In [1]:

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

In [2]:

```
df=pd.read_csv(r'C:\Users\user\Downloads\12_mobile_prices_2023.csv')
df
```

Out[2]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor
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...
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
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
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
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
...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

1836 rows × 11 columns

In [3]:

```
df.head(10)
```

Out[3]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	Pri
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
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
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
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
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
5	POCO M4 5G (Power Black, 64 GB)	4.2	77,128	4 GB RAM	64 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatek Dimensity 700 Processor	₹11
6	POCO C55 (Power Black, 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
7	POCO C55 (Forest Green, 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
8	POCO C55 (Cool Blue, 128 GB)	4.1	13,647	6 GB RAM	128 GB ROM	50MP Dual Rear Camera	5MP Front Camera	5000 mAh	Mediatek Helio G85 Processor	₹9
9	POCO M4 5G (Yellow, 128 GB)	4.2	40,525	6 GB RAM	128 GB ROM	50MP + 2MP	8MP Front Camera	5000 mAh	Mediatek Dimensity 700 Processor	₹13

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]:

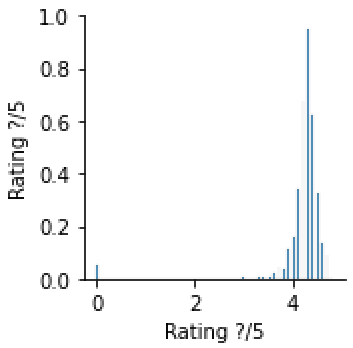
```
Index(['Phone Name', 'Rating ?/5', 'Number of Ratings', 'RAM', 'ROM/Storag
e',
       'Back/Rare Camera', 'Front Camera', 'Battery', 'Processor',
       'Price in INR', 'Date of Scraping'],
      dtype='object')
```

In [7]:

```
sns.pairplot(df)
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x1ea82e6b100>
```



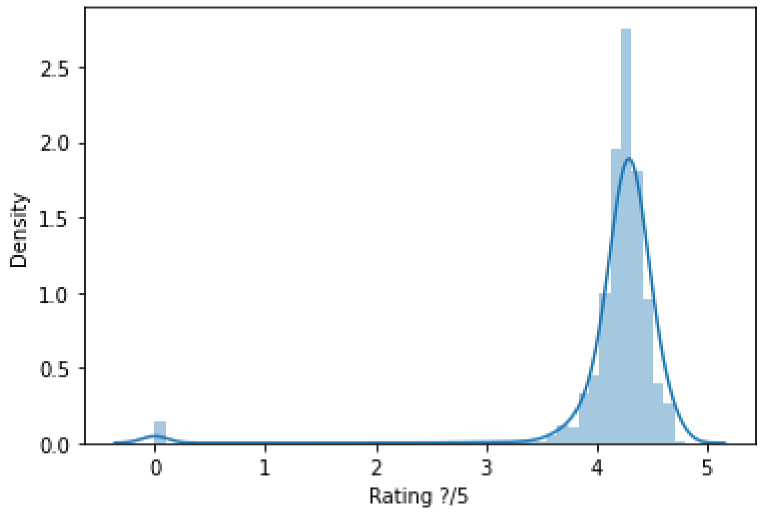
In [8]:

```
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 adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]:

```
<AxesSubplot:xlabel='Rating ?/5', ylabel='Density'>
```

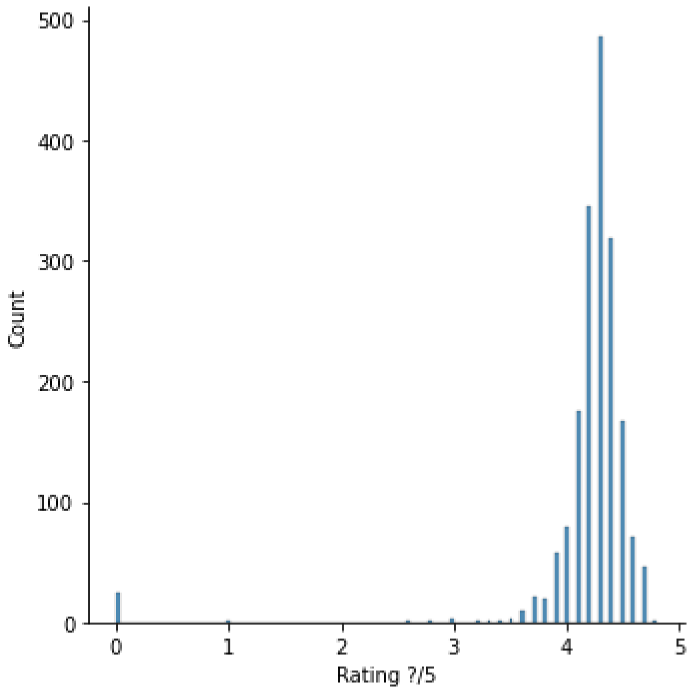


In [9]:

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

Out[9]:

```
<seaborn.axisgrid.FacetGrid at 0x1ea83948730>
```



In [10]:

```
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 [11]:

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

Out[11]:

```
<AxesSubplot:>
```

In [12]:

```
x=df1[['Rating ?/5']]
y=df1[['Rating ?/5']]
```

In [13]:

```
from sklearn.model_selection import train_test_split
```

In [14]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [15]:

```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)#ValueError: Input contains NaN, infinity or a value too large for
```

Out[15]:

```
LinearRegression()
```

In [16]:

```
print(lr.intercept_)
```

```
[2.66453526e-15]
```

In [17]:

```
coef= pd.DataFrame(lr.coef_)
coef
```

Out[17]:

	0
0	1.0

In [18]:

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

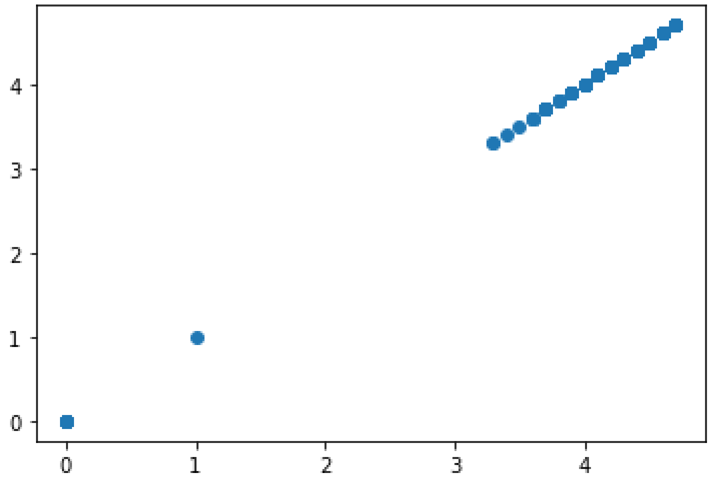
```
1.0
```

In [19]:

```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[19]:

```
<matplotlib.collections.PathCollection at 0x1ea8448f5e0>
```



In [20]:

```
lr.score(x_test,y_test)
```

Out[20]:

```
1.0
```

In [21]:

```
lr.score(x_train,y_train)
```

Out[21]:

```
1.0
```

In [22]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [23]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[23]:

```
Ridge(alpha=10)
```

In [24]:

```
rr.score(x_test,y_test)
```

Out[24]:

```
0.9992863129437463
```

In [25]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[25]:

```
Lasso(alpha=10)
```

In [26]:

```
la.score(x_test,y_test)
```

Out[26]:

```
-0.0014651937130194526
```

Elastic Net

In [27]:

```
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(x_train,y_train)
```

Out[27]:

```
ElasticNet()
```

In [28]:

```
print(en.coef_)
```

```
[0.]
```

In [29]:

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

```
[4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
```

```
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428
 4.2170428 4.2170428 4.2170428 4.2170428 4.2170428]
```

In [30]:

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

```
-0.0014651937130194526
```

Evaluation Metrics

In [31]:

```python
from sklearn import metrics
```

In [32]:

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

```
Mean Absolute Error: 0.2284012795977601
```

In [33]:

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

```
Mean Squared Error: 0.3236817866095619
```

In [34]:

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

```
Root Mean Squared Error: 0.5689303881931091
```