In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import pylab
import statsmodels.stats.api as sms
sns.set()
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Here im imprting all my packages. I may not use them all but it saves me from doing it later.

In [2]:

```
df = pd.read_csv('/Users/conne/Downloads/used_device_data.csv')
```

Here I'm loading my data

In [3]:

```
df.info()
df.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   brand_name             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   main_camera_mp         3275 non-null   float64
 6   selfie_camera_mp       3452 non-null   float64
 7   int_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
```

In [4]:

```
df.describe()
```

Out[4]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	ba
count	3454.000000	3275.000000	3452.000000	3450.000000	3450.000000	3448.00
mean	13.713115	9.460208	6.554229	54.573099	4.036122	3133.40
std	3.805280	4.815461	6.970372	84.972371	1.365105	1299.68
min	5.080000	0.080000	0.000000	0.010000	0.020000	500.00
25%	12.700000	5.000000	2.000000	16.000000	4.000000	2100.00
50%	12.830000	8.000000	5.000000	32.000000	4.000000	3000.00
75%	15.340000	13.000000	8.000000	64.000000	4.000000	4000.00
max	30.710000	48.000000	32.000000	1024.000000	12.000000	9720.00

In [5]:

```
df.value_counts('main_camera_mp')
df.value_counts('selfie_camera_mp')
df.value_counts('int_memory')
df.value_counts('ram')
df.value_counts('battery')
df.value_counts('weight')
```

Out[5]:

```
weight
150.0    112
140.0     86
160.0     80
145.0     68
155.0     68
        ...
159.9      1
158.8      1
158.6      1
158.4      1
855.0      1
Length: 555, dtype: int64
```

Here I want to see what data is presenting as null

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

Out[6]:
```
screen_size                12.830000
main_camera_mp              8.000000
selfie_camera_mp            5.000000
int_memory                 32.000000
ram                         4.000000
battery                  3000.000000
weight                    160.000000
release_year             2015.500000
days_used                 690.500000
normalized_used_price       4.405133
normalized_new_price        5.245892
dtype: float64
```

In [7]:
```
median_mcm = df['main_camera_mp'].median()
df['main_camera_mp'] = df['main_camera_mp'].fillna(median_mcm)
median_scm = df['selfie_camera_mp'].median()
df['selfie_camera_mp'] = df['selfie_camera_mp'].fillna(median_scm)
median_mem = df['int_memory'].median()
df['int_memory'] = df['int_memory'].fillna(median_mem)
median_ram = df['ram'].median()
df['ram'] = df['ram'].fillna(median_ram)
median_bat = df['battery'].median()
df['battery'] = df['battery'].fillna(median_bat)
median_wei = df['weight'].median()
df['weight'] = df['weight'].fillna(median_wei)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   brand_name             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   main_camera_mp         3454 non-null   float64
 6   selfie_camera_mp       3454 non-null   float64
 7   int_memory             3454 non-null   float64
 8   ram                    3454 non-null   float64
 9   battery                3454 non-null   float64
 10  weight                 3454 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
```

Here I'm fixing my null data and replacing it with the median.

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   brand_name             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   main_camera_mp         3454 non-null   float64
 6   selfie_camera_mp       3454 non-null   float64
 7   int_memory             3454 non-null   float64
 8   ram                    3454 non-null   float64
 9   battery                3454 non-null   float64
 10  weight                 3454 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
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['brand_name', 'os', 'screen_size', '4g', '5g', 'main_camera_mp',
       'selfie_camera_mp', 'int_memory', 'ram', 'battery', 'weight',
       'release_year', 'days_used', 'normalized_used_price',
       'normalized_new_price'],
      dtype='object')
```

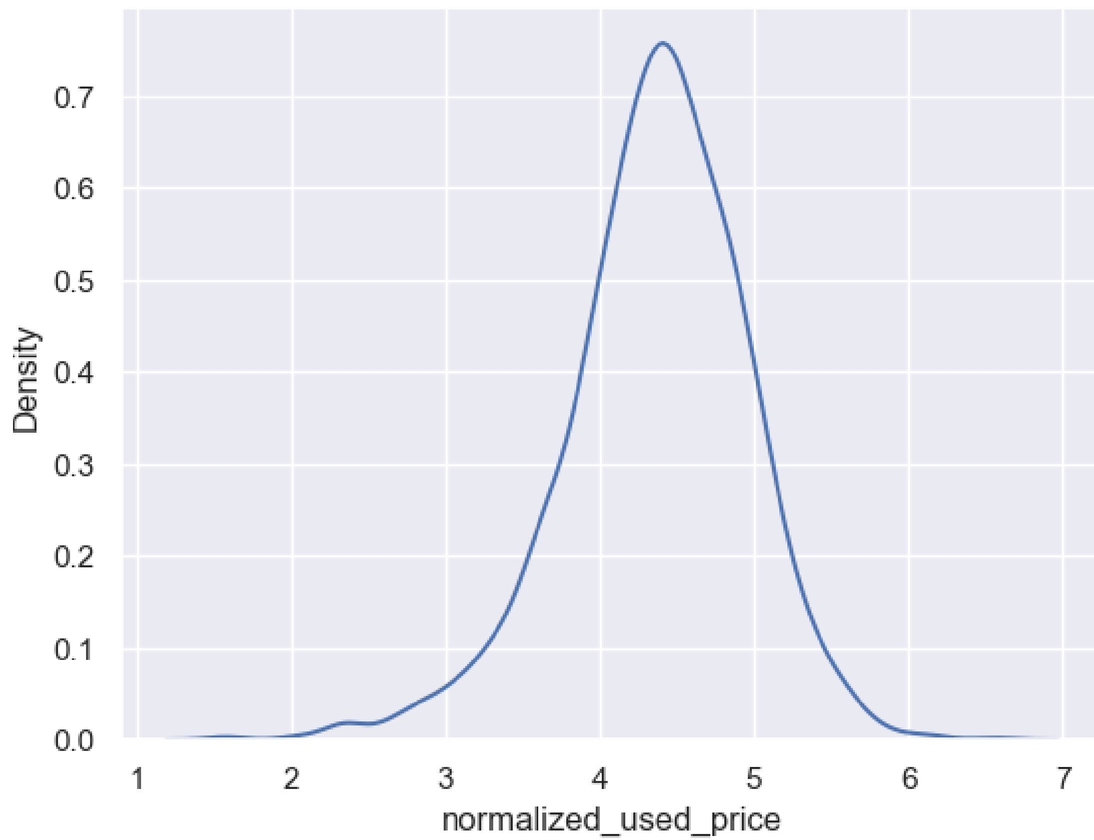
In [10]:

```
sns.pairplot(df[['brand_name', 'os', 'screen_size', '4g', '5g', 'main_came
       'selfie_camera_mp', 'int_memory', 'ram', 'battery', 'weight',
       'release_year', 'days_used', 'normalized_used_price',
       'normalized_new_price']])
plt.show()
```

The only clear relationship seen above is a positive correlation between normalized used price and normalized new price.

In [11]:
```
sns.kdeplot(data=df, x='normalized_used_price')
```

Out[11]: <Axes: xlabel='normalized_used_price', ylabel='Density'>



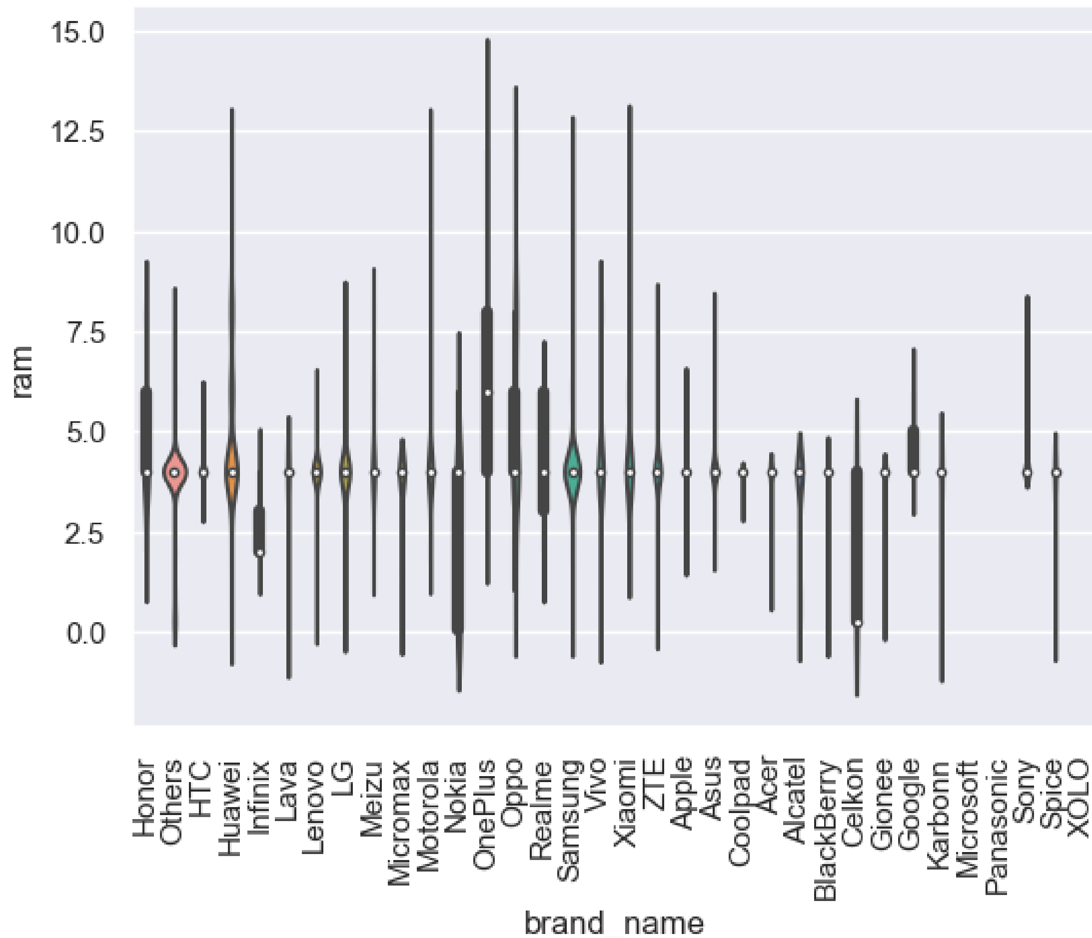
Q1.The chart above shows the disturbution of normalized used price the most prices being between 4 and 5.

In [12]:
```
df.value_counts(subset='os')
```

Out[12]:
```
os
Android    3214
Others      137
Windows      67
iOS          36
dtype: int64
```

Q2. Androids make up 3214/3454 or 93.05% of the market

In [13]:
```
sns.violinplot(data=df, x='brand_name', y='ram', scale='count')
plt.xticks(rotation=90)
plt.show()
```

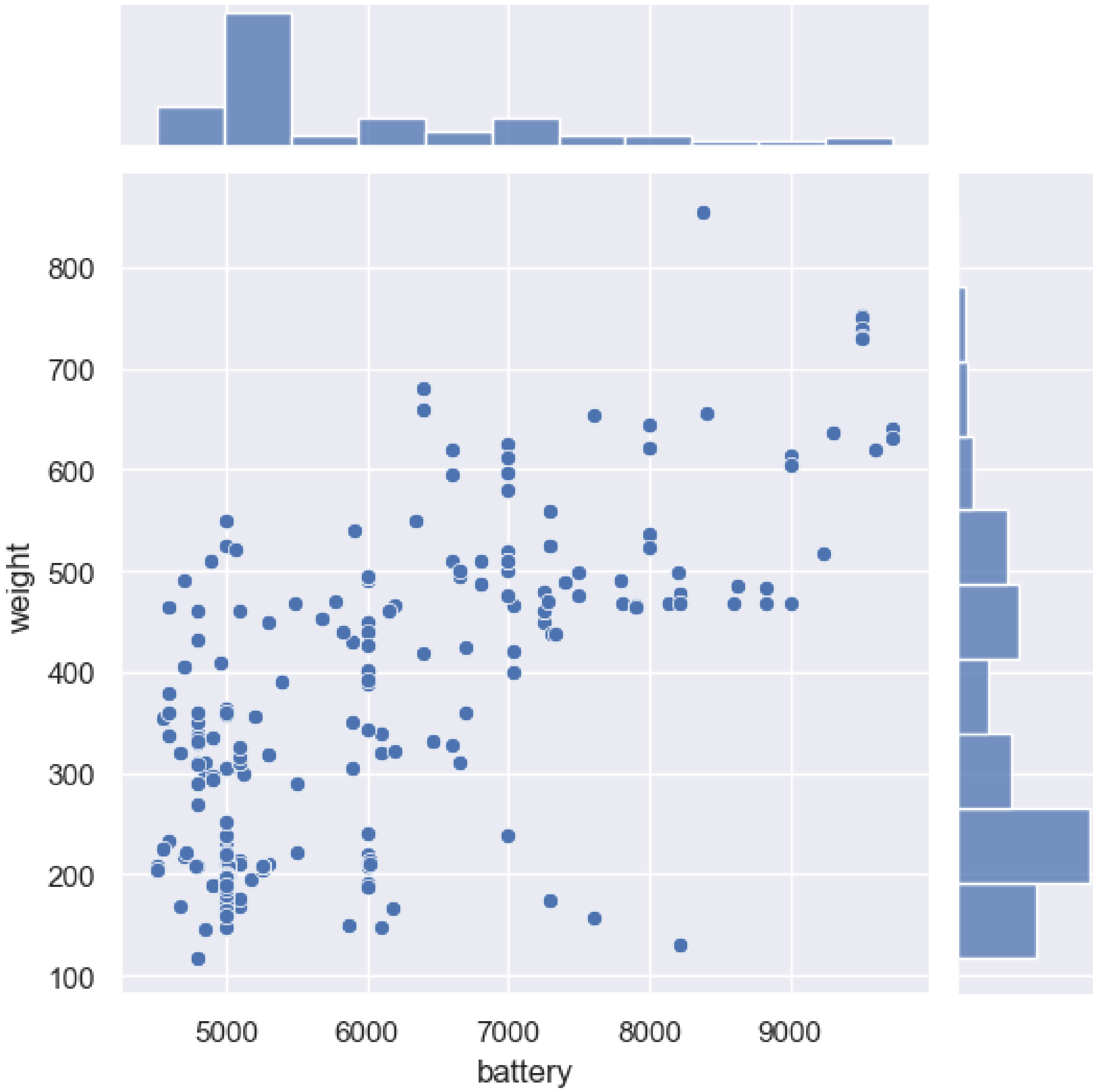


Q3. We can see that ram varies quiet a bit between brands with most brands sticking to a specfic range of ram for most of their devices.

In [14]:

```
df_new = df.copy()
filtered_df = df_new.loc[df_new['battery'] > 4500]
sns.jointplot(data=filtered_df, x='battery', y='weight')
```

Out[14]: <seaborn.axisgrid.JointGrid at 0x193da6e0110>



Q4. We can see that mild increasing battery omh does tend to increase weight but there is still a high degree of variance in weight. This means that while weight will tend to increase with battery size this is not always true.

In [15]:

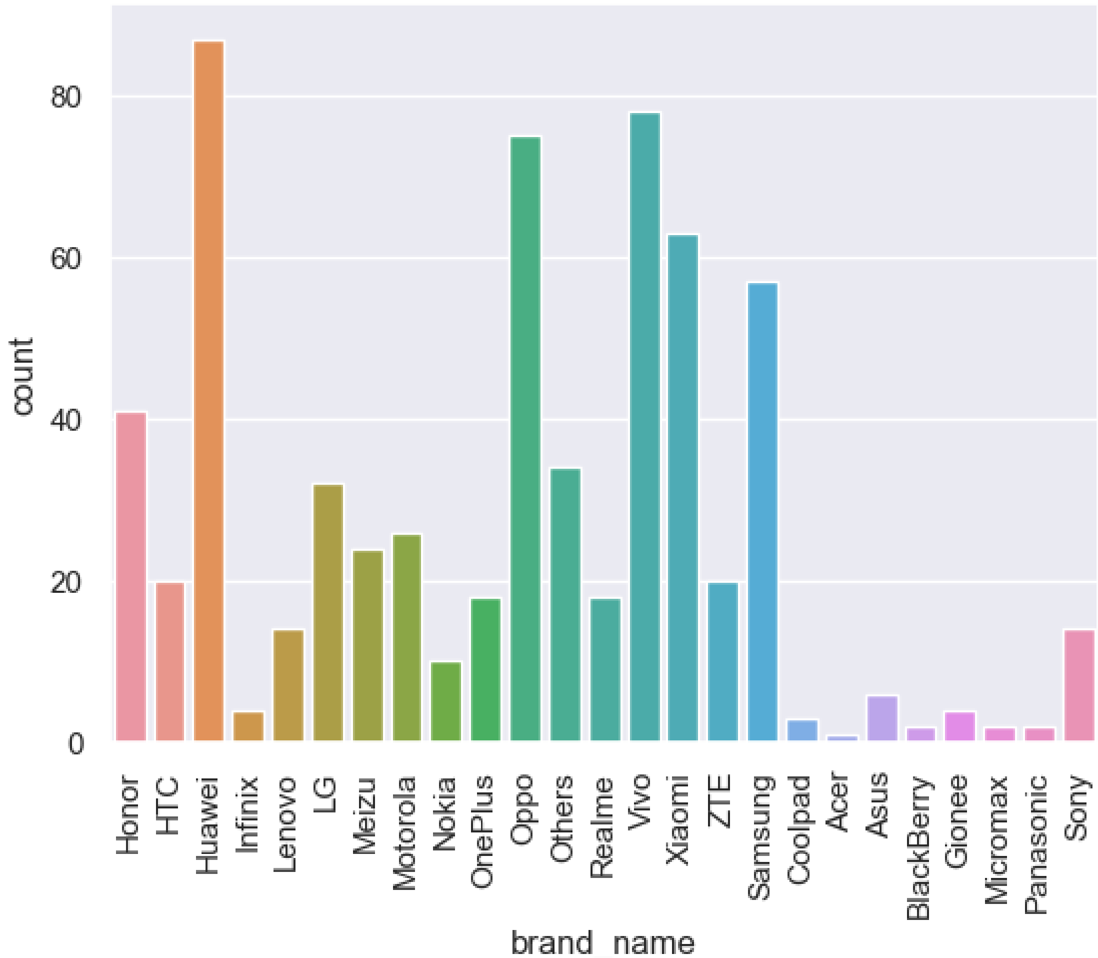
```
scr_df = df.copy()
scr_df = scr_df.loc[scr_df['screen_size'] > 6]
print(scr_df.value_counts('brand_name'))
scr_df.info()
```

```
brand_name
Others        479
Samsung       334
Huawei        251
LG            197
Lenovo        171
ZTE           140
Xiaomi        132
Oppo          129
Asus          122
Vivo          117
Honor         116
Alcatel       115
HTC           110
Micromax      108
Motorola      106
Sony           86
Nokia          72
Meizu          62
```

Q5. We can see that 3362 devices have screens bigger than 6 inches.

In [16]:
```
scmp_df = df.copy()
scmp_df = scmp_df.loc[scmp_df['selfie_camera_mp'] > 8.0]
sns.countplot(data=scmp_df, x='brand_name')
plt.xticks(rotation=90)
scmp_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 655 entries, 1 to 3448
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   brand_name             655 non-null    object 
 1   os                     655 non-null    object 
 2   screen_size            655 non-null    float64
 3   4g                     655 non-null    object 
 4   5g                     655 non-null    object 
 5   main_camera_mp         655 non-null    float64
 6   selfie_camera_mp       655 non-null    float64
 7   int_memory             655 non-null    float64
 8   ram                    655 non-null    float64
 9   battery                655 non-null    float64
 10  weight                 655 non-null    float64
 11  release_year           655 non-null    int64  
 12  days_used              655 non-null    int64  
 13  normalized_used_price  655 non-null    float64
 14  normalized_new_price   655 non-null    float64
dtypes: float64(9), int64(2), object(4)
memory usage: 81.9+ KB
```



Q6. We can see that the distubtion of devices with selfie camera mp above 8 is domatanted by 5 brands.

In [17]:

```
df.corr()
```

Out[17]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	
screen_size	1.000000	0.139385	0.271615	0.071746	0.27
main_camera_mp	0.139385	1.000000	0.373565	0.009507	0.21
selfie_camera_mp	0.271615	0.373565	1.000000	0.296531	0.47
int_memory	0.071746	0.009507	0.296531	1.000000	0.12
ram	0.273810	0.211150	0.477191	0.122774	1.00
battery	0.811240	0.225791	0.369661	0.118108	0.28
weight	0.828872	-0.088483	-0.004688	0.015374	0.08
release_year	0.364223	0.301558	0.690661	0.235166	0.31
days_used	-0.291723	-0.108173	-0.552377	-0.242377	-0.27
normalized_used_price	0.614785	0.552477	0.607548	0.190954	0.51
normalized_new_price	0.460889	0.512655	0.474444	0.196067	0.53

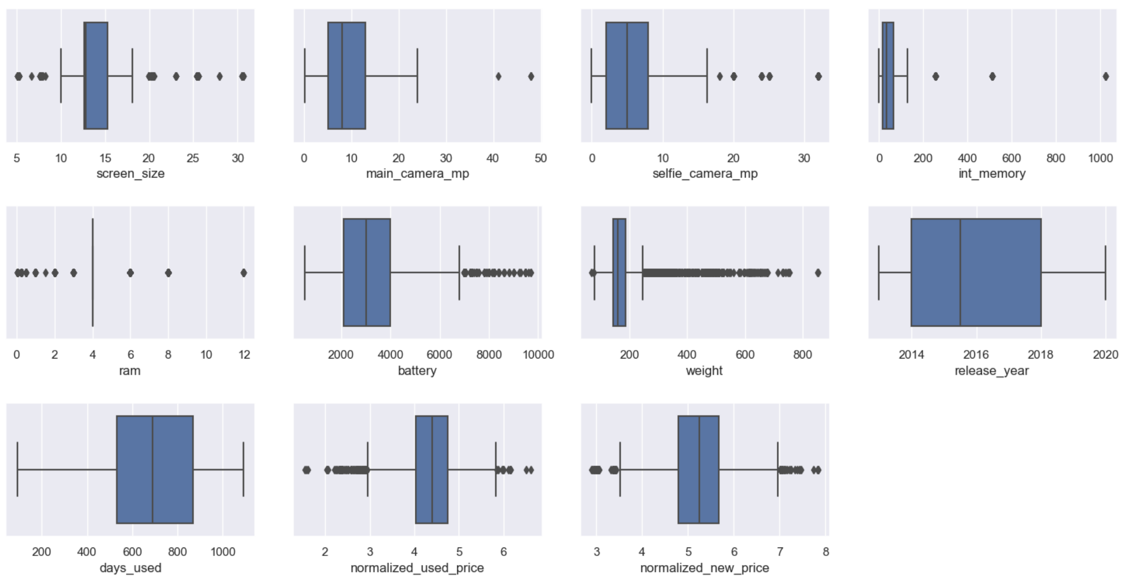
Q7. The columns most correlated with normalized_used_price are in order from highest to lowest are normalized_new_price, screen_size, and battery.

In [18]:
```
df1 = df.copy()
num_cols = df1.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(15, 10))

for i, variable in enumerate(num_cols):
    plt.subplot(4, 4, i + 1)
    sns.boxplot(data=df1, x=variable)
    plt.tight_layout(pad=2)

plt.show()
```



We see a lot of outliers but we wont treat them because they fit logically and will help improve our model.

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   brand_name             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   main_camera_mp         3454 non-null   float64
 6   selfie_camera_mp       3454 non-null   float64
 7   int_memory             3454 non-null   float64
 8   ram                    3454 non-null   float64
 9   battery                3454 non-null   float64
 10  weight                 3454 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
```

In [20]:
```
df = pd.get_dummies(df, columns=["brand_name","os","4g","5g"], drop_first=
df.head()
```

Out[20]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	rele
0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	
1	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	
2	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	
3	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	
4	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	

5 rows × 49 columns

Here I am creating dummy variables for my model

In [21]:
```
# independent variables
X = df.drop(["normalized_used_price"], axis=1)
# dependent variable
y = df[["normalized_used_price"]]
```

Here I am creating my independent and dependent variables.

In [22]:
```
X = sm.add_constant(X)
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
```

In [24]:
```
print(X_train.head())
```

```
      const  screen_size  main_camera_mp  selfie_camera_mp  int_memory
ram  \
3026    1.0        10.29             8.0               0.3        16.0
4.0
1525    1.0        15.34            13.0               5.0        32.0
4.0
1128    1.0        12.70            13.0               5.0        32.0
4.0
3003    1.0        12.83             8.0               5.0        16.0
4.0
2907    1.0        12.88            13.0              16.0        16.0
4.0

      battery  weight  release_year  days_used  ...  brand_name_Spice
\
3026   1800.0   120.0          2014        819  ...                 0
1525   4050.0   225.0          2016        585  ...                 0
1128   2550.0   162.0          2015        727  ...                 0
3003   3200.0   160.0          2015        800  ...                 0
```

In [25]:

```
print(X_test.head())
```

```
      const  screen_size  main_camera_mp  selfie_camera_mp  int_memory  
ram  \
866     1.0        15.24            8.00               2.0        16.0  
4.00
957     1.0        10.16            3.15               0.3       512.0  
0.25
280     1.0        15.39            8.00               8.0        32.0  
2.00
2150    1.0        12.83           13.00              16.0        64.0  
4.00
93      1.0        15.29           13.00               5.0        32.0  
3.00

      battery  weight  release_year  days_used  ...  brand_name_Spice  \
866    3000.0   206.0          2014        632  ...                 0   
957    1400.0   140.0          2013        637  ...                 0   
280    5000.0   185.0          2020        329  ...                 0   
2150   3200.0   148.0          2017        648  ...                 0   
93     3500.0   179.0          2019        216  ...                 0   

      brand_name_Vivo  brand_name_XOLO  brand_name_Xiaomi  brand_name_ZTE  
\
866                 0                0                  0               0   
957                 0                0                  0               0   
280                 0                0                  0               0   
2150                0                0                  0               0   
93                  0                0                  0               0   

      os_Others  os_Windows  os_iOS  4g_yes  5g_yes  
866           0           0       0       0       0  
957           0           0       0       0       0  
280           0           0       0       1       0  
2150          0           0       0       1       0  
93            0           0       0       1       0  

[5 rows x 49 columns]
```

Both x models seem to be correct.

In [26]:
```
olsmod = sm.OLS(y_train, X_train)
olsres = olsmod.fit()
print(olsres.summary())
```

```
                            OLS Regression Results
==============================================================================
========
Dep. Variable:     normalized_used_price   R-squared:
0.845
Model:                               OLS   Adj. R-squared:
0.842
Method:                    Least Squares   F-statistic:
268.8
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:18   Log-Likelihood:
124.22
No. Observations:                   2417   AIC:
-150.4
Df Residuals:                       2368   BIC:
133.3
Df Model:                             48
Covariance Type:               nonrobust
==============================================================================
================
                           coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
----------------
const                  -48.6977      9.184     -5.303      0.000     -6
6.707     -30.689
screen_size              0.0243      0.003      7.145      0.000
0.018       0.031
main_camera_mp           0.0203      0.001     13.806      0.000
0.017       0.023
selfie_camera_mp         0.0136      0.001     12.084      0.000
0.011       0.016
int_memory               0.0001   6.97e-05      1.542      0.123   -2.92
e-05       0.000
ram                      0.0239      0.005      4.657      0.000
0.014       0.034
battery              -1.585e-05   7.27e-06     -2.181      0.029   -3.01
e-05    -1.6e-06
weight                   0.0010      0.000      7.421      0.000
0.001       0.001
release_year             0.0248      0.005      5.441      0.000
0.016       0.034
days_used             3.485e-05   3.09e-05      1.127      0.260   -2.58
e-05    9.55e-05
normalized_new_price     0.4310      0.012     35.133      0.000
0.407       0.455
brand_name_Alcatel       0.0153      0.048      0.321      0.748      -
0.078       0.109
brand_name_Apple        -0.0116      0.147     -0.079      0.937      -
0.300       0.277
brand_name_Asus          0.0195      0.048      0.408      0.683      -
0.074       0.113
brand_name_BlackBerry   -0.0295      0.070     -0.420      0.675      -
0.167       0.108
brand_name_Celkon       -0.0424      0.066     -0.640      0.522      -
0.172       0.088
```

```
brand_name_Coolpad         0.0401      0.073      0.551      0.582      -
0.103       0.183
brand_name_Gionee          0.0454      0.058      0.787      0.431      -
0.068       0.159
brand_name_Google         -0.0312      0.085     -0.369      0.712      -
0.197       0.135
brand_name_HTC            -0.0115      0.048     -0.240      0.811      -
0.106       0.083
brand_name_Honor           0.0244      0.049      0.496      0.620      -
0.072       0.121
brand_name_Huawei         -0.0081      0.044     -0.181      0.856      -
0.095       0.079
brand_name_Infinix         0.1548      0.093      1.661      0.097      -
0.028       0.337
brand_name_Karbonn         0.0971      0.067      1.447      0.148      -
0.034       0.229
brand_name_LG             -0.0152      0.045     -0.335      0.738      -
0.104       0.074
brand_name_Lava            0.0337      0.062      0.541      0.589      -
0.089       0.156
brand_name_Lenovo          0.0449      0.045      0.994      0.320      -
0.044       0.134
brand_name_Meizu           0.0080      0.056      0.143      0.887      -
0.102       0.118
brand_name_Micromax       -0.0335      0.048     -0.700      0.484      -
0.127       0.060
brand_name_Microsoft       0.0945      0.088      1.070      0.285      -
0.079       0.268
brand_name_Motorola        0.0045      0.050      0.091      0.928      -
0.093       0.102
brand_name_Nokia           0.0671      0.052      1.297      0.195      -
0.034       0.169
brand_name_OnePlus         0.1235      0.077      1.596      0.111      -
0.028       0.275
brand_name_Oppo            0.0198      0.048      0.414      0.679      -
0.074       0.113
brand_name_Others         -0.0080      0.042     -0.191      0.849      -
0.091       0.074
brand_name_Panasonic       0.0574      0.056      1.028      0.304      -
0.052       0.167
brand_name_Realme          0.1197      0.061      1.951      0.051      -
0.001       0.240
brand_name_Samsung        -0.0324      0.043     -0.749      0.454      -
0.117       0.052
brand_name_Sony           -0.0493      0.050     -0.979      0.328      -
0.148       0.049
brand_name_Spice          -0.0132      0.063     -0.208      0.835      -
0.137       0.111
brand_name_Vivo           -0.0082      0.048     -0.170      0.865      -
0.103       0.087
brand_name_XOLO            0.0102      0.055      0.187      0.852      -
0.097       0.118
brand_name_Xiaomi          0.0978      0.048      2.034      0.042
0.004       0.192
brand_name_ZTE            -0.0038      0.047     -0.079      0.937      -
0.097       0.089
os_Others                 -0.0513      0.033     -1.566      0.117      -
```

```
0.116       0.013
os_Windows             -0.0176      0.045     -0.389      0.697      -
0.106       0.071
os_iOS                 -0.0585      0.146     -0.399      0.690      -
0.346       0.229
4g_yes                  0.0507      0.016      3.190      0.001
0.020       0.082
5g_yes                 -0.0435      0.032     -1.369      0.171      -
0.106       0.019
==========================================================================
=====
Omnibus:                      217.620   Durbin-Watson:
1.904
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              40
9.702
Skew:                          -0.607   Prob(JB):                     1.0
8e-89
Kurtosis:                       4.611   Cond. No.                     7.6
9e+06
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 7.69e+06. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

When we first fit our model we see a lot a variables thats p-value is to high and thus we will need to come back and correct this. We also see we have an adj. R-squared of.841 which is fairly good.

In [27]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def checking_vif(predictors):
    vif = pd.DataFrame()
    vif["aspect"] = predictors.columns
    vif["VIF"] = [
        variance_inflation_factor(predictors.values, i)
        for i in range(len(predictors.columns))
    ]
    return vif
checking_vif(X_train)
```

Out[27]:

	aspect	VIF
0	const	3.780344e+06
1	screen_size	7.680705e+00
2	main_camera_mp	2.136597e+00
3	selfie_camera_mp	2.808416e+00
4	int_memory	1.361465e+00
5	ram	2.258272e+00
6	battery	4.073582e+00
7	weight	6.380746e+00
8	release_year	4.884645e+00
9	days_used	2.669393e+00
10	normalized_new_price	3.121941e+00
11	brand_name_Alcatel	3.405629e+00
12	brand_name_Apple	1.305691e+01
13	brand_name_Asus	3.330500e+00
14	brand_name_BlackBerry	1.632240e+00
15	brand_name_Celkon	1.773986e+00
16	brand_name_Coolpad	1.466522e+00
17	brand_name_Gionee	1.951248e+00
18	brand_name_Google	1.322242e+00
19	brand_name_HTC	3.409765e+00
20	brand_name_Honor	3.345910e+00
21	brand_name_Huawei	5.986382e+00
22	brand_name_Infinix	1.283540e+00
23	brand_name_Karbonn	1.573183e+00
24	brand_name_LG	4.848734e+00
25	brand_name_Lava	1.711294e+00
26	brand_name_Lenovo	4.559101e+00
27	brand_name_Meizu	2.172894e+00
28	brand_name_Micromax	3.363483e+00
29	brand_name_Microsoft	1.869447e+00
30	brand_name_Motorola	3.259778e+00
31	brand_name_Nokia	3.471596e+00
32	brand_name_OnePlus	1.436575e+00
33	brand_name_Oppo	3.971623e+00
34	brand_name_Others	9.710790e+00
35	brand_name_Panasonic	2.105493e+00

	aspect	VIF
36	brand_name_Realme	1.931102e+00
37	brand_name_Samsung	7.539528e+00
38	brand_name_Sony	2.931789e+00
39	brand_name_Spice	1.688738e+00
40	brand_name_Vivo	3.647700e+00
41	brand_name_XOLO	2.136708e+00
42	brand_name_Xiaomi	3.711997e+00
43	brand_name_ZTE	3.795991e+00
44	os_Others	1.855401e+00
45	os_Windows	1.595333e+00
46	os_iOS	1.178485e+01
47	4g_yes	2.479097e+00
48	5g_yes	1.845023e+00

All my VIF are low so I have no mullticollinarity. I wont need to do anything to change it for this aspect.

In [28]:

```
olsmod_1 = sm.OLS(y_train, X_train)
olsres_1 = olsmod_1.fit()
print(olsres_1.summary())
```

```
                            OLS Regression Results                            
==========================================================================
========
Dep. Variable:     normalized_used_price   R-squared:                       
0.845
Model:                               OLS   Adj. R-squared:                  
0.842
Method:                    Least Squares   F-statistic:                     
268.8
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):              
0.00
Time:                           01:55:18   Log-Likelihood:                  
124.22
No. Observations:                   2417   AIC:                             
-150.4
Df Residuals:                       2368   BIC:                             
133.3
Df Model:                             48                                         
Covariance Type:               nonrobust                                         
==========================================================================
================
                            coef    std err          t      P>|t|      
[0.025      0.975]
--------------------------------------------------------------------------
----------------
const                   -48.6977      9.184     -5.303      0.000     -6
6.707     -30.689
screen_size               0.0243      0.003      7.145      0.000       
0.018       0.031
main_camera_mp            0.0203      0.001     13.806      0.000       
0.017       0.023
selfie_camera_mp          0.0136      0.001     12.084      0.000       
0.011       0.016
int_memory                0.0001   6.97e-05      1.542      0.123   -2.92
e-05       0.000
ram                       0.0239      0.005      4.657      0.000       
0.014       0.034
battery               -1.585e-05   7.27e-06     -2.181      0.029   -3.01
e-05    -1.6e-06
weight                    0.0010      0.000      7.421      0.000       
0.001       0.001
release_year              0.0248      0.005      5.441      0.000       
0.016       0.034
days_used              3.485e-05   3.09e-05      1.127      0.260   -2.58
e-05    9.55e-05
normalized_new_price      0.4310      0.012     35.133      0.000       
0.407       0.455
brand_name_Alcatel        0.0153      0.048      0.321      0.748       -
0.078       0.109
brand_name_Apple         -0.0116      0.147     -0.079      0.937       -
0.300       0.277
brand_name_Asus           0.0195      0.048      0.408      0.683       -
0.074       0.113
brand_name_BlackBerry    -0.0295      0.070     -0.420      0.675       -
0.167       0.108
brand_name_Celkon        -0.0424      0.066     -0.640      0.522       -
0.172       0.088
```

```
brand_name_Coolpad        0.0401      0.073      0.551      0.582      -
0.103       0.183
brand_name_Gionee         0.0454      0.058      0.787      0.431      -
0.068       0.159
brand_name_Google        -0.0312      0.085     -0.369      0.712      -
0.197       0.135
brand_name_HTC           -0.0115      0.048     -0.240      0.811      -
0.106       0.083
brand_name_Honor          0.0244      0.049      0.496      0.620      -
0.072       0.121
brand_name_Huawei        -0.0081      0.044     -0.181      0.856      -
0.095       0.079
brand_name_Infinix        0.1548      0.093      1.661      0.097      -
0.028       0.337
brand_name_Karbonn        0.0971      0.067      1.447      0.148      -
0.034       0.229
brand_name_LG            -0.0152      0.045     -0.335      0.738      -
0.104       0.074
brand_name_Lava           0.0337      0.062      0.541      0.589      -
0.089       0.156
brand_name_Lenovo         0.0449      0.045      0.994      0.320      -
0.044       0.134
brand_name_Meizu          0.0080      0.056      0.143      0.887      -
0.102       0.118
brand_name_Micromax      -0.0335      0.048     -0.700      0.484      -
0.127       0.060
brand_name_Microsoft      0.0945      0.088      1.070      0.285      -
0.079       0.268
brand_name_Motorola       0.0045      0.050      0.091      0.928      -
0.093       0.102
brand_name_Nokia          0.0671      0.052      1.297      0.195      -
0.034       0.169
brand_name_OnePlus        0.1235      0.077      1.596      0.111      -
0.028       0.275
brand_name_Oppo           0.0198      0.048      0.414      0.679      -
0.074       0.113
brand_name_Others        -0.0080      0.042     -0.191      0.849      -
0.091       0.074
brand_name_Panasonic      0.0574      0.056      1.028      0.304      -
0.052       0.167
brand_name_Realme         0.1197      0.061      1.951      0.051      -
0.001       0.240
brand_name_Samsung       -0.0324      0.043     -0.749      0.454      -
0.117       0.052
brand_name_Sony          -0.0493      0.050     -0.979      0.328      -
0.148       0.049
brand_name_Spice         -0.0132      0.063     -0.208      0.835      -
0.137       0.111
brand_name_Vivo          -0.0082      0.048     -0.170      0.865      -
0.103       0.087
brand_name_XOLO           0.0102      0.055      0.187      0.852      -
0.097       0.118
brand_name_Xiaomi         0.0978      0.048      2.034      0.042
0.004       0.192
brand_name_ZTE           -0.0038      0.047     -0.079      0.937      -
0.097       0.089
os_Others                -0.0513      0.033     -1.566      0.117      -
```

```
0.116       0.013
os_Windows               -0.0176      0.045     -0.389      0.697      -
0.106       0.071
os_iOS                   -0.0585      0.146     -0.399      0.690      -
0.346       0.229
4g_yes                    0.0507      0.016      3.190      0.001
0.020       0.082
5g_yes                   -0.0435      0.032     -1.369      0.171      -
0.106       0.019
========================================================================
=====
Omnibus:                      217.620   Durbin-Watson:
1.904
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               40
9.702
Skew:                          -0.607   Prob(JB):                      1.0
8e-89
Kurtosis:                       4.611   Cond. No.                      7.6
9e+06
========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 7.69e+06. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [29]:

```
cols = X_train.columns.tolist()
max_p_value = 1
while len(cols) > 0:
    x_train_aux = X_train[cols]
    model = sm.OLS(y_train, x_train_aux).fit()
    p_values = model.pvalues
    max_p_value = max(p_values)
    feature_with_p_max = p_values.idxmax()
    if max_p_value > 0.05:
        cols.remove(feature_with_p_max)
    else:
        break

passed = cols
print(passed)
```

```
['const', 'screen_size', 'main_camera_mp', 'selfie_camera_mp', 'ram', 'ba
ttery', 'weight', 'release_year', 'normalized_new_price', 'brand_name_Len
ovo', 'brand_name_Nokia', 'brand_name_Realme', 'brand_name_Xiaomi', 'os_O
thers', '4g_yes']
```

Here I am dropping every p-value above .05 in order to remove no signifigant variables.

In [30]:

```
X_train1 = X_train[passed]
X_test1 = X_test[passed]
olsmod3 = sm.OLS(y_train, X_train1).fit()
print(olsmod3.summary())
```

```
                              OLS Regression Results
==============================================================================
===========
Dep. Variable:     normalized_used_price   R-squared:
0.843
Model:                               OLS   Adj. R-squared:
0.842
Method:                    Least Squares   F-statistic:
918.5
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:18   Log-Likelihood:
106.20
No. Observations:                   2417   AIC:
-182.4
Df Residuals:                       2402   BIC:
-95.55
Df Model:                             14
Covariance Type:               nonrobust
```

In [31]:

```
X_train2 = X_train1.drop(["battery"], axis=1)
olsmod_4 = sm.OLS(y_train, X_train2)
olsres_4 = olsmod_4.fit()
print(olsres_4.summary())
```

```
                              OLS Regression Results
==============================================================================
===========
Dep. Variable:     normalized_used_price   R-squared:
0.842
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
987.5
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:18   Log-Likelihood:
104.09
No. Observations:                   2417   AIC:
-180.2
Df Residuals:                       2403   BIC:
-99.11
Df Model:                             13
Covariance Type:               nonrobust
```

In [32]:
```
X_train3 = X_train2.drop(["brand_name_Nokia"], axis=1)
olsmod_5 = sm.OLS(y_train, X_train3)
olsres_5 = olsmod_5.fit()
print(olsres_5.summary())
```

```
                            OLS Regression Results
======================================================================
===========
Dep. Variable:     normalized_used_price   R-squared:
0.842
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
1068.
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:18   Log-Likelihood:
101.51
No. Observations:                   2417   AIC:
-177.0
Df Residuals:                       2404   BIC:
-101.7
Df Model:                             12
Covariance Type:               nonrobust
```

In [33]:

```
X_train4 = X_train3.drop(["os_Others"], axis=1)
olsmod_6 = sm.OLS(y_train, X_train4)
olsres_6 = olsmod_6.fit()
print(olsres_6.summary())
```

```
                            OLS Regression Results
==========================================================================
========
Dep. Variable:     normalized_used_price   R-squared:
0.842
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
1163.
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:19   Log-Likelihood:
99.656
No. Observations:                   2417   AIC:
-175.3
Df Residuals:                       2405   BIC:
-105.8
Df Model:                             11
Covariance Type:               nonrobust
==========================================================================
===============
                          coef    std err          t      P>|t|      [0.
025      0.975]
--------------------------------------------------------------------------
---------------
const                 -39.8838      7.003     -5.695      0.000     -53.
617     -26.150
screen_size             0.0256      0.003      8.674      0.000       0.
020       0.031
main_camera_mp          0.0206      0.001     15.358      0.000       0.
018       0.023
selfie_camera_mp        0.0138      0.001     13.105      0.000       0.
012       0.016
ram                     0.0212      0.004      4.916      0.000       0.
013       0.030
weight                  0.0008      0.000      6.621      0.000       0.
001       0.001
release_year            0.0204      0.003      5.871      0.000       0.
014       0.027
normalized_new_price    0.4238      0.011     39.394      0.000       0.
403       0.445
brand_name_Lenovo       0.0455      0.021      2.120      0.034       0.
003       0.088
brand_name_Realme       0.0983      0.045      2.170      0.030       0.
009       0.187
brand_name_Xiaomi       0.0929      0.025      3.652      0.000       0.
043       0.143
4g_yes                  0.0429      0.015      2.891      0.004       0.
014       0.072
==========================================================================
=====
Omnibus:                     232.186   Durbin-Watson:
1.906
Prob(Omnibus):                 0.000   Jarque-Bera (JB):                44
3.950
Skew:                         -0.635   Prob(JB):                       3.9
6e-97
```

```
Kurtosis:                       4.671   Cond. No.                       2.9
9e+06
==============================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 2.99e+06. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [34]:

```
X_train5 = X_train4.drop(["brand_name_Lenovo"], axis=1)
olsmod_7 = sm.OLS(y_train, X_train5)
olsres_7 = olsmod_7.fit()
print(olsres_7.summary())
```

```
                             OLS Regression Results
==============================================================================
========
Dep. Variable:     normalized_used_price   R-squared:
0.841
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
1277.
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:19   Log-Likelihood:
97.399
No. Observations:                   2417   AIC:
-172.8
Df Residuals:                       2406   BIC:
-109.1
Df Model:                             10
Covariance Type:               nonrobust
==============================================================================
===============
                          coef    std err          t      P>|t|      [0.
025      0.975]
------------------------------------------------------------------------------
---------------
const                 -39.2647      7.002     -5.607      0.000     -52.
996     -25.533
screen_size             0.0259      0.003      8.808      0.000       0.
020       0.032
main_camera_mp          0.0206      0.001     15.408      0.000       0.
018       0.023
selfie_camera_mp        0.0138      0.001     13.083      0.000       0.
012       0.016
ram                     0.0213      0.004      4.936      0.000       0.
013       0.030
weight                  0.0008      0.000      6.631      0.000       0.
001       0.001
release_year            0.0201      0.003      5.784      0.000       0.
013       0.027
normalized_new_price    0.4220      0.011     39.320      0.000       0.
401       0.443
brand_name_Realme       0.0959      0.045      2.117      0.034       0.
007       0.185
brand_name_Xiaomi       0.0905      0.025      3.557      0.000       0.
041       0.140
4g_yes                  0.0433      0.015      2.914      0.004       0.
014       0.072
==============================================================================
=====
Omnibus:                     237.168   Durbin-Watson:
1.903
Prob(Omnibus):                 0.000   Jarque-Bera (JB):                 45
3.649
Skew:                         -0.647   Prob(JB):                        3.1
0e-99
Kurtosis:                      4.683   Cond. No.                        2.9
9e+06
```

```
==============================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 2.99e+06. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [35]:

```
X_train6 = X_train5.drop(["brand_name_Realme"], axis=1)
olsmod_8 = sm.OLS(y_train, X_train6)
olsres_8 = olsmod_8.fit()
print(olsres_8.summary())
```

```
                              OLS Regression Results
==============================================================================
========
Dep. Variable:     normalized_used_price   R-squared:
0.841
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
1416.
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:19   Log-Likelihood:
95.151
No. Observations:                   2417   AIC:
-170.3
Df Residuals:                       2407   BIC:
-112.4
Df Model:                              9
Covariance Type:               nonrobust
==============================================================================
===============
                          coef    std err          t      P>|t|      [0.
025      0.975]
------------------------------------------------------------------------------
---------------
const                 -41.3312      6.939     -5.956      0.000     -54.
938     -27.724
screen_size             0.0261      0.003      8.865      0.000       0.
020       0.032
main_camera_mp          0.0205      0.001     15.305      0.000       0.
018       0.023
selfie_camera_mp        0.0137      0.001     13.001      0.000       0.
012       0.016
ram                     0.0214      0.004      4.942      0.000       0.
013       0.030
weight                  0.0008      0.000      6.576      0.000       0.
001       0.001
release_year            0.0211      0.003      6.136      0.000       0.
014       0.028
normalized_new_price    0.4212      0.011     39.241      0.000       0.
400       0.442
brand_name_Xiaomi       0.0883      0.025      3.473      0.001       0.
038       0.138
4g_yes                  0.0436      0.015      2.929      0.003       0.
014       0.073
==============================================================================
=====
Omnibus:                     223.098   Durbin-Watson:
1.904
Prob(Omnibus):                 0.000   Jarque-Bera (JB):               41
0.667
Skew:                         -0.627   Prob(JB):                      6.6
8e-90
Kurtosis:                      4.583   Cond. No.                      2.9
6e+06
==============================================================================
=====
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 2.96e+06. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

All p-values below .05 have been dropped.

In [36]:

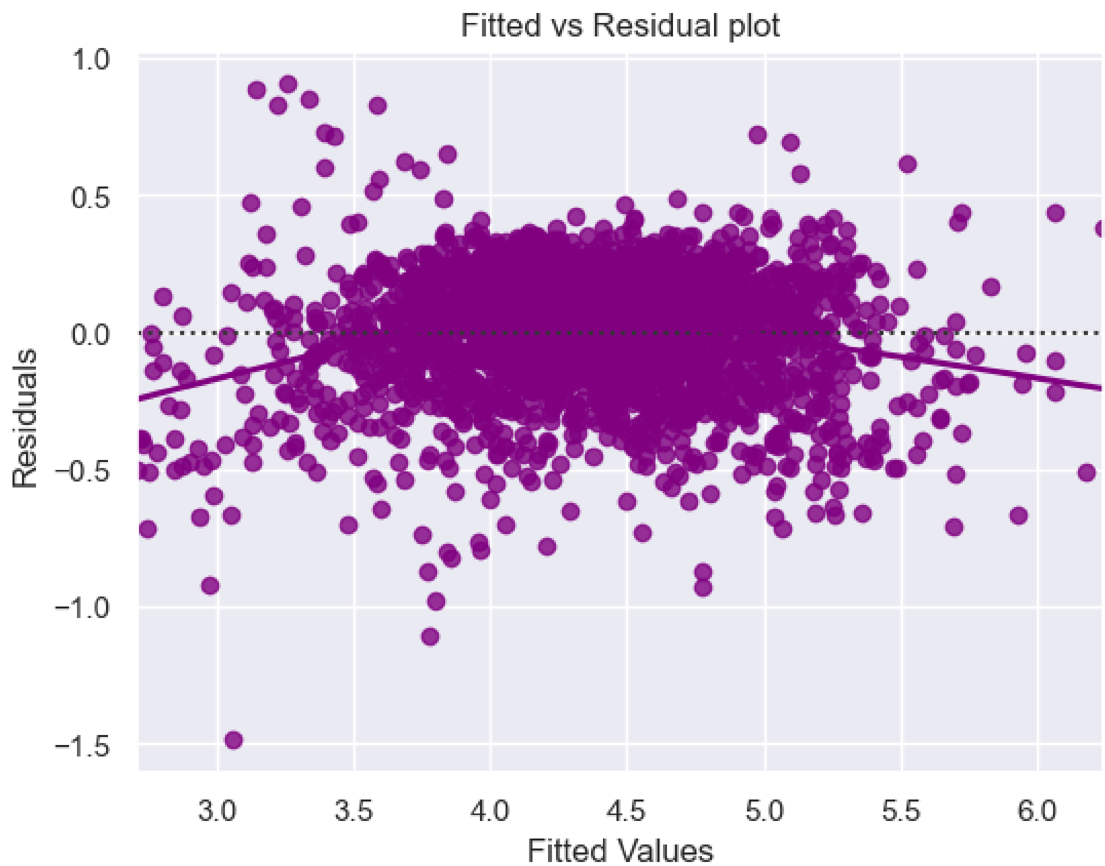
```
df_pred = pd.DataFrame()
df_pred["Actual Values"] = y_train.values.flatten()
df_pred["Fitted Values"] = olsres_8.fittedvalues.values
df_pred["Residuals"] = olsres_8.resid.values
df_pred.head()
```

Out[36]:

	Actual Values	Fitted Values	Residuals
0	4.087488	3.854967	0.232520
1	4.448399	4.589640	-0.141241
2	4.315353	4.282336	0.033016
3	4.282068	4.246969	0.035099
4	4.456438	4.471019	-0.014581

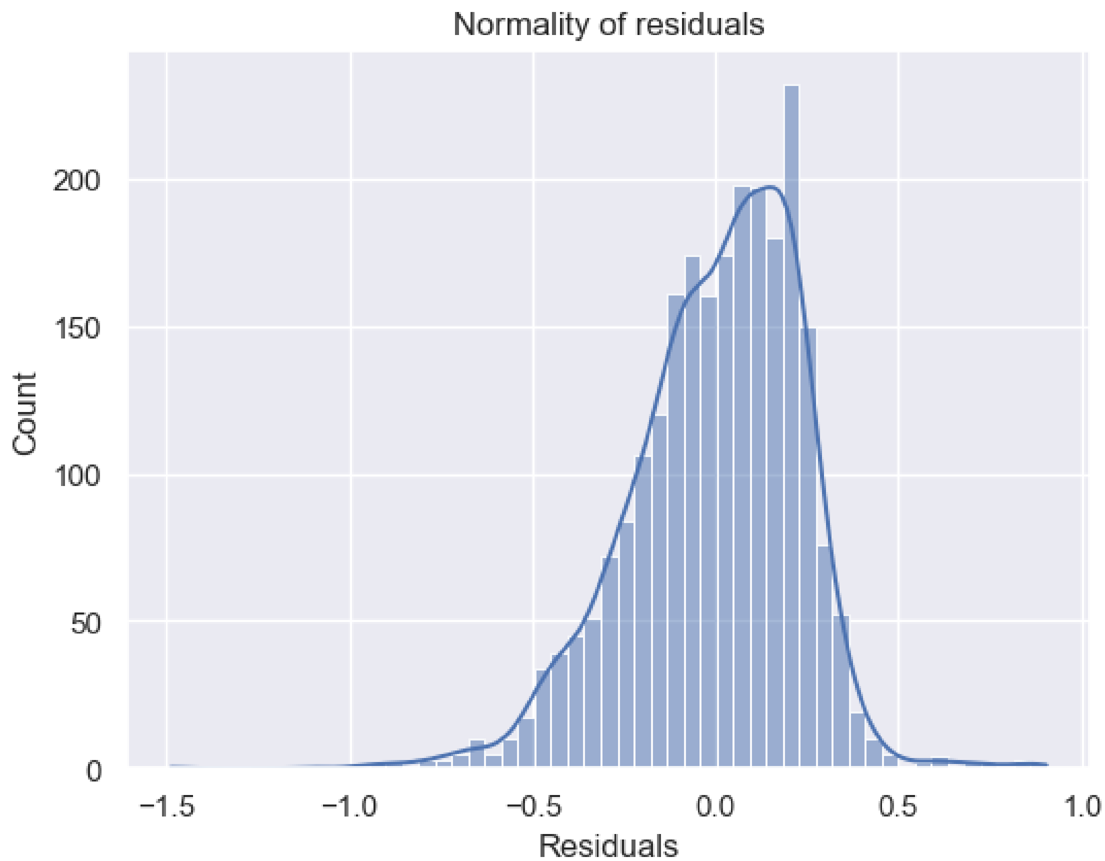
In [37]:

```
sns.residplot(data=df_pred, x="Fitted Values", y="Residuals", color="purpl
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Fitted vs Residual plot")
plt.show()
```



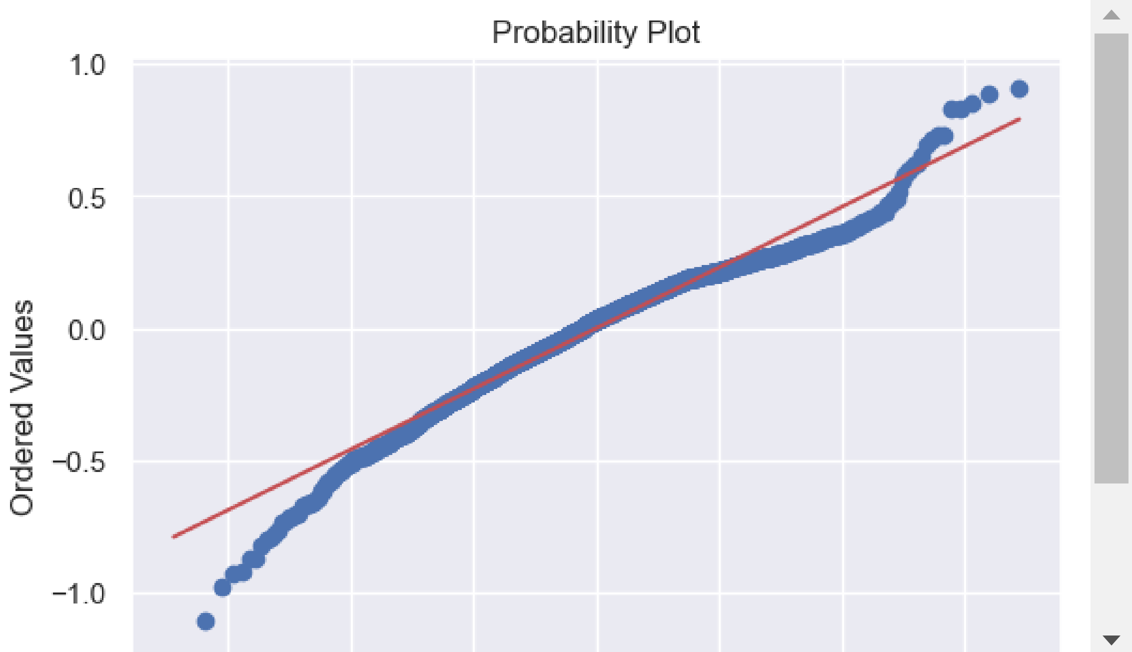
You can see it is independent.

In [38]:
```
sns.histplot(data=df_pred, x="Residuals", kde=True)
plt.title("Normality of residuals")
plt.show()
```



You can see it is normally distrubited.

In [39]:
```
stats.probplot(df_pred["Residuals"], dist="norm", plot=pylab)
plt.show()
```



In [40]:
```
stats.shapiro(df_pred["Residuals"])
```

Out[40]:
```
ShapiroResult(statistic=0.9697744250297546, pvalue=3.7057961548349588e-2
2)
```

though the p-value may indicate innormality visual analysis confirms there is normality.

In [41]:
```
from statsmodels.compat import lzip
name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(df_pred["Residuals"], X_train2)
lzip(name, test)
```

Out[41]:
```
[('F statistic', 1.046771276832646), ('p-value', 0.21484488064429128)]
```

since the p-value is > .05 we can assume homoscedacity

In [42]:
```
olsmodel_final = sm.OLS(y_train, X_train6).fit()
print(olsmodel_final.summary())
```

```
                              OLS Regression Results
======================================================================
===========
Dep. Variable:     normalized_used_price   R-squared:
0.841
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
1416.
Date:                   Sat, 09 Sep 2023   Prob (F-statistic):
0.00
Time:                           01:55:20   Log-Likelihood:
95.151
No. Observations:                   2417   AIC:
-170.3
Df Residuals:                       2407   BIC:
-112.4
Df Model:                              9
Covariance Type:               nonrobust
```

```
Since our adj. R-squared is .841 we can explain 84% variance in the data
which is pretty good.
A unit increase of normalized_new_price would result in .421 unit increase in
normalized_used_price

Our model tells us that the most important factor in predicting
normalized_used_price is understanding normalized_new_price
screen_size and main_camera_mp seem to be key factors in predicting
normalized_used_price
It seems if you want to know what a phone will cost you should first look at
what it did cost.
```