In [1]:

```
#Import All neccessary Libraries
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
#Load data & using pandas function Convert it into DataFrame.
df=pd.read_csv('used_device_data.csv')
df
```

Out[2]:

	device_brand	os	screen_size	4g	5g	rear_camera_mp	front_camera_mp	inter
0	Honor	Android	14.50	yes	no	13.0	5.0	
1	Honor	Android	17.30	yes	yes	13.0	16.0	
2	Honor	Android	16.69	yes	yes	13.0	8.0	
3	Honor	Android	25.50	yes	yes	13.0	8.0	
4	Honor	Android	15.32	yes	no	13.0	8.0	
...	...	...	...	...	...	...	...	
3449	Asus	Android	15.34	yes	no	NaN	8.0	
3450	Asus	Android	15.24	yes	no	13.0	8.0	
3451	Alcatel	Android	15.80	yes	no	13.0	5.0	
3452	Alcatel	Android	15.80	yes	no	13.0	5.0	
3453	Alcatel	Android	12.83	yes	no	13.0	5.0	

3454 rows × 15 columns

In [3]:

```
#Head return top rows of a dataframe(by default returns first five rows).
df.head()
```

Out[3]:

	device_brand	os	screen_size	4g	5g	rear_camera_mp	front_camera_mp	internal_
0	Honor	Android	14.50	yes	no	13.0	5.0	
1	Honor	Android	17.30	yes	yes	13.0	16.0	
2	Honor	Android	16.69	yes	yes	13.0	8.0	
3	Honor	Android	25.50	yes	yes	13.0	8.0	
4	Honor	Android	15.32	yes	no	13.0	8.0	

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   device_brand           3454 non-null   object 
 1   os                     3454 non-null   object 
 2   screen_size            3454 non-null   float64
 3   4g                     3454 non-null   object 
 4   5g                     3454 non-null   object 
 5   rear_camera_mp         3275 non-null   float64
 6   front_camera_mp        3452 non-null   float64
 7   internal_memory        3450 non-null   float64
 8   ram                    3450 non-null   float64
 9   battery                3448 non-null   float64
 10  weight                 3447 non-null   float64
 11  release_year           3454 non-null   int64  
 12  days_used              3454 non-null   int64  
 13  normalized_used_price  3454 non-null   float64
 14  normalized_new_price   3454 non-null   float64
dtypes: float64(9), int64(2), object(4)
memory usage: 404.9+ KB
```

#As we can see in the output, the summary includes list of all columns with their data types and the number of non-null values in each column. we also have the value of rangeindex provided for the index axis.

In [5]:

```
#returns the number of missing values in the dataset.
df.isnull().sum()
```

Out[5]:

```
device_brand              0
os                        0
screen_size               0
4g                        0
5g                        0
rear_camera_mp          179
front_camera_mp           2
internal_memory           4
ram                       4
battery                   6
weight                    7
release_year              0
days_used                 0
normalized_used_price     0
normalized_new_price      0
dtype: int64
```

In [6]:

```
df['rear_camera_mp'].value_counts()
```

Out[6]:

```
13.00    1035
8.00      755
5.00      546
12.00     189
2.00      171
16.00     154
3.15      122
0.30       69
10.50      30
1.30       26
23.00      19
21.00      19
20.70      16
20.00      13
14.50      13
12.20      12
4.00       10
12.30       9
19.00       8
8.10        7
13.10       6
10.00       6
24.00       5
3.00        5
12.50       4
48.00       4
6.50        4
6.70        4
21.20       2
21.50       1
1.20        1
16.30       1
22.60       1
18.00       1
12.60       1
20.10       1
41.00       1
20.20       1
1.00        1
0.08        1
22.50       1
Name: rear_camera_mp, dtype: int64
```

```
# rear_camera_mp column has 5% of null values so replace it with mean
```

In [7]:

```
df['rear_camera_mp']=df['rear_camera_mp'].fillna(df['rear_camera_mp'].mean())
```

In [8]:

```
df['front_camera_mp'].value_counts()
```

Out[8]:

```
5.00     791
8.00     549
2.00     538
0.30     492
16.00    298
1.30     148
32.00     94
13.00     90
20.00     67
1.20      43
0.00      39
2.10      37
1.00      34
1.60      31
24.00     30
25.00     28
12.00     21
7.00      21
4.00      20
1.90      12
2.20      11
10.00     10
5.10       9
3.00       7
14.50      5
1.10       5
9.00       4
14.00      3
3.70       3
0.65       2
18.00      2
1.25       2
10.50      2
16.30      1
0.90       1
3.50       1
1.80       1
Name: front_camera_mp, dtype: int64
```

In [9]:

```
df['internal_memory'].value_counts()
```

Out[9]:

```
16.00      1283
32.00      1083
64.00       509
128.00      372
256.00       86
512.00       44
0.06         18
0.10         17
8.00         12
4.00         10
1024.00       8
0.50          4
0.20          2
24.00         1
0.01          1
Name: internal_memory, dtype: int64
```

In [10]:

```
df['ram'].value_counts()
```

Out[10]:

```
4.00     2815
6.00      154
8.00      130
2.00       90
0.25       83
3.00       81
1.00       34
12.00      18
0.02       18
0.03       17
0.50        9
1.50        1
Name: ram, dtype: int64
```

In [11]:

```
df['battery'].value_counts()
```

Out[11]:

```
4000.0    341
3000.0    314
2000.0    244
2500.0    137
2100.0    121
         ...
6180.0      1
4180.0      1
2180.0      1
2880.0      1
3110.0      1
Name: battery, Length: 324, dtype: int64
```

In [12]:

```
df['weight'].value_counts()
```

Out[12]:

```
150.0    112
140.0     86
160.0     80
155.0     68
145.0     68
        ...
157.5      1
372.0      1
340.2      1
319.8      1
240.0      1
Name: weight, Length: 555, dtype: int64
```

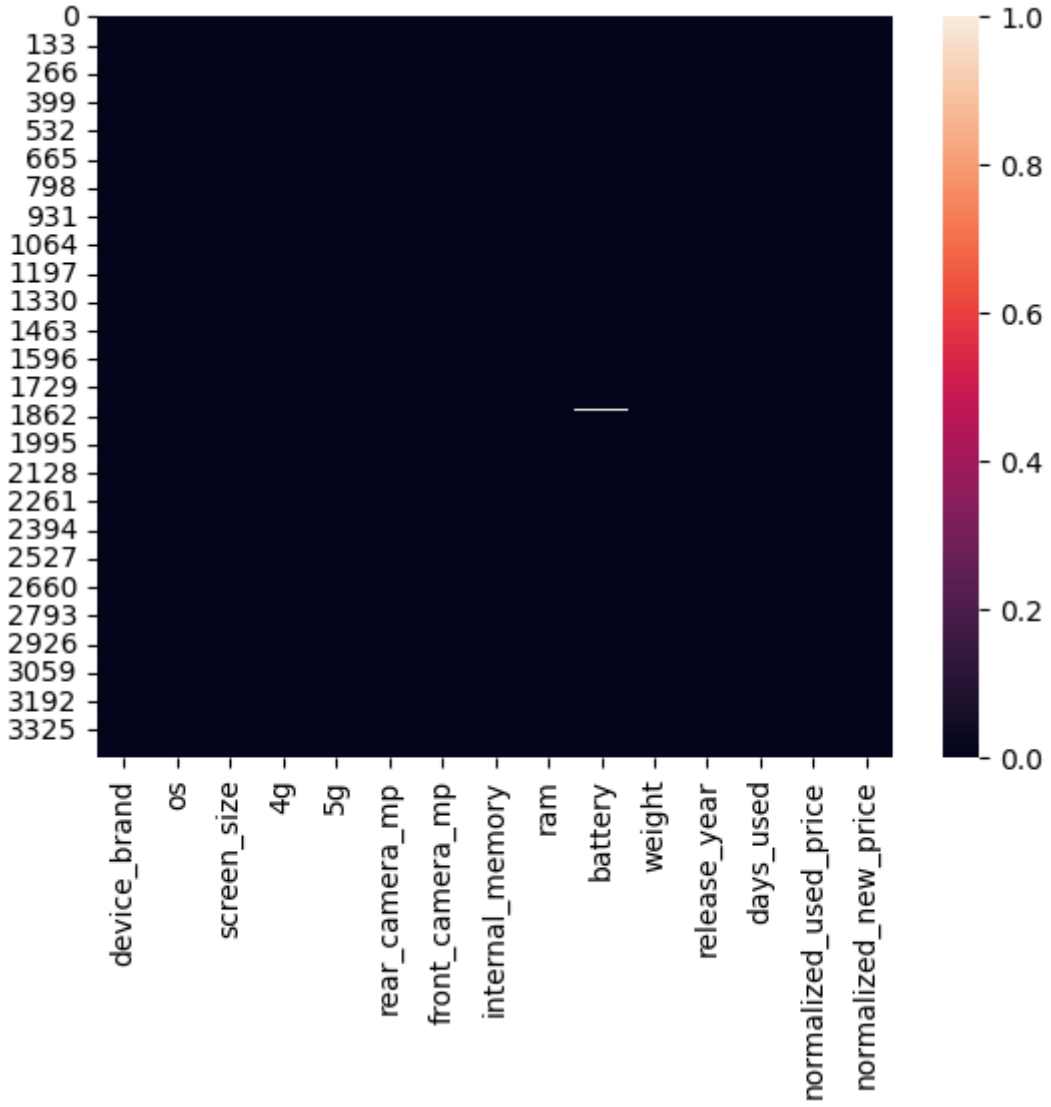
```
#It contains 2.25% null values so drop only null values at last after Cleanning the
data.
```

In [13]:

```
#Plot Heatmap to find out Null Values of DataFrame.
sns.heatmap(df.isnull())
```

Out[13]:

```
<AxesSubplot:>
```



In [14]:

```
#drop all remaining null values.
df.dropna(inplace=True)
df.shape
```

Out[14]:

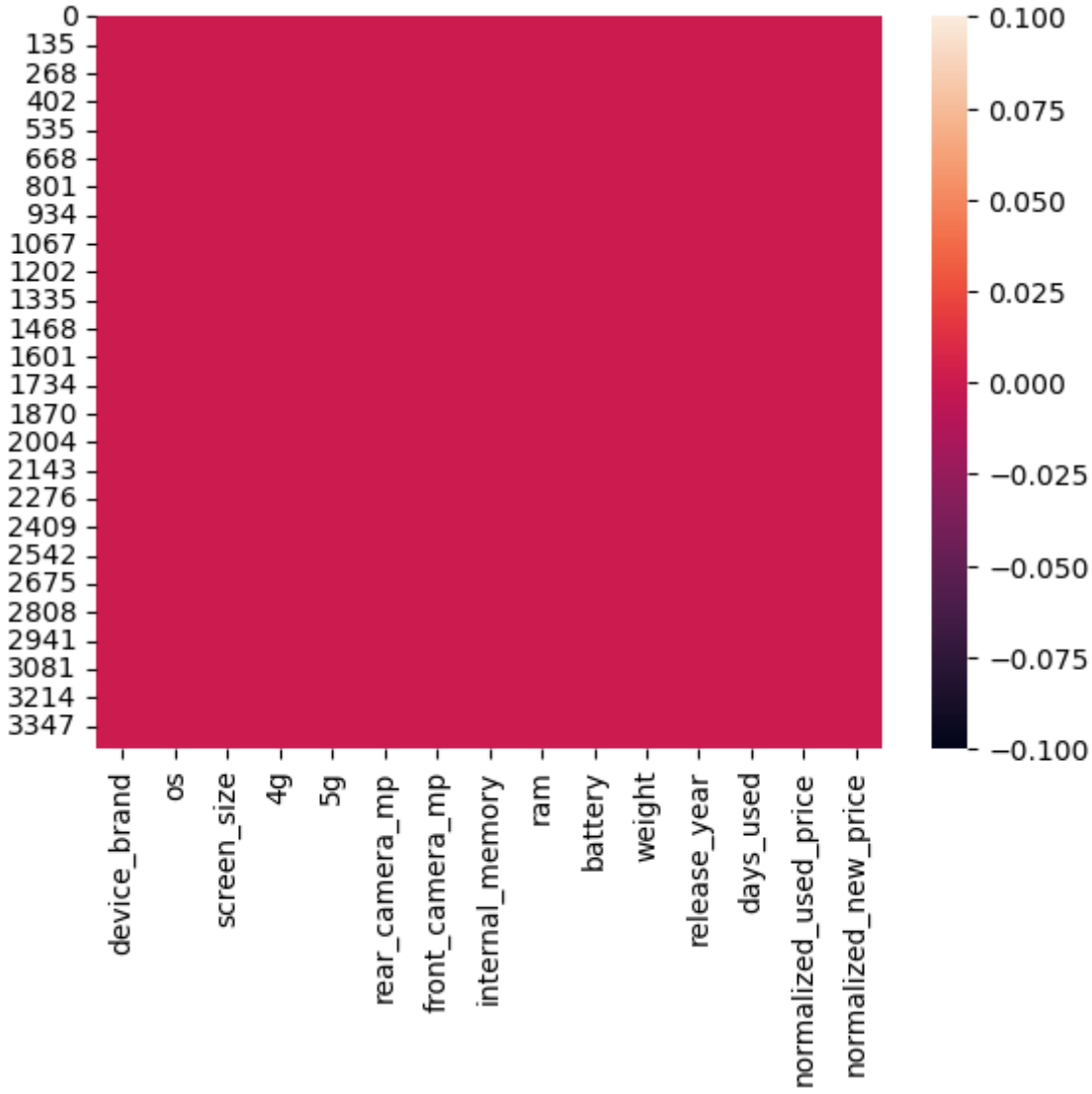
```
(3432, 15)
```

In [15]:

```
#Plot Heatmap to verfiy our data is clean or not.
sns.heatmap(df.isnull())
```

Out[15]:

<AxesSubplot:>



In [16]:

```
df.describe()
```

Out[16]:

	screen_size	rear_camera_mp	front_camera_mp	internal_memory	ram	I
count	3432.000000	3432.000000	3432.000000	3432.000000	3432.000000	3432.(
mean	13.733686	9.475512	6.582197	54.742672	4.042107	3139.(
std	3.788795	4.675254	6.979159	85.151126	1.360061	1298.8
min	5.080000	0.080000	0.000000	0.010000	0.020000	500.(
25%	12.700000	5.000000	2.000000	16.000000	4.000000	2100.(
50%	12.830000	9.460208	5.000000	32.000000	4.000000	3000.(
75%	15.370000	13.000000	8.000000	64.000000	4.000000	4000.(
max	30.710000	48.000000	32.000000	1024.000000	12.000000	9720.(

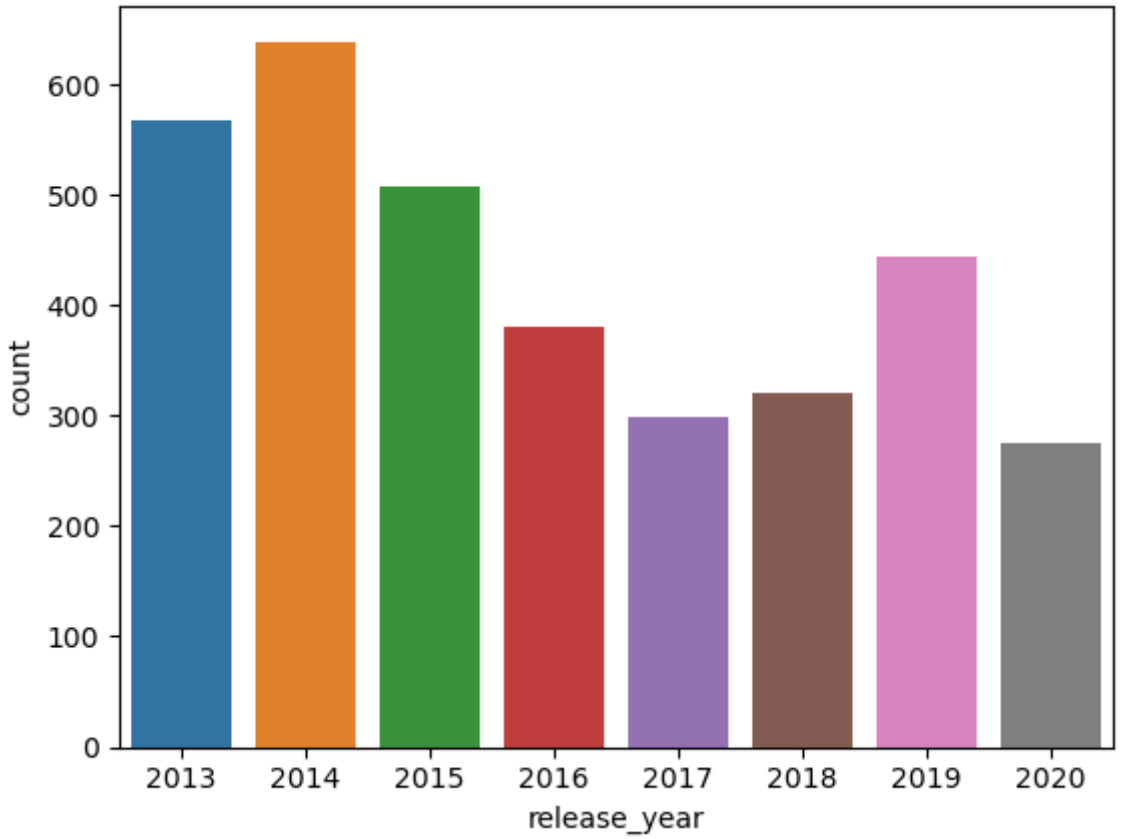
The describe() method returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains these information for each column: count - The number of not-empty values. mean - The average (mean) value. std - The standard deviation. min - the minimum value. 25% - The 25% percentile. *50% - The 50% percentile*. 75% - The 75% percentile*. max - the maximum value.

In [17]:

```
sns.countplot(data=df,x='release_year')
```

Out[17]:

```
<AxesSubplot:xlabel='release_year', ylabel='count'>
```

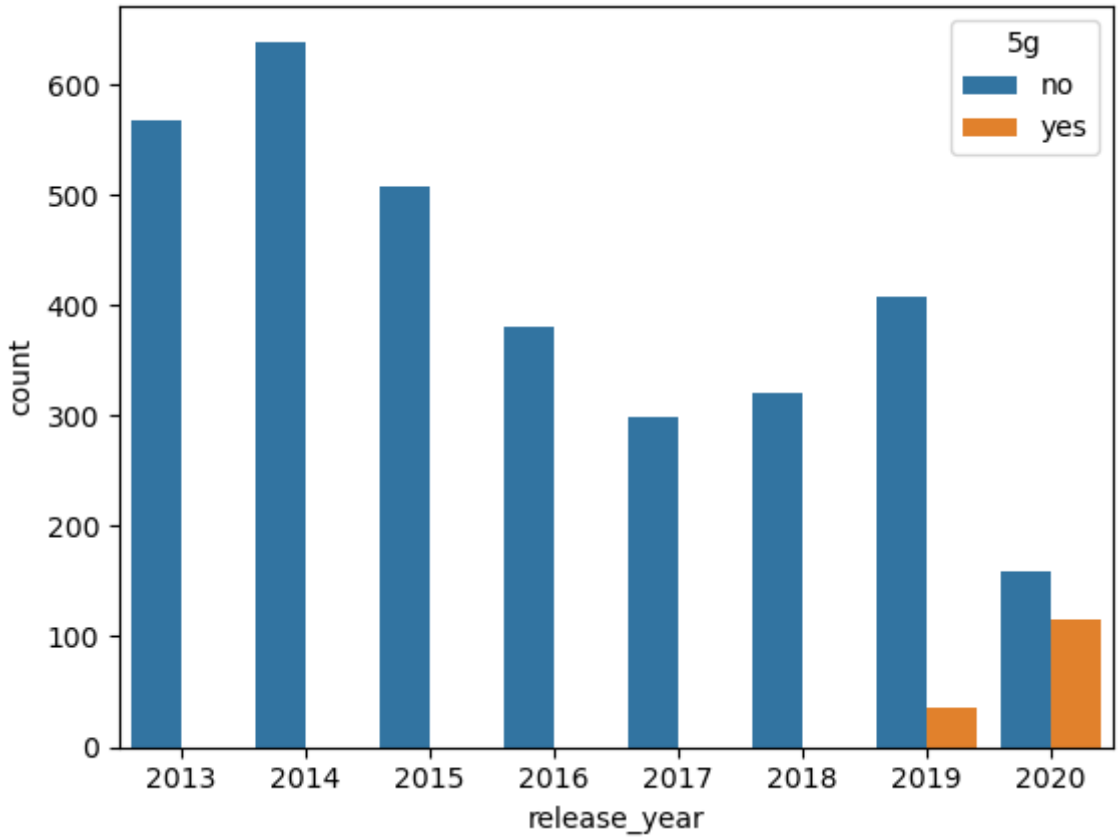


In [18]:

```
sns.countplot(data=df,x='release_year',hue='5g')
```

Out[18]:

```
<AxesSubplot:xlabel='release_year', ylabel='count'>
```

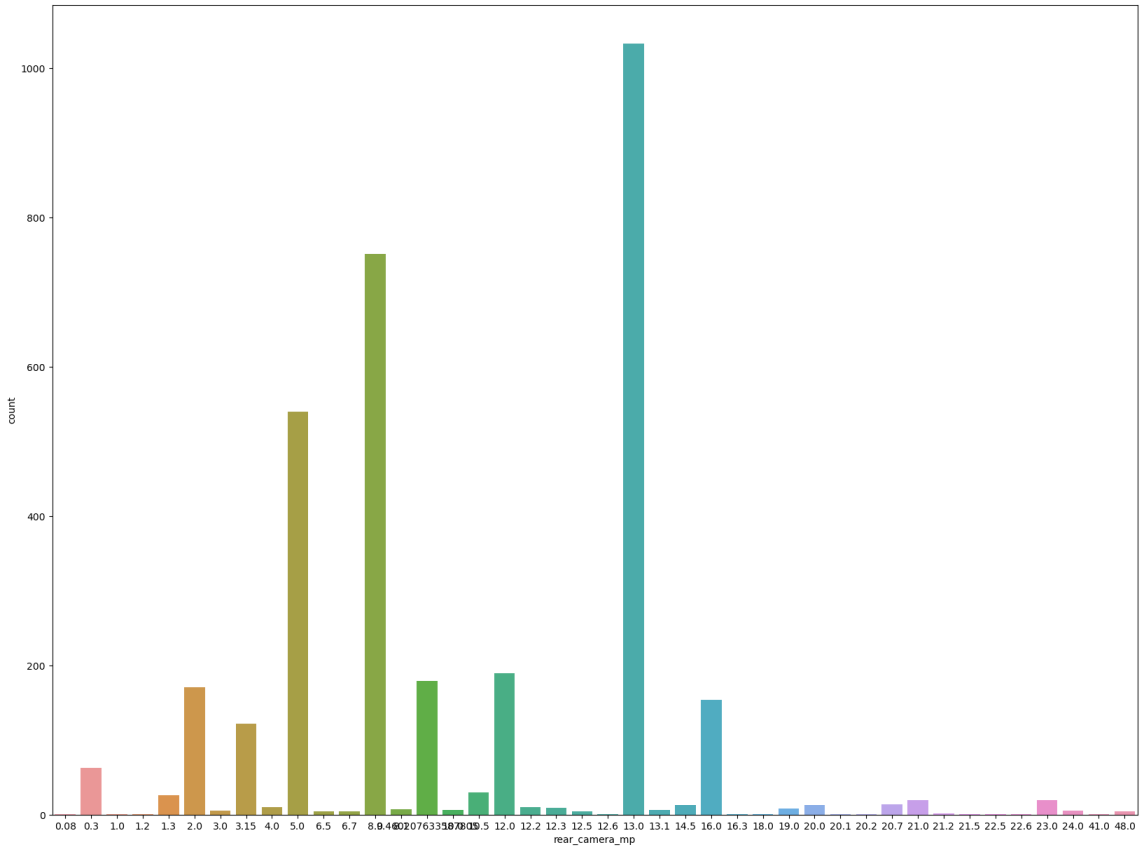


In [19]:

```
plt.figure(figsize=(20,15))
sns.countplot(data=df,x='rear_camera_mp')
```

Out[19]:

```
<AxesSubplot:xlabel='rear_camera_mp', ylabel='count'>
```

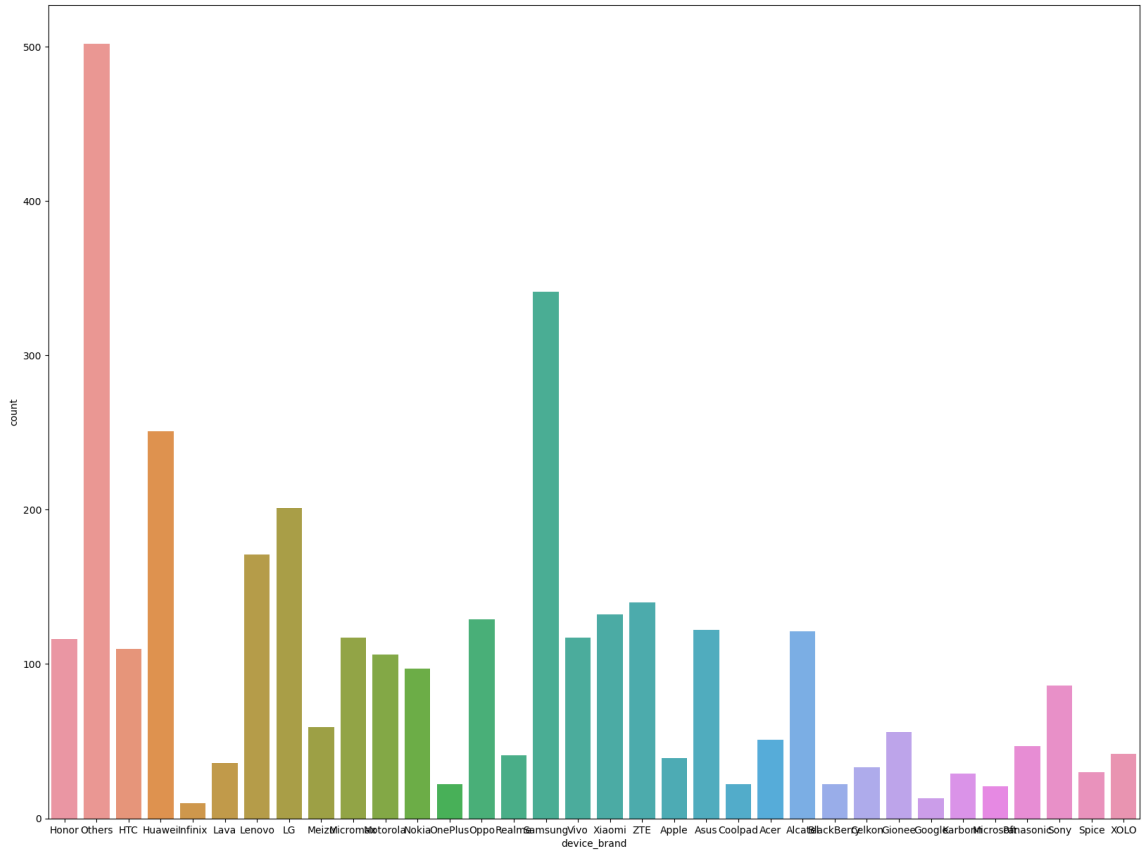


In [20]:

```
plt.figure(figsize=(20,15))
sns.countplot(data=df,x='device_brand')
```

Out[20]:

```
<AxesSubplot:xlabel='device_brand', ylabel='count'>
```

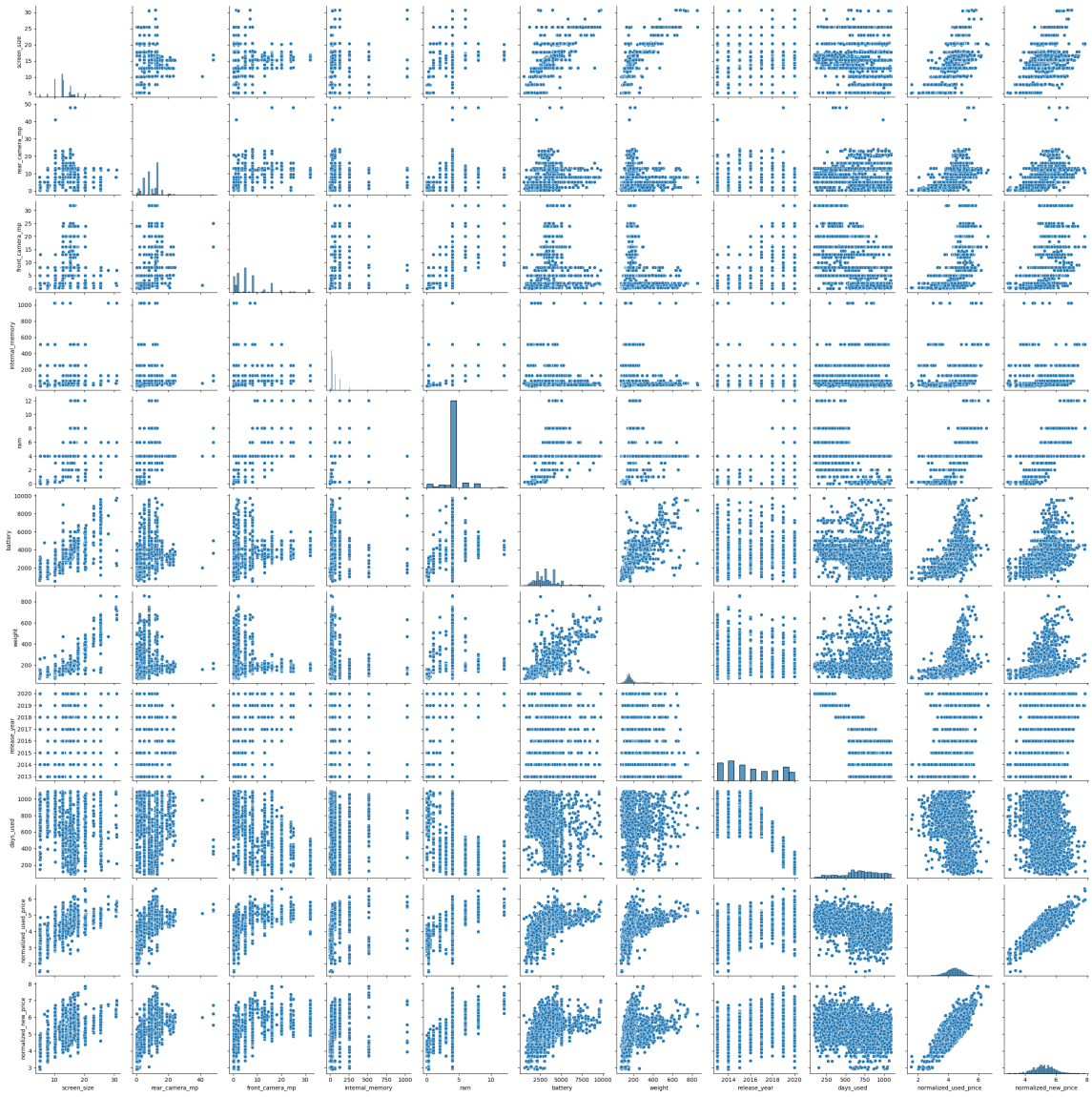


In [21]:

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

Out[21]:

```
<seaborn.axisgrid.PairGrid at 0x298bcbb5550>
```



In [22]:

```
#split the data into Numerical column.
numcol=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numcol.append(i)
numcol
```

Out[22]:

```
['screen_size',
 'rear_camera_mp',
 'front_camera_mp',
 'internal_memory',
 'ram',
 'battery',
 'weight',
 'release_year',
 'days_used',
 'normalized_used_price',
 'normalized_new_price']
```

In [23]:

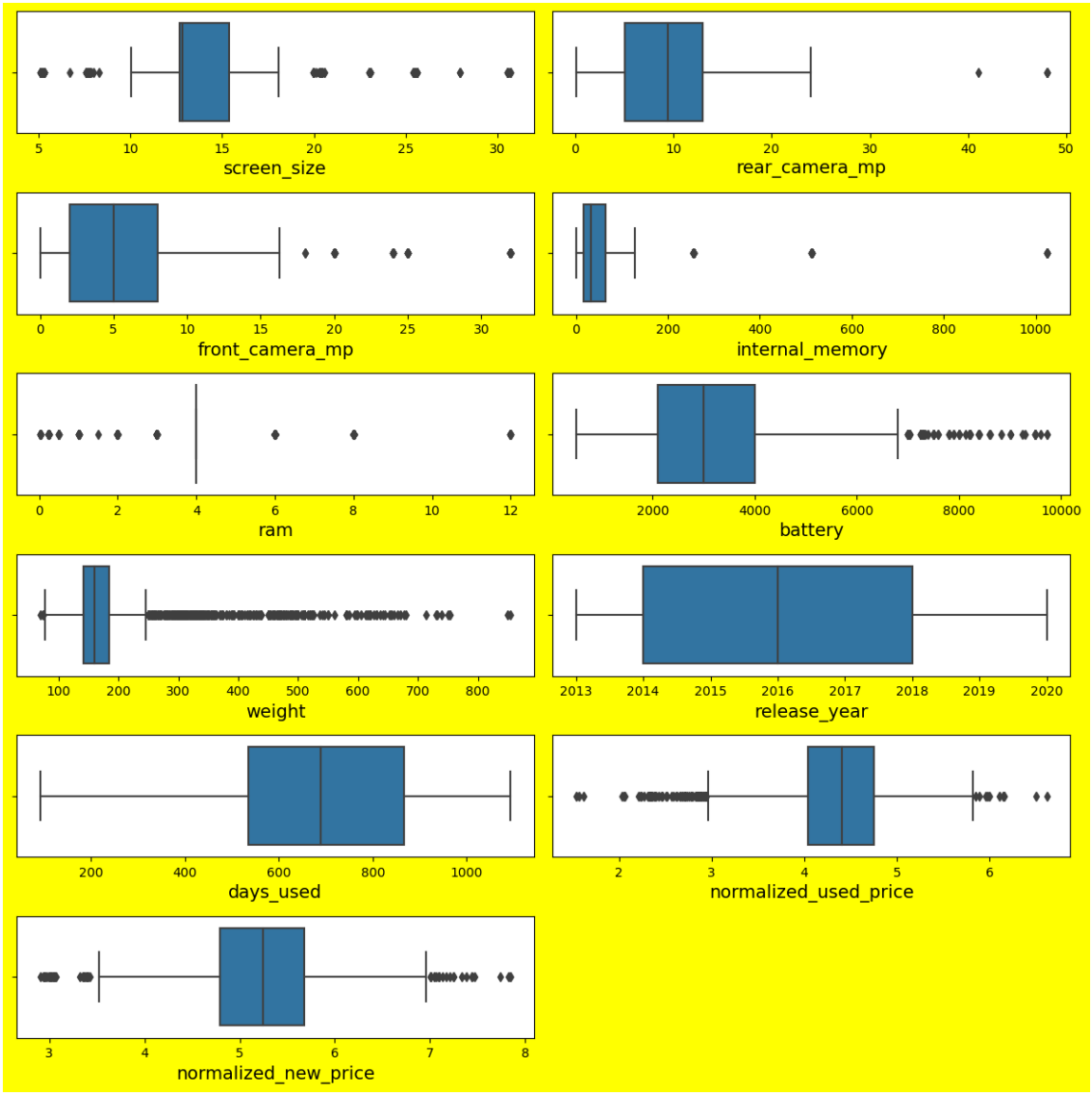
```
#split the data into Categorical column.
catcol=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        catcol.append(i)
catcol
```

Out[23]:

```
['device_brand', 'os', '4g', '5g']
```

In [24]:

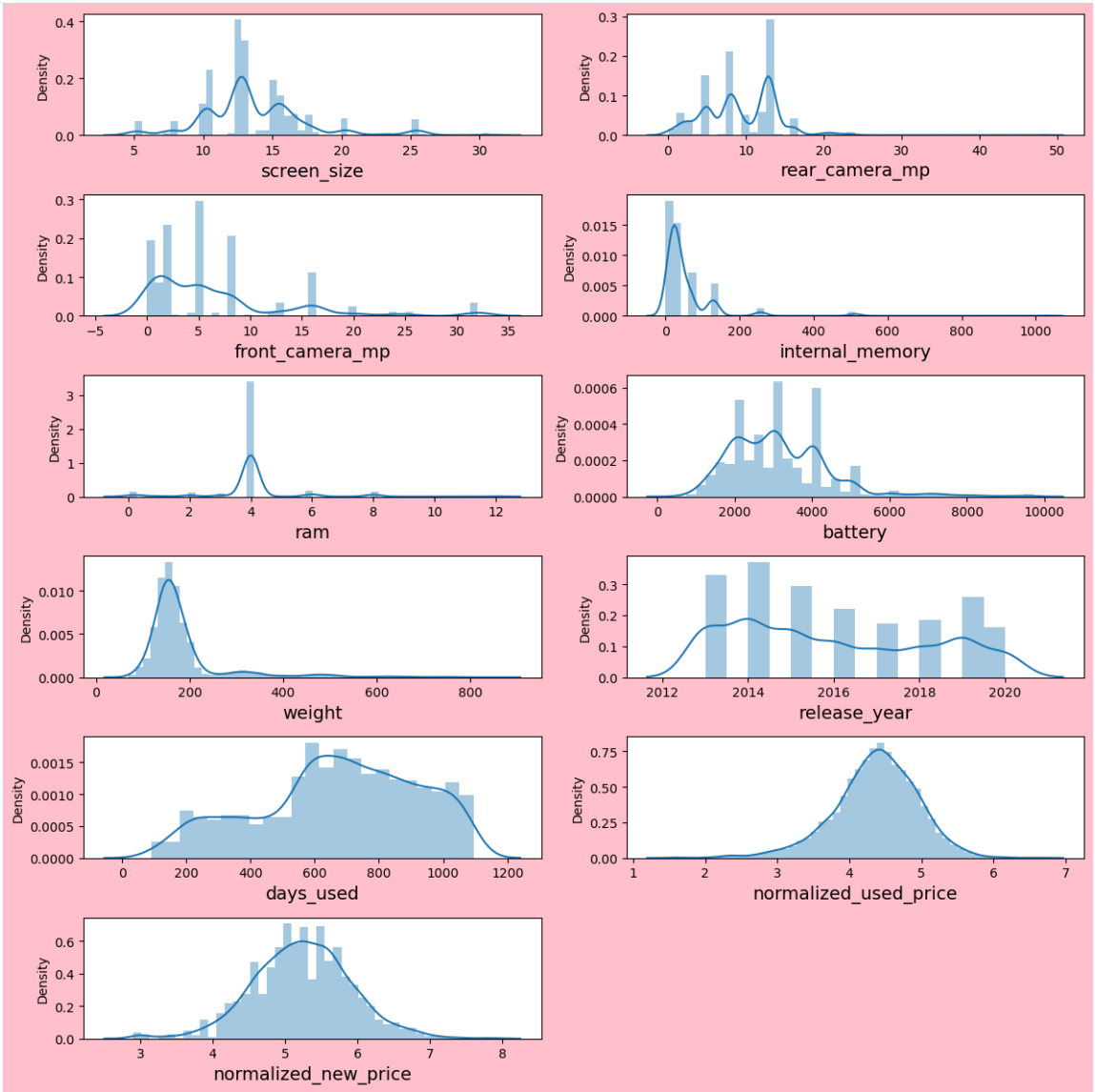
```
#Plot Boxplot to find out the outliers from numerical column.
plt.figure(figsize=(12,12),facecolor="yellow")
plotn=1
for column in numcol:
    if plotn<=12:
        ax=plt.subplot(6,2,plotn)
        sns.boxplot(df[column])
        plt.xlabel(column,fontsize=14)
        plotn+=1
plt.tight_layout()
```



#WE have outliers in every column except release_year and days_used.

In [25]:

```
#Plot Distplot to find out the skewness from numerical column.
plt.figure(figsize=(12,12),facecolor="pink")
plotn=1
for column in numcol:
    if plotn<=12:
        ax=plt.subplot(6,2,plotn)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize=14)
        plotn+=1
plt.tight_layout()
```



In [26]:

```
df.corr()
```

Out[26]:

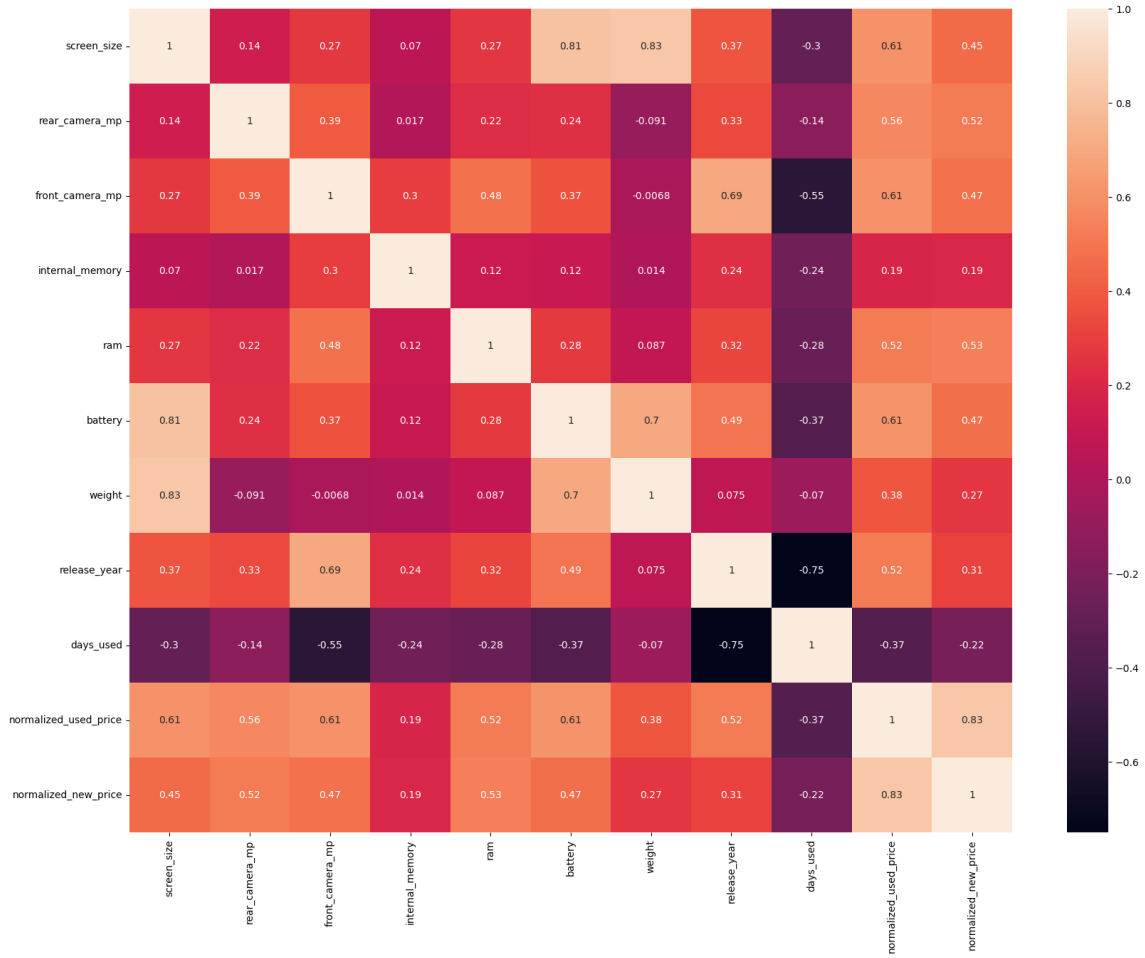
	screen_size	rear_camera_mp	front_camera_mp	internal_memory	
screen_size	1.000000	0.142265	0.268826	0.069643	0.2
rear_camera_mp	0.142265	1.000000	0.394166	0.017098	0.2
front_camera_mp	0.268826	0.394166	1.000000	0.295499	0.4
internal_memory	0.069643	0.017098	0.295499	1.000000	0.1
ram	0.268412	0.224868	0.476884	0.122101	1.0
battery	0.812583	0.237760	0.367514	0.116743	0.2
weight	0.829406	-0.090850	-0.006835	0.014358	0.0
release_year	0.371304	0.333410	0.693887	0.235697	0.3
days_used	-0.296507	-0.137093	-0.554687	-0.242514	-0.2
normalized_used_price	0.609638	0.563671	0.608389	0.189858	0.5
normalized_new_price	0.454568	0.518002	0.474022	0.194935	0.5

In [27]:

```
#Find out the correlation of Features and Target using Heatmap.
plt.figure(figsize=(20,15))
sns.heatmap(df.corr(),annot=True)
```

Out[27]:

<AxesSubplot:>



In [28]:

```
#Remove outliers using Z score method.
f=df[['screen_size','rear_camera_mp','front_camera_mp','internal_memory','ram','battery'
```

In [29]:

```
from scipy.stats import zscore
z=abs(zscore(f))
z
```

Out[29]:

	screen_size	rear_camera_mp	front_camera_mp	internal_memory	ram	battery
0	0.202287	0.753970	0.226736	0.108732	0.766332	0.091659
1	0.941416	0.753970	1.349615	0.860447	2.910510	0.893941
2	0.780392	0.753970	0.203178	0.860447	2.910510	0.816941
3	3.106008	0.753970	0.203178	0.108732	1.439773	3.165443
4	0.418747	0.753970	0.203178	0.108732	0.766332	1.432942
...	...	...	...	...	...	...
3449	0.424026	0.003274	0.203178	0.108732	1.439773	1.432942
3450	0.397629	0.753970	0.203178	0.860447	2.910510	0.662941
3451	0.545454	0.753970	0.226736	0.267125	0.766332	0.662941
3452	0.545454	0.753970	0.226736	0.267125	1.501701	0.662941
3453	0.238550	0.753970	0.226736	0.455053	1.501701	0.662941

3432 rows × 9 columns

In [30]:

```
newdf=df[(z<3).all(axis=1)]
newdf
```

Out[30]:

	device_brand	os	screen_size	4g	5g	rear_camera_mp	front_camera_mp	inter
0	Honor	Android	14.50	yes	no	13.000000	5.0	
1	Honor	Android	17.30	yes	yes	13.000000	16.0	
2	Honor	Android	16.69	yes	yes	13.000000	8.0	
4	Honor	Android	15.32	yes	no	13.000000	8.0	
5	Honor	Android	16.23	yes	no	13.000000	8.0	
...	...	...	...	...	...	...	...	
3449	Asus	Android	15.34	yes	no	9.460208	8.0	
3450	Asus	Android	15.24	yes	no	13.000000	8.0	
3451	Alcatel	Android	15.80	yes	no	13.000000	5.0	
3452	Alcatel	Android	15.80	yes	no	13.000000	5.0	
3453	Alcatel	Android	12.83	yes	no	13.000000	5.0	

3088 rows × 15 columns

In [31]:

```
print(df.shape)
print(newdf.shape)
```

```
(3432, 15)
(3088, 15)
```

In [32]:

```
data_loss=((3432-3088)/3432)*100
data_loss
```

Out[32]:

```
10.023310023310025
```

In [33]:

```
newdf.skew()
```

Out[33]:

```
screen_size              0.127369
rear_camera_mp           0.084966
front_camera_mp          1.285625
internal_memory          2.716251
ram                      0.373662
battery                  0.456749
weight                   2.181577
release_year             0.362529
days_used               -0.380988
normalized_used_price   -0.279443
normalized_new_price     0.116453
dtype: float64
```

In [34]:

```
s=['rear_camera_mp','front_camera_mp','internal_memory','ram','battery','weight','releas
from sklearn.preprocessing import PowerTransformer
scaler=PowerTransformer(method='yeo-johnson')
newdf[s]=scaler.fit_transform(newdf[s].values)
```

In [35]:

```
newdf.skew()
```

Out[35]:

```
screen_size             0.127369
rear_camera_mp         -0.087572
front_camera_mp        -0.019026
internal_memory         0.011406
ram                     0.591933
battery                -0.021101
weight                 -0.096302
release_year            0.000000
days_used              -0.179790
normalized_used_price  -0.279443
normalized_new_price    0.116453
dtype: float64
```

In [36]:

```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
newdf[catcol] = oe.fit_transform(newdf[catcol])
```

In [37]:

```
newdf.skew()
```

Out[37]:

```
device_brand           -0.374403
os                      4.965037
screen_size             0.127369
4g                     -0.798079
5g                      5.469142
rear_camera_mp         -0.087572
front_camera_mp        -0.019026
internal_memory         0.011406
ram                     0.591933
battery                -0.021101
weight                 -0.096302
release_year            0.000000
days_used              -0.179790
normalized_used_price  -0.279443
normalized_new_price    0.116453
dtype: float64
```

In [38]:

```
x = newdf.drop('normalized_used_price', axis = 1)# this is removing the normalized_used_
# features are always refered as x.
#2D dataframe
x
```

Out[38]:

	device_brand	os	screen_size	4g	5g	rear_camera_mp	front_camera_mp	internal_n
0	10.0	0.0	14.50	1.0	0.0	0.777923	0.170707	0.
1	10.0	0.0	17.30	1.0	1.0	0.777923	1.483681	1.
2	10.0	0.0	16.69	1.0	1.0	0.777923	0.669479	1.
4	10.0	0.0	15.32	1.0	0.0	0.777923	0.669479	0.
5	10.0	0.0	16.23	1.0	0.0	0.777923	0.669479	0.
...	...	...	...	...	...	...	...	
3449	3.0	0.0	15.34	1.0	0.0	0.000054	0.669479	0.
3450	3.0	0.0	15.24	1.0	0.0	0.777923	0.669479	1.
3451	1.0	0.0	15.80	1.0	0.0	0.777923	0.170707	0.
3452	1.0	0.0	15.80	1.0	0.0	0.777923	0.170707	0.
3453	1.0	0.0	12.83	1.0	0.0	0.777923	0.170707	-0.

3088 rows × 14 columns

In [39]:

```
# y contains only the target.
#1D series
y = newdf['normalized_used_price']
y
```

Out[39]:

```
0       4.307572
1       5.162097
2       5.111084
4       4.389995
5       4.413889
          ...
3449    4.492337
3450    5.037732
3451    4.357350
3452    4.349762
3453    4.132122
Name: normalized_used_price, Length: 3088, dtype: float64
```

In [40]:

```
#Split the Data For Training And Testing
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

In [41]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression, Lasso,Ridge
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
```

Linear Regression

In [42]:

```
#Apply Linear Regression() and find out accuracy
lr=LinearRegression()
lr.fit(xtrain,ytrain)

#Prediction
ypred=lr.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=lr.score(xtrain,ytrain)
test=lr.score(xtest,ytest)
print(f"Training score :{train}\nTesting score: {test}")
```

```
MAE:0.17606835727761533
MSE:0.04893606331692776
RMSE:0.22121497082459804
r2_score:0.7991405789981689
Training score :0.8134870410723317
Testing score: 0.7991405789981689
```

In [43]:

```
# Checking Cross Validation Score for Linear Regression
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(lr,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
79.9140578998169

 Cross validation Score: 79.36788713704901

 R2 Score - Cross Validation score is 0.5461707627678862
```

Regularization

In [44]:

```
#Lasso Regressor
l1=Lasso()
l1.fit(xtrain,ytrain)

#Prediction
ypred=l1.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=l1.score(xtrain,ytrain)
test=l1.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.3870015381330674
MSE:0.2438550440278183
RMSE:0.4938168122166542
r2_score:-0.0009097510477604764
Training score:0.0
Testing score:-0.0009097510477604764
```

In [45]:

```
# Checking Cross Validation Score for Lasso Regressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(l1,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
-0.09097510477604764

 Cross validation Score: -5.451601247326372

 R2 Score - Cross Validation score is 5.360626142550324
```

In [46]:

```
#Ridge Regressor
l2=Ridge()
l2.fit(xtrain,ytrain)

#Prediction
ypred=l2.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=l2.score(xtrain,ytrain)
test=l2.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.17600840939396137
MSE:0.04891164844201754
RMSE:0.2211597803444775
r2_score:0.7992407905253311
Training score:0.8134860327485417
Testing score:0.7992407905253311
```

In [47]:

```
# Checking Cross Validation Score for Ridge Regressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(l2,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
79.92407905253312

 Cross validation Score: 79.37301511837045

 R2 Score - Cross Validation score is 0.55106393416267
```

Knn Regressor

In [48]:

```
#knn
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor()
knn.fit(xtrain,ytrain)

#Prediction
ypred=knn.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=knn.score(xtrain,ytrain)
test=knn.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.20665601473311757
MSE:0.07266591467219231
RMSE:0.26956616010210244
r2_score:0.7017407499026154
Training score:0.8250202327277942
Testing score:0.7017407499026154
```

In [49]:

```
# Checking Cross Validation Score for knn Regressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(knn,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
70.17407499026153

 Cross validation Score: 59.13962116759439

 R2 Score - Cross Validation score is 11.034453822667139
```

Decision Tree Regressor

In [50]:

```
#Decision Tree
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(xtrain,ytrain)

#Prediction
ypred=dt.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=dt.score(xtrain,ytrain)
test=dt.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.23570132316073358
MSE:0.09494615865465011
RMSE:0.308133345574039
r2_score:0.610290874232948
Training score:1.0
Testing score:0.610290874232948
```

In [51]:

```
# Checking Cross Validation Score for decision tree Regressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(dt,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
61.029087423294804

 Cross validation Score: 60.116867087257866

 R2 Score - Cross Validation score is 0.9122203360369383
```

Random Forest Regressor

In [52]:

```
#Random Forest
rfr=RandomForestRegressor()
rfr.fit(xtrain,ytrain)

#Prediction
ypred=rfr.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=rfr.score(xtrain,ytrain)
test=rfr.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.17846702697456301
MSE:0.049379224403056185
RMSE:0.22221436587911275
r2_score:0.7973216121026543
Training score:0.9735690262710986
Testing score:0.7973216121026543
```

In [53]:

```
# Checking Cross Validation Score for random forest Regressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(rfr,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
79.73216121026543

 Cross validation Score: 78.7405962793212

 R2 Score - Cross Validation score is 0.9915649309442358
```

AdaBoosting Regressor

In [54]:

```
#AdaBoosting
from sklearn.ensemble import AdaBoostRegressor
abr=AdaBoostRegressor()
abr.fit(xtrain,ytrain)

#Prediction
ypred=abr.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=abr.score(xtrain,ytrain)
test=abr.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.2045077348119704
MSE:0.06215120493125723
RMSE:0.24930143387324757
r2_score:0.7448986659141381
Training score:0.7827377822956223
Testing score:0.7448986659141381
```

In [55]:

```
# Checking Cross Validation Score for AdaBoosting Regressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(abr,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
74.48986659141381

 Cross validation Score: 75.67422931929163

 R2 Score - Cross Validation score is -1.1843627278778115
```

Gadient Boosting Regressor

In [56]:

```
#Gradient Boosting
gbr=GradientBoostingRegressor()
gbr.fit(xtrain,ytrain)

#Prediction
ypred=gbr.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=gbr.score(xtrain,ytrain)
test=gbr.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.17441616406788849
MSE:0.04762685727835556
RMSE:0.21823578367984375
r2_score:0.8045142512769676
Training score:0.8699746330057013
Testing score:0.8045142512769676
```

In [57]:

```
# Checking Cross Validation Score for Gradient Boosting Regressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(gbr,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
80.45142512769677

 Cross validation Score: 79.01721747040335

 R2 Score - Cross Validation score is 1.4342076572934133
```

Extreme Gradient Boosting Regressor (XGB)

In [58]:

```
#XGB Boosting
from xgboost import XGBRegressor
xgb=XGBRegressor(verbosity=0)
xgb.fit(xtrain,ytrain)

#Prediction
ypred=xgb.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=xgb.score(xtrain,ytrain)
test=xgb.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.18420100663852668
MSE:0.05535864656506207
RMSE:0.23528418256453634
r2_score:0.7727789089921129
Training score:0.9842098878348459
Testing score:0.7727789089921129
```

In [59]:

```
# Checking Cross Validation Score for XGBRegressor
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(xgb,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
77.27789089921129

 Cross validation Score: 76.35211720309663

 R2 Score - Cross Validation score is 0.9257736961146605
```

Support Vector Machine Regressor

In [60]:

```
#Hard Margin
from sklearn.svm import SVR
svr1 = SVR(kernel = 'rbf')
svr1.fit(xtrain,ytrain)

#Prediction
ypred=svr1.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=svr1.score(xtrain,ytrain)
test=svr1.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.17395877462744258
MSE:0.04877383477267223
RMSE:0.2208479901938712
r2_score:0.7998064505305418
Training score:0.8176508574581984
Testing score:0.7998064505305418
```

In [61]:

```
# Checking Cross Validation Score for SVM Regressor (hard margin)
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(svr1,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
79.98064505305418

 Cross validation Score: 79.28024384972754

 R2 Score - Cross Validation score is 0.7004012033266349
```

In [62]:

```
# soft margin
svr2=SVR(kernel="rbf",C=0.5)
svr2.fit(xtrain,ytrain)

#Prediction
ypred=svr2.predict(xtest)

#Metric Evaluation
mae=mean_absolute_error(ytest,ypred)
mse=mean_squared_error(ytest,ypred)
rmse=np.sqrt(mse)
r2=r2_score(ytest,ypred)
print(f"MAE:{mae}\nMSE:{mse}\nRMSE:{rmse}\nr2_score:{r2}")

train=svr2.score(xtrain,ytrain)
test=svr2.score(xtest,ytest)
print(f"Training score:{train}\nTesting score:{test}")
```

```
MAE:0.17521169303380527
MSE:0.050092306552559744
RMSE:0.22381310630202098
r2_score:0.7943947467610672
Training score:0.8075161158162047
Testing score:0.7943947467610672
```

In [63]:

```
# Checking Cross Validation Score for SVM Regressor(soft margin)
ac=r2_score(ytest,ypred)*100
print(ac)
cv_score=(cross_val_score(svr2,x,y,cv=5).mean())*100
print("\n Cross validation Score:",cv_score)

#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
79.43947467610673

 Cross validation Score: 78.20564151153626

 R2 Score - Cross Validation score is 1.2338331645704699
```

In [64]:

```
model_list=['Linear Regressor','Lasso Regressor','Ridge Regressor','Knn Regressor','Deci
```

In [65]:

```
accuracyscore=[79.914,-0.09,79.92,70.17,59.03,79.66,75.503,80.44,77.27,79.098,79.43]
```

In [66]:

```
crossvalidationscore=[79.367,-5.45,79.37,59.13,60.41,78.93,76.004,79.02,76.35,79.28,78.2
```

In [67]:

```
a=pd.DataFrame({})
a["Regressor Type"]=model_list
a["Accuracy score"]=accuracyscore
a["Cross validation Score"]=crossvalidationscore
a
```

Out[67]:

	Regressor Type	Accuracy score	Cross validation Score
0	Linear Regressor	79.914	79.367
1	Lasso Regressor	-0.090	-5.450
2	Ridge Regressor	79.920	79.370
3	Knn Regressor	70.170	59.130
4	Decision Tree Regressor	59.030	60.410
5	Random Forest Regressor	79.660	78.930
6	AdaBoosting Regressor	75.503	76.004
7	Gradient Boosting Regressor	80.440	79.020
8	Extreme Gradient Boosting Regressor	77.270	76.350
9	SVM Regressor(hard margin)	79.098	79.280
10	SVM Regressor(soft margin)	79.430	78.200

In [68]:

```
a['Difference']=(a['Accuracy score']-a['Cross validation Score'])
a
```

Out[68]:

	Regressor Type	Accuracy score	Cross validation Score	Difference
0	Linear Regressor	79.914	79.367	0.547
1	Lasso Regressor	-0.090	-5.450	5.360
2	Ridge Regressor	79.920	79.370	0.550
3	Knn Regressor	70.170	59.130	11.040
4	Decision Tree Regressor	59.030	60.410	-1.380
5	Random Forest Regressor	79.660	78.930	0.730
6	AdaBoosting Regressor	75.503	76.004	-0.501
7	Gradient Boosting Regressor	80.440	79.020	1.420
8	Extreme Gradient Boosting Regressor	77.270	76.350	0.920
9	SVM Regressor(hard margin)	79.098	79.280	-0.182
10	SVM Regressor(soft margin)	79.430	78.200	1.230

In [69]:

```
# hyper-parameter tunning
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
```

In [70]:

```
# specify range of hyperparameters
hyper_params = [{'n_features_to_select': list(range(2,20))}]

# specify model
lm = LinearRegression()
lm.fit(xtrain, ytrain)
rfe = RFE(lm)

# set up GridSearchCV()
model_cv = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = 5,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(xtrain, ytrain)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

Out[70]:

```
GridSearchCV(cv=5, estimator=RFE(estimator=LinearRegression()),
             param_grid=[{'n_features_to_select': [2, 3, 4, 5, 6, 7, 8, 9,
10,
                                                   11, 12, 13, 14, 15, 16,
17,
                                                   18, 19]}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [71]:

```
model_cv.best_score_
```

Out[71]:

```
0.8107448577334846
```

In [72]:

```
ypred = model_cv.predict(xtest)
print(ypred)
```

```
 4.33646934 4.43080067 4.22397626 4.8299597  4.87837148 4.61089386
 4.15739697 4.16453975 4.26380169 4.32083145 4.72023989 3.78194115
 4.33186637 4.8813912  4.57595505 4.80814941 4.54183083 4.61047407
 4.49040426 4.56500001 4.62977043 4.30415201 5.13361269 4.45807888
 4.91801583 4.58661632 4.70711661 3.27582974 4.35067335 4.4717652
 4.11079461 4.68828108 4.65896452 4.49934051 4.2169892  4.53063014
 4.44441093 4.44998675 4.28310815 4.00845636 4.53205432 4.25470081
 3.93144754 4.84472166 4.61688953 4.69566183 4.94624343 4.68785499
 4.5489413  4.5694514  3.97030931 4.45526137 3.99823985 4.20440963
 4.43922047 4.32061578 3.69064252 4.54561676 3.7075824  4.17434996
 4.59362501 4.31709768 3.62408825 5.07847937 4.37666894 3.48786428
 4.53252018 3.777184   3.76305612 3.75962439 4.61819102 4.85086298
 4.91850041 4.75641435 3.90248587 4.06520091 3.94795948 4.39770707
 3.79279637 4.81381602 2.86622832 3.90652346 5.13299565 4.71725803
 3.46687453 4.18707109 4.44028955 3.77348902 3.43059268 3.72200367
 4.76386133 3.68772765 4.10877325 4.8921652  4.8838708  4.36136382
 3.95877159 4.52716988 4.95329695 4.65703325 3.99547217 4.115383
 3.86822313 4.47106612 4.00479218 4.55292234 4.71190215 4.6053454
 4.07302105 5.33162588 5.25856516 3.66903594 4.58879912 4.41874907
```

In [73]:

```
from sklearn.metrics import accuracy_score
print(r2_score(ytest, ypred), ": is the accuracy score")
```

```
0.8002827159635503 : is the accuracy score
```

In [74]:

```
from sklearn.model_selection import cross_val_score
ac=r2_score(ytest,ypred)
print(ac)
cv_score=(cross_val_score(model_cv,x,y,cv=5).mean())
print("\n Cross validation Score:",cv_score)
#Difference of R2 score minus CV score
difference=ac-cv_score
print("\n R2 Score - Cross Validation score is", difference)
```

```
0.8002827159635503
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits

 Cross validation Score: 0.7914116268305903

 R2 Score - Cross Validation score is 0.008871089132959975
```