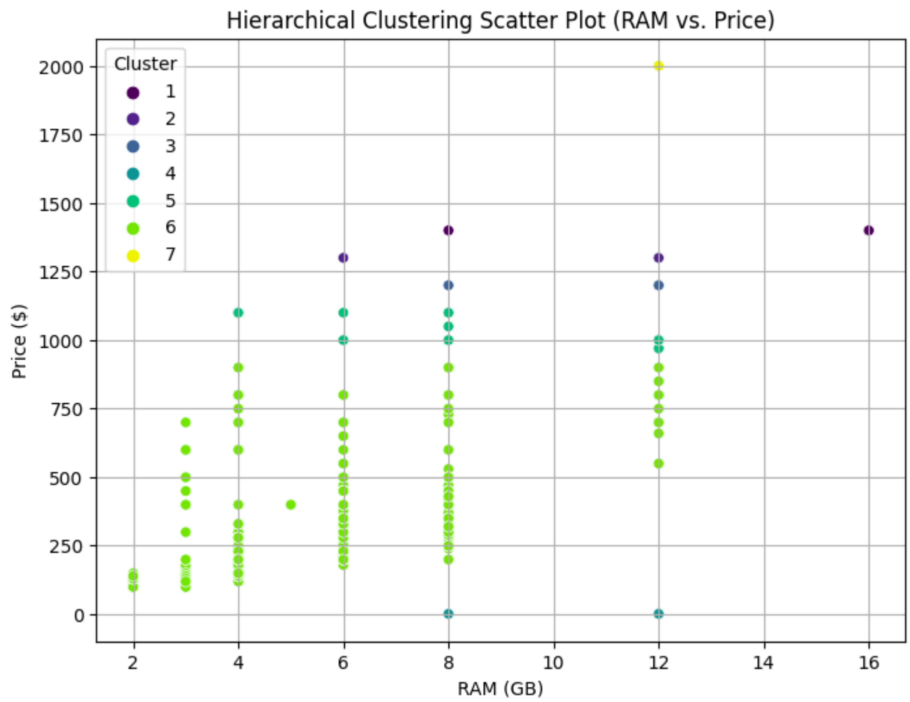
```
Mounted at /content/drive
```

```
df.head()
```

	Brand	Model	Storage	RAM	Screen Size (inches)	Camera (MP)	Battery Capacity (mAh)	Price ($)
0	Apple	iPhone 13 Pro	128 GB	6 GB	6.1	12 + 12 + 12	3095	999
1	Samsung	Galaxy S21 Ultra	256 GB	12 GB	6.8	108 + 10 + 10 + 12	5000	1199
2	OnePlus	9 Pro	128 GB	8 GB	6.7	48 + 50 + 8 + 2	4500	899
3	Xiaomi	Redmi Note 10 Pro	128 GB	6 GB	6.67	64 + 8 + 5 + 2	5020	279
4	Google	Pixel 6	128 GB	8 GB	6.4	50 + 12.2	4614	799

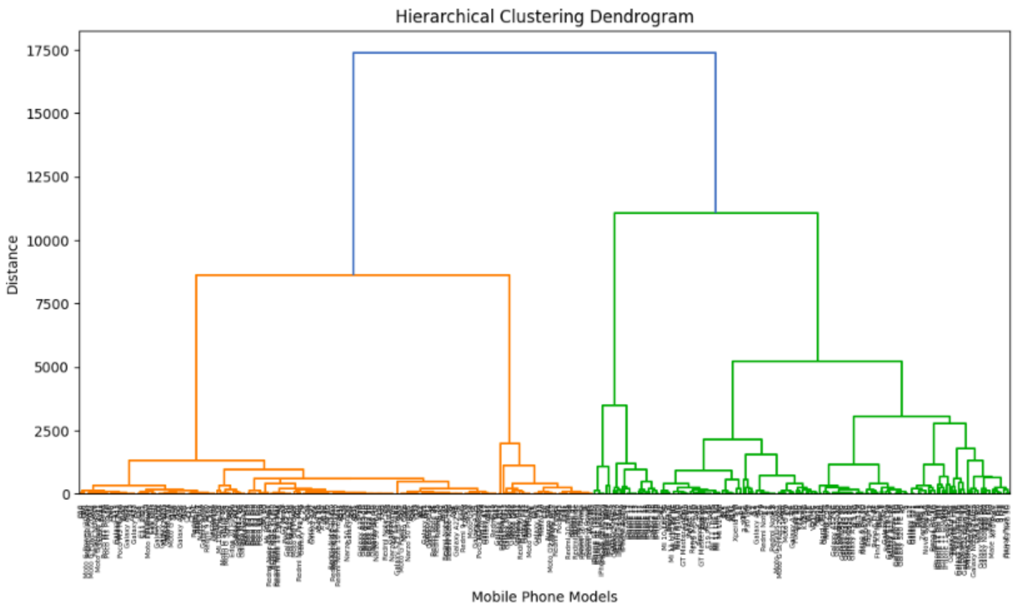
HIERARCHICAL CLUSTERING

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you already have the 'df' DataFrame and 'clusters' variable.

ram = df["RAM "]
price = df["Price ($)"]

# Add the 'clusters' column to the DataFrame with cluster assignments
df["Cluster"] = clusters

# Create a scatter plot without the legend
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x="RAM ", y="Price ($)", hue="Cluster", palette="viridis")
plt.title("Hierarchical Clustering Scatter Plot (RAM vs. Price)")
plt.xlabel("RAM (GB)")
plt.ylabel("Price ($)")
plt.grid(True)
plt.show()
```



Ward Linkage

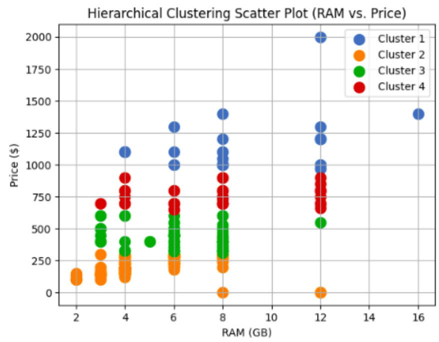
```
features = df[['Storage ', 'RAM ', 'Battery Capacity (mAh)', 'Price ($)']]
Z = linkage(features, method='ward')

# Create a dendrogram to visualize the hierarchical clustering
plt.figure(figsize=(12, 6))
dendrogram(Z, labels=df["Model"].tolist(), orientation='top', leaf_rotation=90)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Mobile Phone Models")
plt.ylabel("Distance")
plt.show()
```



```
[50] features = df[["RAM ", "Price ($)"]]

n_clusters = 4

hc = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='ward')

y_hc = hc.fit_predict(features)
```

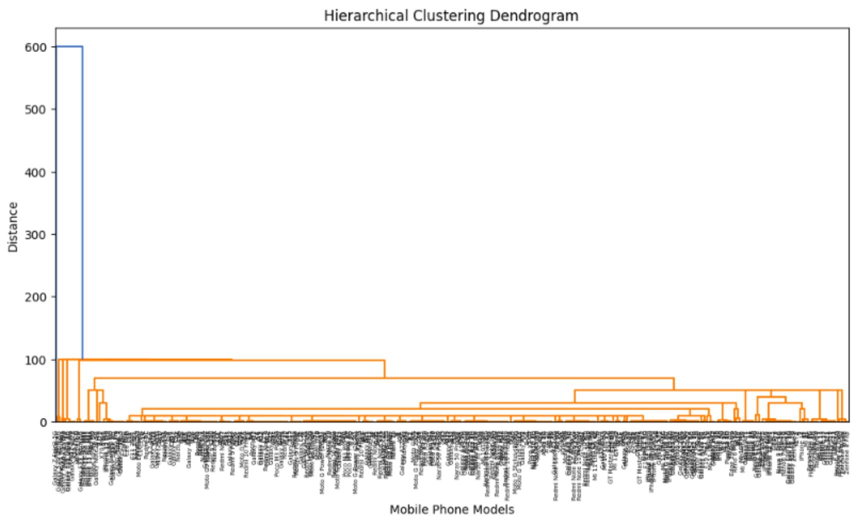
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

```
# Create scatter plots to visualize the clusters
for cluster_label in range(n_clusters):
    plt.scatter(features[y_hc == cluster_label]["RAM "], features[y_hc == cluster_label]["Price ($)"], s=100, label=f'Cluster {cluster_label + 1}')

# Customize the plot
plt.title("Hierarchical Clustering Scatter Plot (RAM vs. Price)")
plt.xlabel("RAM (GB)")
plt.ylabel("Price ($)")
plt.legend()
plt.grid(True)
plt.show()
```



Single Linkage

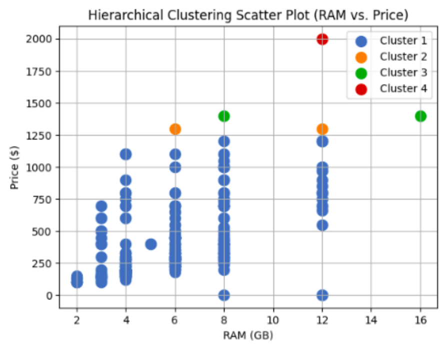
```
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

Z = linkage(features, method='single', metric='euclidean')
threshold = 50
clusters = fcluster(Z, threshold, criterion='distance')
plt.figure(figsize=(12, 6))
dendrogram(Z, labels= df["Model"].tolist(), orientation='top', leaf_rotation=90)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Mobile Phone Models")
plt.ylabel("Distance")
plt.show()
```



```
[53] features = df[["RAM ", "Price ($)"]]

n_clusters = 4

hc = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='single')

y_hc = hc.fit_predict(features)
```

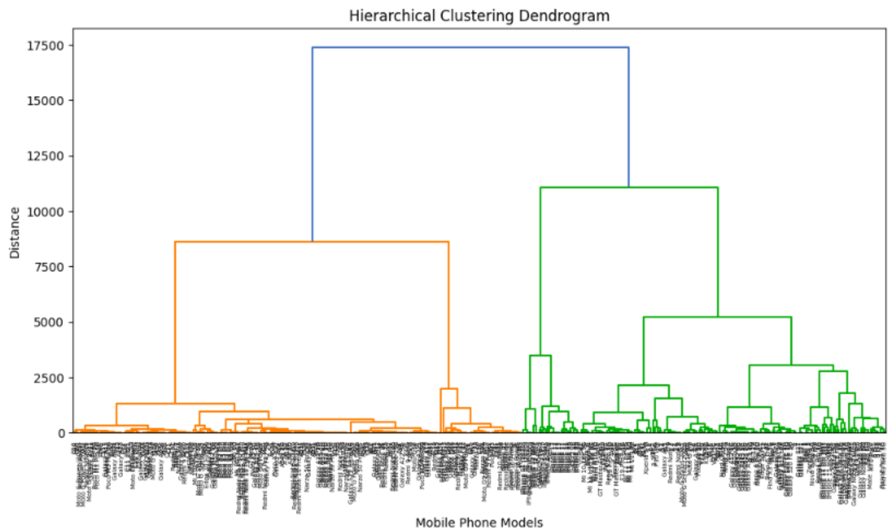
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

```
# Create scatter plots to visualize the clusters
for cluster_label in range(n_clusters):
    plt.scatter(features[y_hc == cluster_label]["RAM "], features[y_hc == cluster_label]["Price ($)"], s=100, label=f'Cluster {cluster_label + 1}')

# Customize the plot
plt.title("Hierarchical Clustering Scatter Plot (RAM vs. Price)")
plt.xlabel("RAM (GB)")
plt.ylabel("Price ($)")
plt.legend()
plt.grid(True)
plt.show()
```



Complete Linkage

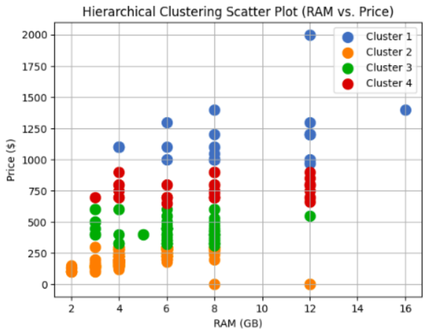
```
features = df[['Storage ', 'RAM ', 'Battery Capacity (mAh)', 'Price ($)']]
Z = linkage(features, method='complete')

# Create a dendrogram to visualize the hierarchical clustering
plt.figure(figsize=(12, 6))
dendrogram(Z, labels=df["Model"].tolist(), orientation='top', leaf_rotation=90)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Mobile Phone Models")
plt.ylabel("Distance")
plt.show()
```



```python
features = df[["RAM ", "Price ($)"]]

n_clusters = 4

hc = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean', linkage='complete')

y_hc = hc.fit_predict(features)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be removed in 1.4. Use `metric` instead
  warnings.warn(
```

```python
# Create scatter plots to visualize the clusters
for cluster_label in range(n_clusters):
    plt.scatter(features[y_hc == cluster_label]["RAM "], features[y_hc == cluster_label]["Price ($)"], s=100, label=f'Cluster {cluster_label + 1}')

# Customize the plot
plt.title("Hierarchical Clustering Scatter Plot (RAM vs. Price)")
plt.xlabel("RAM (GB)")
plt.ylabel("Price ($)")
plt.legend()
plt.grid(True)
plt.show()
```



Cluster Evaluation

```python
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

# Select the "RAM" and "Price" columns as features
x = df[["RAM ", "Price ($)"]]

# Create a KMeans clustering model
n_clusters = 3
km = KMeans(n_clusters=n_clusters, random_state=42)

# Fit the model and get cluster assignments
cluster_labels = km.fit_predict(x)

# Calculate the silhouette score
score = silhouette_score(x, cluster_labels, metric='euclidean')

print('Silhouette Score: %.3f' % score)
```

```
Silhouette Score: 0.681
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

CONCLUSION:

Thus the given dataset is clustered using the hierarchical clustering with agglomerative clustering method with 4 different types of linkage methods called as wards, single, complete ,average linkage methods and results were verified.

Ex No: 10

K-MEANS CLUSTERING

AIM:

To perform Non-hierarchical clustering using K-Means algorithm using given dataset.

Dataset Description:

The dataset contains information on over 2,000 mobile phones from different brands. It includes details such as the storage capacity, RAM, screen size, camera specifications, battery capacity, and price of each device.

The dataset is structured as a CSV file with 7 columns:

- Brand: The brand name of the mobile phone.
- Model: The model name of the mobile phone.
- Storage: The amount of storage space available on the mobile phone in GB.
- RAM: The amount of random access memory available on the mobile phone in GB.
- Screen Size: The size of the mobile phone's screen in inches.
- Camera: The quality of the mobile phone's cameras, measured in megapixels.
- Battery Capacity: The amount of battery life the mobile phone has in mAh.
- Price: The price of the mobile phone in USD.

PROCEDURE:

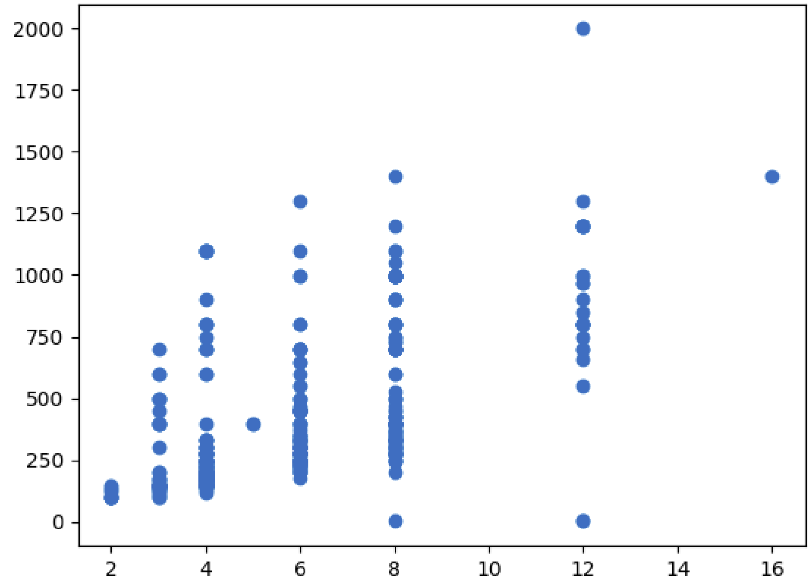
1. Import the necessary library functions.
2. Load the required dataset into the dataframe. (Dataset used : Iris dataset)
3. Print the head and shape of the dataset to find the dimensions of the given data.
4. Load the training dataset and fit the data into the K-Means clustering model.
5. Display the scatterplot for the two columns.
6. Using min-maxscaler find the number of cluster required and plot the graph.
7. With the help of the elbow diagram , find the number of clusters needed and do the k-means clustering.

PROGRAM AND OUTPUT:

K-MEANS CLUSTERING

```
plt.scatter(df['RAM '],df['Price ($)'])
```

```
<matplotlib.collections.PathCollection at 0x78f236db2b60>
```




```
[ ] km=KMeans(n_clusters=3)
```

```
[ ] km
```

```
KMeans
KMeans(n_clusters=3)
```

```
y_predicted = km.fit_predict(df[['RAM ','Price ($)']])
y_predicted
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'au
  warnings.warn(
array([2, 2, 2, 0, 1, 1, 2, 0, 1, 2, 0, 1, 0, 1, 2, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 2, 1, 1, 1, 0,
       2, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 2, 2, 0, 0, 0, 0, 1, 0,
       0, 0, 2, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 2, 0, 0, 1, 2, 0, 1, 1, 1, 2, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 2, 0, 2, 2, 2, 0, 1, 0, 0, 1, 0, 0, 1, 1, 2, 0, 1, 1, 0, 0, 0,
       2, 0, 1, 0, 1, 1, 0, 0, 2, 0, 0, 0, 2, 1, 1, 0, 0, 1, 0, 0, 0, 2,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 2, 2, 2, 0, 0, 1, 2, 0, 2, 0, 1, 0, 2,
       1, 1, 2, 1, 0, 1, 0, 0, 0, 0, 2, 0, 0, 1, 1, 0, 1, 0, 2, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 2, 0, 2, 0, 1], dtype=int32)
```

```
[40] df['cluster'] = y_predicted
```

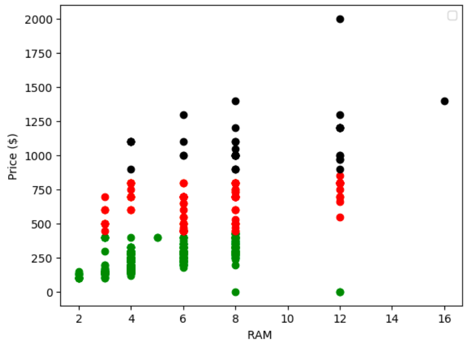
```
[41] df.head()
```

	Brand	Model	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen	cluster
0	Apple	iPhone 13 Pro	128	6	3095	999	3	12	12	12	0	6.10	2
1	Samsung	Galaxy S21 Ultra	256	12	5000	1199	4	108	10	10	12	6.80	2
2	OnePlus	9 Pro	128	8	4500	899	4	48	50	8	2	6.70	2
3	Xiaomi	Redmi Note 10 Pro	128	6	5020	279	4	64	8	5	2	6.67	0
4	Google	Pixel 6	128	8	4614	799	2	50	12	0	0	6.40	1

```
[42] df1 = df[df.cluster==0]
     df2 = df[df.cluster==1]
     df3 = df[df.cluster==2]
```

```
plt.scatter(df1['RAM '],df1['Price ($)'],color='green')
plt.scatter(df2['RAM '],df2['Price ($)'],color='red')
plt.scatter(df3['RAM '],df3['Price ($)'],color='black')
plt.xlabel('RAM ')
plt.ylabel('Price ($)')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
<matplotlib.legend.Legend at 0x78f1f871fdf0>
```




```
scaler = MinMaxScaler()
scaler.fit(df[['Price ($)']])
df['Price ($)']=scaler.transform(df['Price ($)'].values.reshape(-1,1))
df
scaler.fit(df[['RAM ']])
df.SepalLengthCm = scaler.transform(df['RAM '].values.reshape(-1,1))
df
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MinMaxScaler was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MinMaxScaler was fitted with feature names
  warnings.warn(
<ipython-input-48-32215442e1c1>:6: UserWarning: Pandas doesn't allow columns to be created via a new attribute name - see https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access
  df.SepalLengthCm = scaler.transform(df['RAM '].values.reshape(-1,1))
```

	Brand	Model	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen	cluster
0	Apple	iPhone 13 Pro	128	6	3095	0.499499	3	12	12	12	0	6.10	2
1	Samsung	Galaxy S21 Ultra	256	12	5000	0.599600	4	108	10	10	12	6.80	2
2	OnePlus	9 Pro	128	8	4500	0.449449	4	48	50	8	2	6.70	2
3	Xiaomi	Redmi Note 10 Pro	128	6	5020	0.139139	4	64	8	5	2	6.67	0
4	Google	Pixel 6	128	8	4614	0.399399	2	50	12	0	0	6.40	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
402	Samsung	Galaxy Note20 5G	128	8	4300	0.524525	3	12	64	12	0	6.70	2
403	Xiaomi	Mi 10 Lite 5G	128	6	4160	0.174174	4	48	8	2	2	6.57	0
404	Apple	iPhone 12 Pro Max	128	6	3687	0.549550	3	12	12	12	0	6.70	2
405	Oppo	Reno3	128	8	4025	0.214214	4	48	13	8	2	6.40	0
406	Samsung	Galaxy S10 Lite	128	6	4500	0.324324	3	48	12	5	0	6.70	1

407 rows × 13 columns

```
km=KMeans(n_clusters=3)
y_predicted=km.fit_predict(df[['RAM ','Price ($)']])
y_predicted
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init
  warnings.warn(
array([1, 2, 1, 1, 1, 0, 1, 1, 1, 2, 1, 1, 1, 0, 2, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 2, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 2, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 2, 2, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 2, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 2, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 2, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       2, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 2, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 2, 0, 1, 0, 0,
       0, 2, 1, 2, 1, 2, 0, 1, 0, 0, 0, 0, 1, 1, 1, 2, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 2, 2, 1, 0, 1, 1, 0, 2,
       1, 2, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 2, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1], dtype=int32)
```

```python
df['cluster'] = y_predicted
df
```

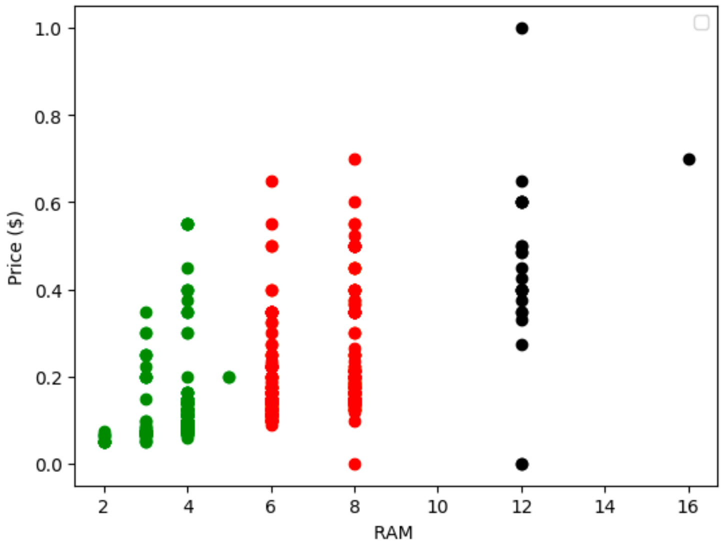
	Brand	Model	Storage	RAM	Battery Capacity (mAh)	Price ($)	n_cameras	res1	res2	res3	res4	screen	cluster
0	Apple	iPhone 13 Pro	128	6	3095	0.499499	3	12	12	12	0	6.10	1
1	Samsung	Galaxy S21 Ultra	256	12	5000	0.599600	4	108	10	10	12	6.80	2
2	OnePlus	9 Pro	128	8	4500	0.449449	4	48	50	8	2	6.70	1
3	Xiaomi	Redmi Note 10 Pro	128	6	5020	0.139139	4	64	8	5	2	6.67	1
4	Google	Pixel 6	128	8	4614	0.399399	2	50	12	0	0	6.40	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
402	Samsung	Galaxy Note20 5G	128	8	4300	0.524525	3	12	64	12	0	6.70	1
403	Xiaomi	Mi 10 Lite 5G	128	6	4160	0.174174	4	48	8	2	2	6.57	1
404	Apple	iPhone 12 Pro Max	128	6	3687	0.549550	3	12	12	12	0	6.70	1
405	Oppo	Reno3	128	8	4025	0.214214	4	48	13	8	2	6.40	1
406	Samsung	Galaxy S10 Lite	128	6	4500	0.324324	3	48	12	5	0	6.70	1

407 rows × 13 columns

```python
[51] df1 = df[df.cluster==0]
     df2 = df[df.cluster==1]
     df3 = df[df.cluster==2]
```

```python
plt.scatter(df1['RAM '],df1['Price ($)'],color='green')
plt.scatter(df2['RAM '],df2['Price ($)'],color='red')
plt.scatter(df3['RAM '],df3['Price ($)'],color='black')
plt.xlabel('RAM ')
plt.ylabel('Price ($)')
plt.legend()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with
<matplotlib.legend.Legend at 0x78f1f8608610>
```



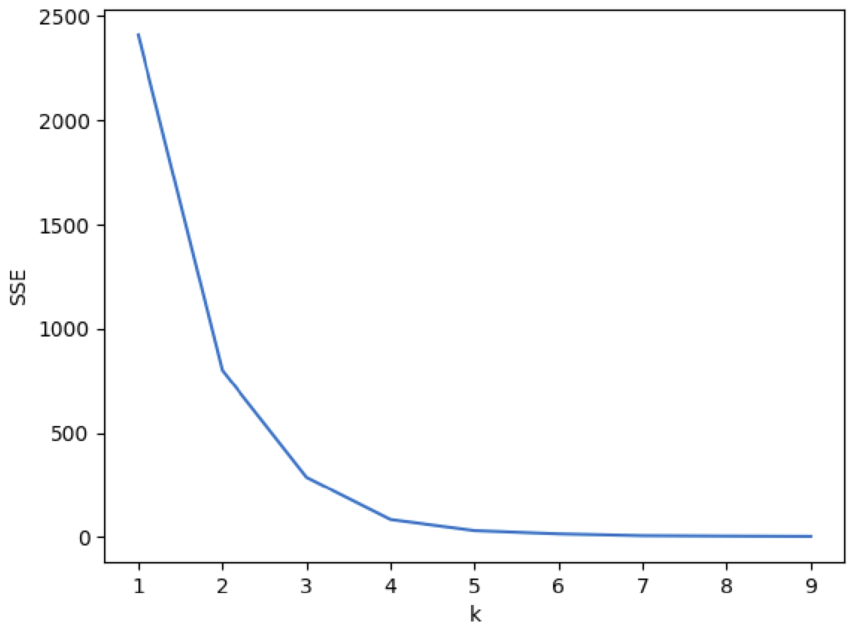
[56]

```python
k_rng= range(1,10)
sse = []
for k in k_rng:
  km=KMeans(n_clusters=k)
  km.fit(df[['RAM ','Price ($)']])
  sse.append(km.inertia_)

sse
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
[2410.0357028404947,
 801.2335263862351,
 287.89043153606735,
 84.73153036717075,
 31.573869212322716,
 16.127622428077935,
 7.557550950680905,
 5.594154106499305,
 4.117571726964012]
```

```python
plt.xlabel('k')
plt.ylabel('SSE')
plt.plot(k_rng,sse)
```

```
[<matplotlib.lines.Line2D at 0x78f1f84172b0>]
```



CONCLUSION:

Thus the given dataset is clustered using k-means clustering algorithm and 4 clusters has been grouped