In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [2]:

```
df = pd.read_csv("mobile.csv")
df
```

Out[2]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	Price in INR
0	POCO C50 (Royal Blue, 32 GB)	4.2	33,561	2 GB RAM	32 GB ROM	8MP Dual Camera	5MP Front Camera	5000 mAh	Mediatek Helio A22 Processor, Upto 2.0 GHz Pro...	₹5,649
1	POCO M4 5G (Cool Blue, 64 GB)	4.2	77,128	4 GB RAM	64 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatek Dimensity 700 Processor	₹11,999
2	POCO C51 (Royal Blue, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G36 Processor	₹6,999
3	POCO C55 (Cool Blue, 64 GB)	4.2	22,621	4 GB RAM	64 GB ROM	50MP Dual Rear Camera	5MP Front Camera	5000 mAh	Mediatek Helio G85 Processor	₹7,749
4	POCO C51 (Power Black, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G36 Processor	₹6,999
...	...	...	...	...	...	...	...	...	...	...
1831	Infinix Note 7 (Forest Green, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTek Helio G70 Processor	₹14,999
1832	Infinix Note 7 (Bolivia Blue, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTek Helio G70 Processor	₹14,999

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	Price in INR
1833	Infinix Note 7 (Aether Black, 64 GB)	4.3	25,582	4 GB RAM	64 GB ROM	48MP + 2MP + 2MP + AI Lens Camera	16MP Front Camera	5000 mAh	MediaTek Helio G70 Processor	₹14,999
1834	Infinix Zero 8i (Silver Diamond, 128 GB)	4.2	7,117	8 GB RAM	128 GB ROM	48MP + 8MP + 2MP + AI Lens Camera	16MP + 8MP Dual Front Camera	4500 mAh	MediaTek Helio G90T Processor	₹18,999
1835	Infinix S5 (Quetzal Cyan, 64 GB)	4.3	15,701	4 GB RAM	64 GB ROM	16MP + 5MP + 2MP + Low Light Sensor	32MP Front Camera	4000 mAh	Helio P22 (MTK6762) Processor	₹10,999

1836 rows × 11 columns

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

Out[3]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	Price in INR	Dat Scra
0	POCO C50 (Royal Blue, 32 GB)	4.2	33,561	2 GB RAM	32 GB ROM	8MP Dual Camera	5MP Front Camera	5000 mAh	Mediatek Helio A22 Processor, Upto 2.0 GHz Pro...	₹5,649	2023
1	POCO M4 5G (Cool Blue, 64 GB)	4.2	77,128	4 GB RAM	64 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatek Dimensity 700 Processor	₹11,999	2023
2	POCO C51 (Royal Blue, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G36 Processor	₹6,999	2023
3	POCO C55 (Cool Blue, 64 GB)	4.2	22,621	4 GB RAM	64 GB ROM	50MP Dual Rear Camera	5MP Front Camera	5000 mAh	Mediatek Helio G85 Processor	₹7,749	2023
4	POCO C51 (Power Black, 64 GB)	4.3	15,175	4 GB RAM	64 GB ROM	8MP Dual Rear Camera	5MP Front Camera	5000 mAh	Helio G36 Processor	₹6,999	2023

Data cleaning and pre processing

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1836 entries, 0 to 1835
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Phone Name         1836 non-null   object 
 1   Rating ?/5         1836 non-null   float64
 2   Number of Ratings  1836 non-null   object 
 3   RAM                1836 non-null   object 
 4   ROM/Storage        1662 non-null   object 
 5   Back/Rare Camera   1827 non-null   object 
 6   Front Camera       1435 non-null   object 
 7   Battery            1826 non-null   object 
 8   Processor          1781 non-null   object 
 9   Price in INR       1836 non-null   object 
 10  Date of Scraping   1836 non-null   object 
dtypes: float64(1), object(10)
memory usage: 157.9+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	Rating ?/5
count	1836.000000
mean	4.210512
std	0.543912
min	0.000000
25%	4.200000
50%	4.300000
75%	4.400000
max	4.800000

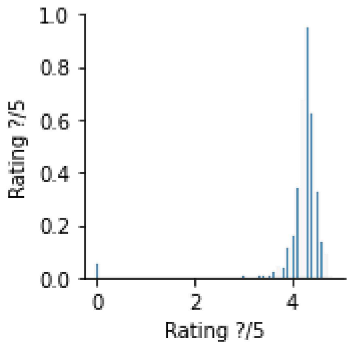
In [6]:
```
df.columns
```

Out[6]:
```
Index(['Phone Name', 'Rating ?/5', 'Number of Ratings', 'RAM', 'ROM/Storage',
       'Back/Rare Camera', 'Front Camera', 'Battery', 'Processor',
       'Price in INR', 'Date of Scraping'],
      dtype='object')
```

EDA and VISUALIZATION

In [7]:
```
sns.pairplot(df)
```

Out[7]: <seaborn.axisgrid.PairGrid at 0x1495a28ca00>

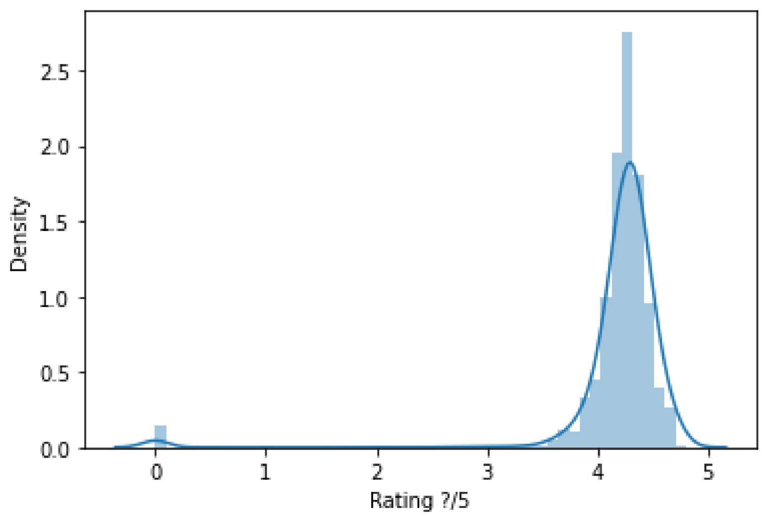


In [8]:

```
sns.distplot(df["Rating ?/5"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]: <AxesSubplot:xlabel='Rating ?/5', ylabel='Density'>



In [9]:

```
df1 = df[['Phone Name', 'Rating ?/5', 'Number of Ratings', 'RAM', 'ROM/Storage',
       'Back/Rare Camera', 'Front Camera', 'Battery', 'Processor',
       'Price in INR', 'Date of Scraping']]
```

In [10]:

```
sns.heatmap(df1.corr())
```

Out[10]: <AxesSubplot:>

In [11]:
```
x = df1[['Rating ?/5','Rating ?/5']]
y = df1['Rating ?/5']
```

split the data into training and test data

In [12]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
```

In [13]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)
```

Out[13]:
```
LinearRegression()
```

In [14]:
```
lr.intercept_
```

Out[14]:
```
-2.6645352591003757e-15
```

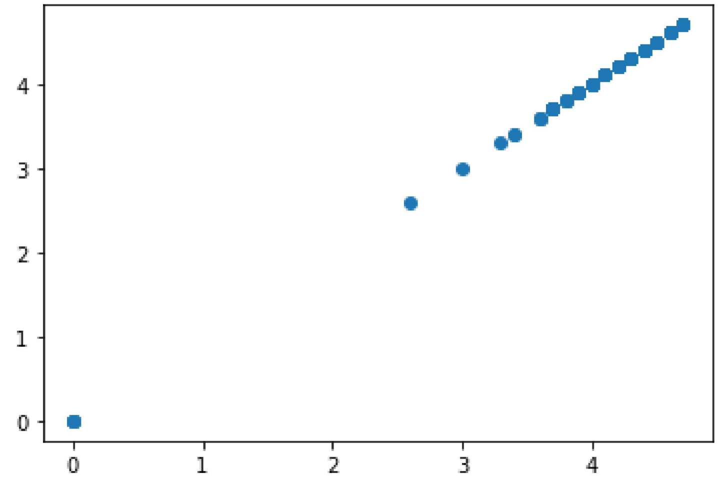
In [15]:
```
coeff = pd.DataFrame(lr.coef_, x.columns, columns =['Co-efficient'])
coeff
```

Out[15]:

	Co-efficient
Rating ?/5	0.5
Rating ?/5	0.5

In [16]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test, prediction)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x1495aed12e0>
```



In [17]:
```
lr.score(x_test,y_test)
```
Out[17]: 1.0

In [18]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [19]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_test,y_test)
rr.score(x_train,y_train)
```
Out[19]: 0.9998643503417138

In [20]:
```
rr.score(x_test,y_test)
```
Out[20]: 0.9998642822291992

In [21]:
```
la = Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[21]: Lasso(alpha=10)

In [22]:
```
la.score(x_test,y_test)
```
Out[22]: -0.0005021207975841602

In [23]:
```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```
Out[23]: ElasticNet()

In [24]:
```
print(en.coef_)
```

```
[0. 0.]
```

In [25]:

```
print(en.intercept_)
```

```
4.20739299610895
```

In [26]:

```
print(en.predict(x_test))
```

```
[4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
```

```
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393
 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393 4.207393]
```

In [27]:

```
print(en.score(x_test,y_test))
```

```
-0.0005021207975841602
```

Evaluation Metrics

In [28]:

```
from sklearn import metrics
```

In [29]:

```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 1.1767155106734871e-16
```

In [30]:

```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 1.3601050090017446e-31
```

In [32]:

```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 3.6879601529866676e-16
```