In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\12_mobile_prices_2023.csv")
df.fillna(0,inplace=True)
df
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

Out[2]:

	Phone Name	Rating ?/5	Number of Ratings	RAM	ROM/Storage	Back/Rare Camera	Front Camera	Battery	Processor	Pı
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	₹1
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	₹1
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	₹1
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	₹1
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	₹1

1836 rows × 11 columns

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

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
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

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        1836 non-null   object 
 5   Back/Rare Camera   1836 non-null   object 
 6   Front Camera       1836 non-null   object 
 7   Battery            1836 non-null   object 
 8   Processor          1836 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]:
```
import seaborn as sns
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

In [6]:
```
df.describe()
```

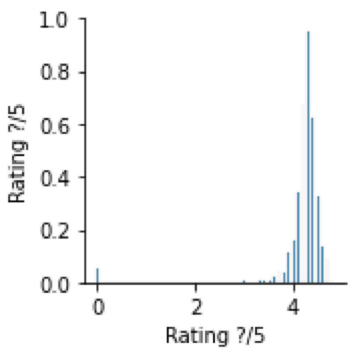
Out[6]:

	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 [ ]:

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

Out[7]: <seaborn.axisgrid.PairGrid at 0x2023ebc3a30>



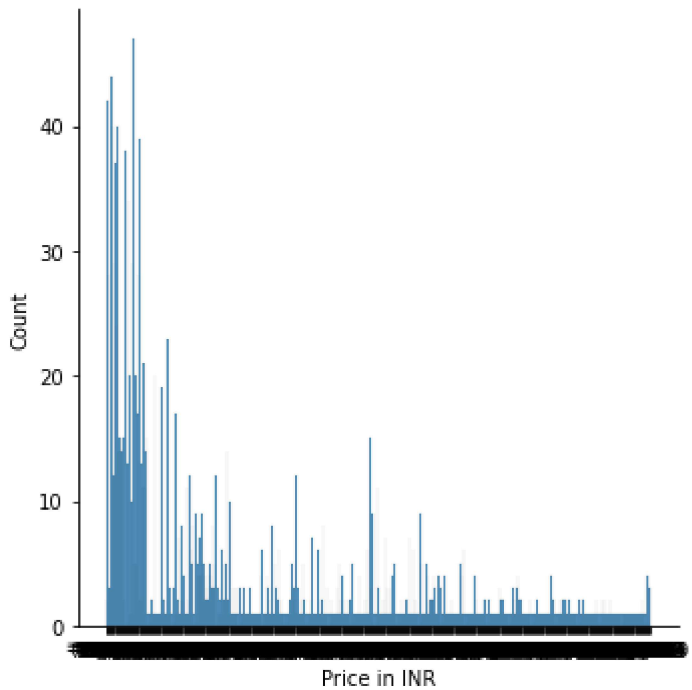
In [8]:
```
df1=df.drop(['Battery'],axis=1)
df1
df1=df1.drop(df1.index[1537:])
df1.isna().sum()
```

Out[8]:
```
Phone Name           0
Rating ?/5           0
Number of Ratings    0
RAM                  0
ROM/Storage          0
Back/Rare Camera     0
Front Camera         0
Processor            0
Price in INR         0
Date of Scraping     0
dtype: int64
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

In [9]:
```
sns.displot(df['Price in INR'])
```

Out[9]: <seaborn.axisgrid.FacetGrid at 0x2023bdd9f10>



In [10]:
```
sns.heatmap(df1.corr())
```

Out[10]: <AxesSubplot:>

In [11]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

In [12]:
```
df1.isna().sum()
```

Out[12]:
```
Phone Name           0
Rating ?/5           0
Number of Ratings    0
RAM                  0
ROM/Storage          0
Back/Rare Camera     0
Front Camera         0
Processor            0
Price in INR         0
Date of Scraping     0
dtype: int64
```

In [13]:
```
y=df1['Rating ?/5']
x=df1.drop(['Phone Name','ROM/Storage','RAM','Back/Rare Camera','Front Camera',
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
print(x_train)
```

```
      Rating ?/5
1417        3.8
1003        4.3
264         4.3
1053        4.3
490         4.6
...         ...
1051        4.4
529         4.2
1347        3.9
612         4.3
1227        4.3

[1075 rows x 1 columns]
```

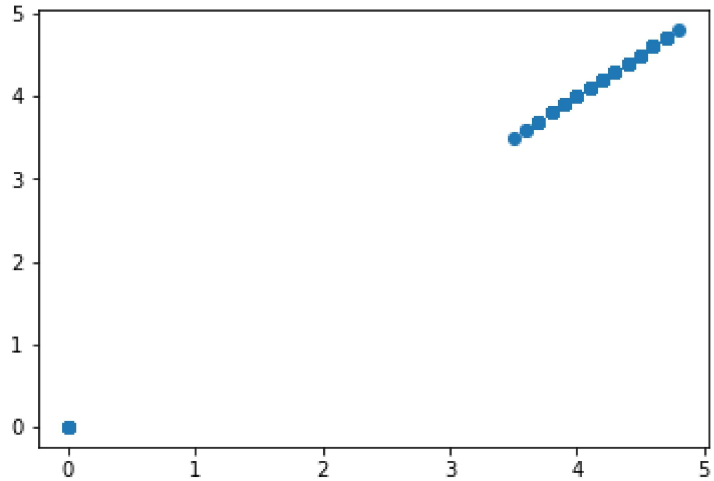
In [14]:
```
model=LinearRegression()
model.fit(x_train,y_train)
model.intercept_
```

Out[14]:
```
1.0658141036401503e-14
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

In [15]:
```
prediction=model.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[15]: <matplotlib.collections.PathCollection at 0x20242871c40>



In [16]:
```
model.score(x_test,y_test)
```

Out[16]: 1.0

In [17]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [18]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[18]: Ridge(alpha=10)

In [19]:
```
rr.score(x_test,y_test)
```

Out[19]: 0.9978250643987941

In [20]:
```
la =Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[20]: Lasso(alpha=10)

In [21]:
```
la.score(x_test,y_test)
```

Out[21]: -0.0076789222100206445

In [22]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[22]: ElasticNet()

In [23]:
```
print(en.coef_)
```

```
[0.]
```

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

```
4.254325581395348
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

In [25]:
```
print(en.predict(x_test))
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/STIX-Web/fontdata.js

```
[4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
```

```
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558
 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558 4.25432558]
```

In [26]:
```
print(en.score(x_test,y_test))
```

```
-0.007678922100206445
```

In [27]:
```
from sklearn import metrics
```

In [28]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 4.719649394726856e-16
```

In [29]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 2.063076166089367e-30
```

In [30]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prec
```

```
Root Mean Squared Error: 1.4363412429117834e-15
```