Supervised Learning - Foundations Project: ReCell

Problem Statement

Business Context

Buying and selling used phones and tablets used to be something that happened on a handful of online marketplace sites. But the used and refurbished device market has grown considerably over the past decade, and a new IDC (International Data Corporation) forecast predicts that the used phone market would be worth $52.7bn by 2023 with a compound annual growth rate (CAGR) of 13.6% from 2018 to 2023. This growth can be attributed to an uptick in demand for used phones and tablets that offer considerable savings compared with new models.

Refurbished and used devices continue to provide cost-effective alternatives to both consumers and businesses that are looking to save money when purchasing one. There are plenty of other benefits associated with the used device market. Used and refurbished devices can be sold with warranties and can also be insured with proof of purchase. Third-party vendors/platforms, such as Verizon, Amazon, etc., provide attractive offers to customers for refurbished devices. Maximizing the longevity of devices through second-hand trade also reduces their environmental impact and helps in recycling and reducing waste. The impact of the COVID-19 outbreak may further boost this segment as consumers cut back on discretionary spending and buy phones and tablets only for immediate needs.

Objective

The rising potential of this comparatively under-the-radar market fuels the need for an ML-based solution to develop a dynamic pricing strategy for used and refurbished devices. ReCell, a startup aiming to tap the potential in this market, has hired you as a data scientist. They want you to analyze the data provided and build a linear regression model to predict the price of a used phone/tablet and identify factors that significantly influence it.

Data Description

The data contains the different attributes of used/refurbished phones and tablets. The data was collected in the year 2021. The detailed data dictionary is given below.

- brand_name: Name of manufacturing brand
- os: OS on which the device runs
- screen_size: Size of the screen in cm
- 4g: Whether 4G is available or not
- 5g: Whether 5G is available or not
- main_camera_mp: Resolution of the rear camera in megapixels
- selfie_camera_mp: Resolution of the front camera in megapixels
- int_memory: Amount of internal memory (ROM) in GB
- ram: Amount of RAM in GB
- battery: Energy capacity of the device battery in mAh
- weight: Weight of the device in grams
- release_year: Year when the device model was released
- days_used: Number of days the used/refurbished device has been used
- normalized_new_price: Normalized price of a new device of the same model in euros
- normalized_used_price: Normalized price of the used/refurbished device in euros

Please read the instructions carefully before starting the project.

This is a commented Python Notebook file in which all the instructions and tasks to be performed are mentioned.

- Blanks '**___**' are provided in the notebook that need to be filled with an appropriate code to get the correct result

- With every '**___**' blank, there is a comment that briefly describes what needs to be filled in the blank space

- Identify the task to be performed correctly and only then proceed to write the required code

- Fill the code wherever asked by the commented lines like "# write your code here" or "# complete the code"

- Running incomplete code may throw an error

- Please run the codes in a sequential manner from the beginning to avoid any unnecessary errors

- Add the results/observations derived from the analysis in the presentation and submit the same in .pdf format

Importing necessary libraries

In [1]:
```
# this will help in making the Python code more structured automatically
'%load_ext nb_black'

# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

# split the data into train and test
from sklearn.model_selection import train_test_split

# to build linear regression_model
from sklearn.linear_model import LinearRegression

# to check model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_s

# to build linear regression_model using statsmodels
import statsmodels.api as sm

# to compute VIF
from statsmodels.stats.outliers_influence import variance_inflation_facto
```

Loading the dataset

In [2]:
```
# loading data
data = pd.read_csv('used_device_data.csv') ## Complete the code to read t
```

Data Overview

The initial steps to get an overview of any dataset is to:

- observe the first few rows of the dataset, to check whether the dataset has been loaded properly or not
- get information about the number of rows and columns in the dataset
- find out the data types of the columns to ensure that data is stored in the preferred format and the value of each property is as expected.
- check the statistical summary of the dataset to get an overview of the numerical columns of the data

Displaying the first few rows of the dataset

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

Out[3]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int
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	

Checking the shape of the dataset

In [4]:
```
data.shape ## Complete the code to get the shape of data
```

Out[4]:
```
(3454, 15)
```

Checking the data types of the columns for the dataset

In [5]:
```
data.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          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
 14  normalized_new_price    3454 non-null   float64
dtypes: float64(9), int64(2), object(4)
memory usage: 404.9+ KB
```

Statistical summary of the dataset

In [6]:
```
data.describe(include='all').T ## Complete the code to print the statisti
```

Out[6]:

	count	unique	top	freq	mean	std	n
brand_name	3454	34	Others	502	NaN	NaN	N
os	3454	4	Android	3214	NaN	NaN	N
screen_size	3454.0	NaN	NaN	NaN	13.713115	3.80528	5.
4g	3454	2	yes	2335	NaN	NaN	N
5g	3454	2	no	3302	NaN	NaN	N
main_camera_mp	3275.0	NaN	NaN	NaN	9.460208	4.815461	0.
selfie_camera_mp	3452.0	NaN	NaN	NaN	6.554229	6.970372	[illegible]
int_memory	3450.0	NaN	NaN	NaN	54.573099	84.972371	0
ram	3450.0	NaN	NaN	NaN	4.036122	1.365105	0.
battery	3448.0	NaN	NaN	NaN	3133.402697	1299.682844	50
weight	3447.0	NaN	NaN	NaN	182.751871	88.413228	6
release_year	3454.0	NaN	NaN	NaN	2015.965258	2.298455	201
days_used	3454.0	NaN	NaN	NaN	674.869716	248.580166	9
normalized_used_price	3454.0	NaN	NaN	NaN	4.364712	0.588914	1.5368
normalized_new_price	3454.0	NaN	NaN	NaN	5.233107	0.683637	2.9014

Checking for duplicate values

In [7]:
```
data.duplicated().sum()## Complete the code to check duplicate entries in
```

Out[7]: 0

Checking for missing values

In [8]:
```
data.isnull().sum().sort_values(ascending= False) ## Complete the code to
```

Out[8]:
```
main_camera_mp           179
weight                     7
battery                    6
int_memory                 4
ram                        4
selfie_camera_mp           2
brand_name                 0
os                         0
screen_size                0
4g                         0
5g                         0
release_year               0
days_used                  0
normalized_used_price      0
normalized_new_price       0
dtype: int64
```

In [9]:
```
# creating a copy of the data so that original data remains unchanged
df = data.copy()
```

Exploratory Data Analysis

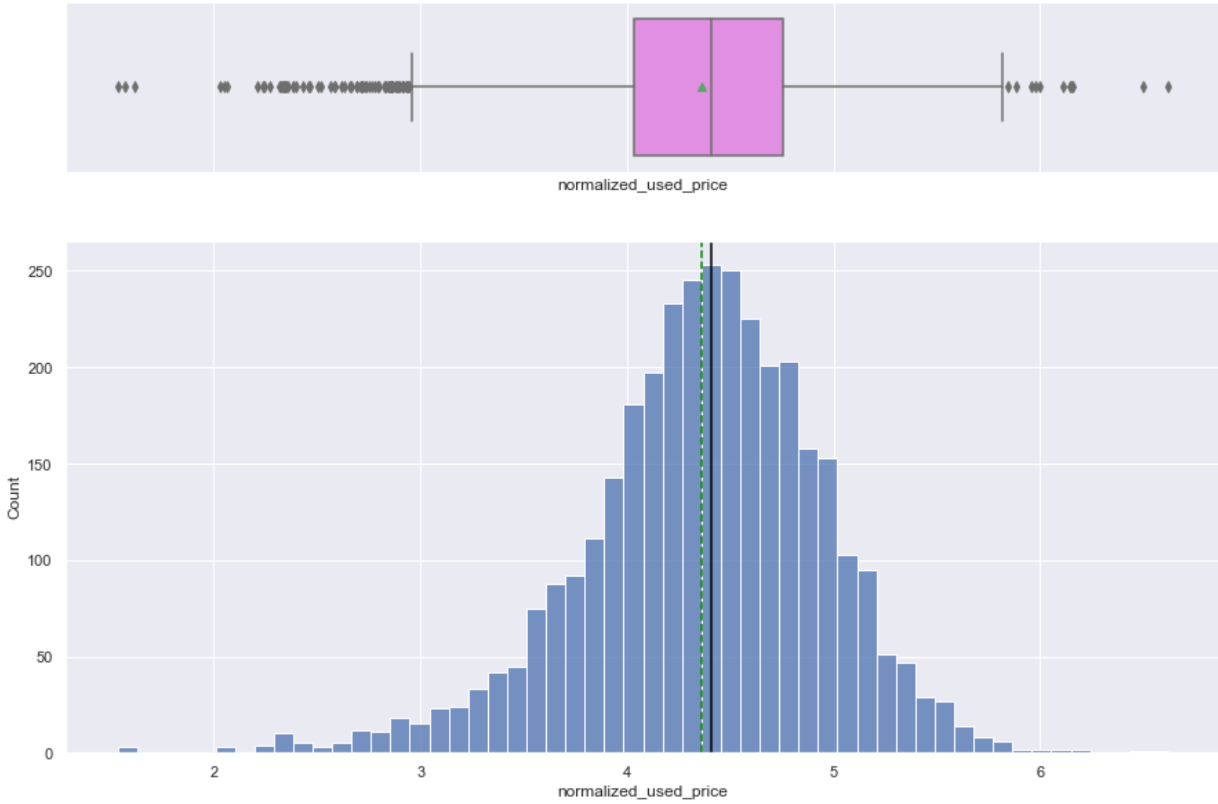
Univariate Analysis

In [10]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=No
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a triangle will indicate the mean va
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [11]:
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is Fal
    n: displays the top n category levels (default is None, i.e., display
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 2, 6))
    else:
        plt.figure(figsize=(n + 2, 6))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n],
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

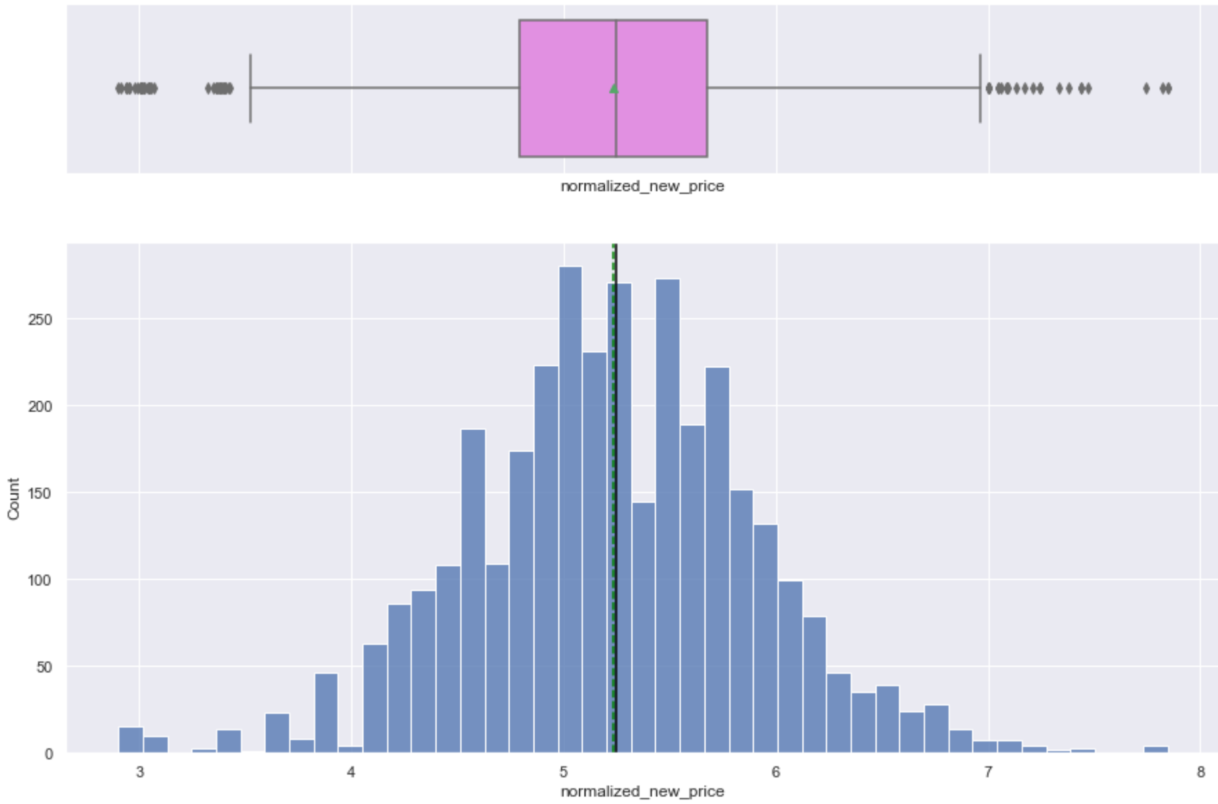
`normalized_used_price`

In [13]:
```
histogram_boxplot(df, "normalized_used_price")
```



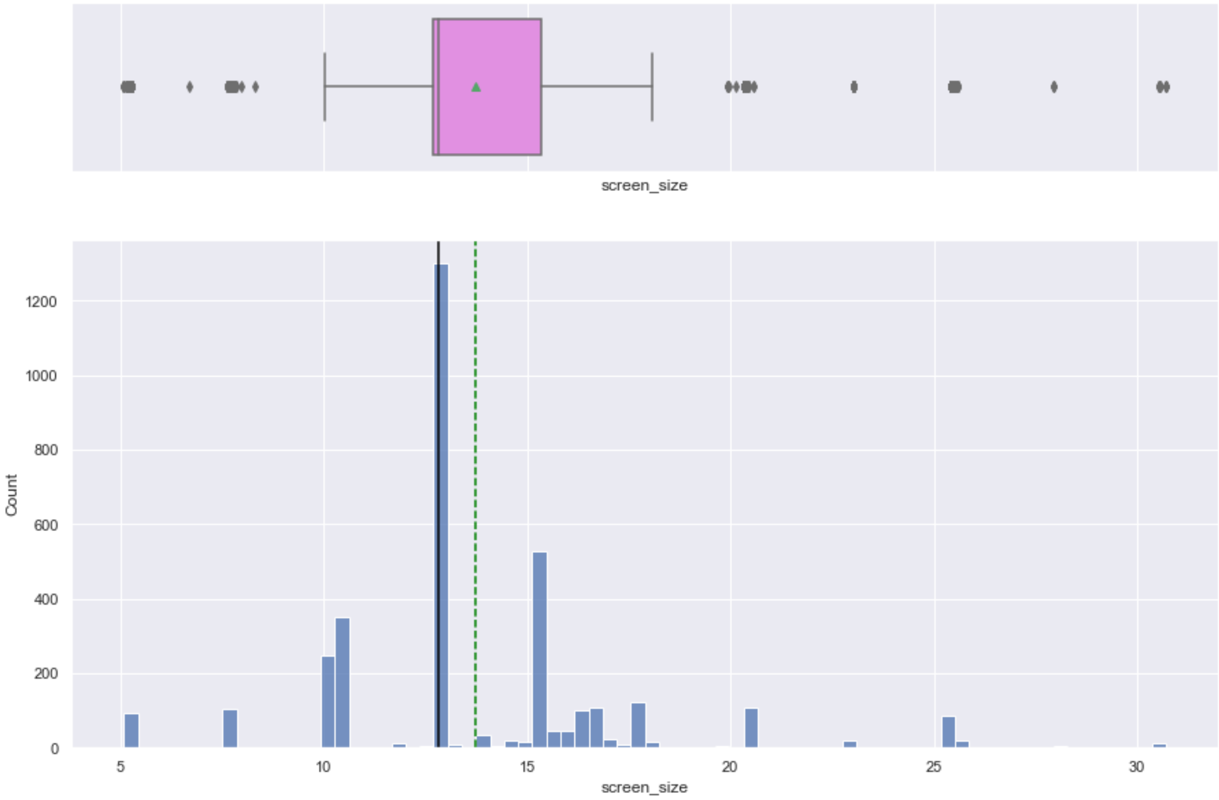
`normalized_new_price`

In [14]:
```
histogram_boxplot(df,'normalized_new_price')  ## Complete the code to cre
```



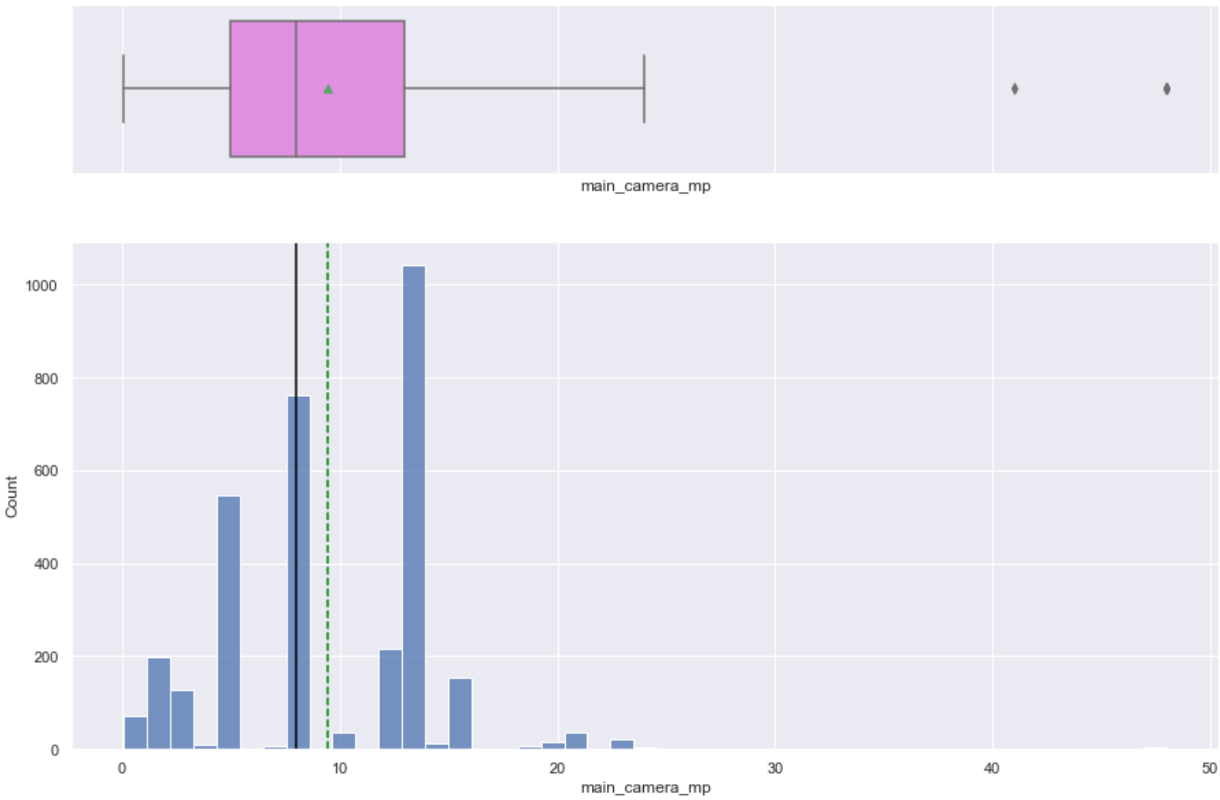
`screen_size`

In [15]:
```
histogram_boxplot(df,'screen_size')  ## Complete the code to create histo
```



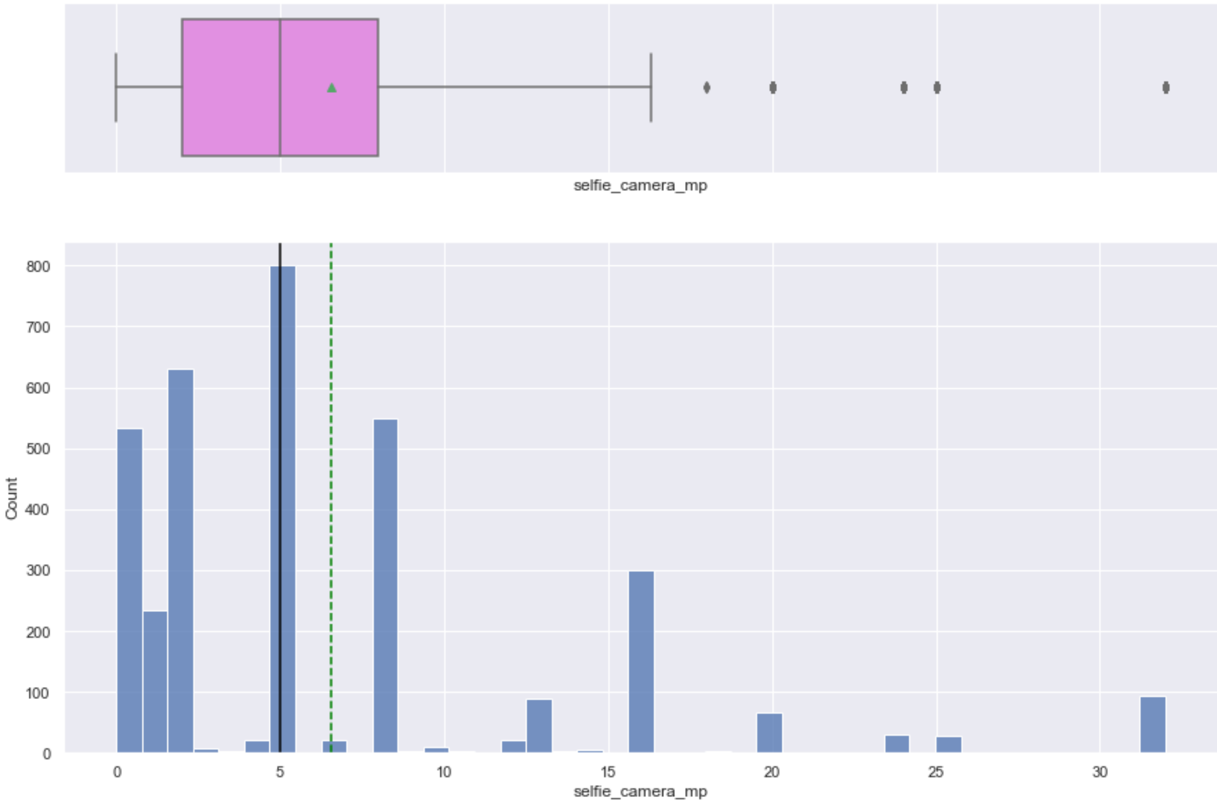
main_camera_mp

In [16]:
```
histogram_boxplot(df,'main_camera_mp')  ## Complete the code to create hi
```



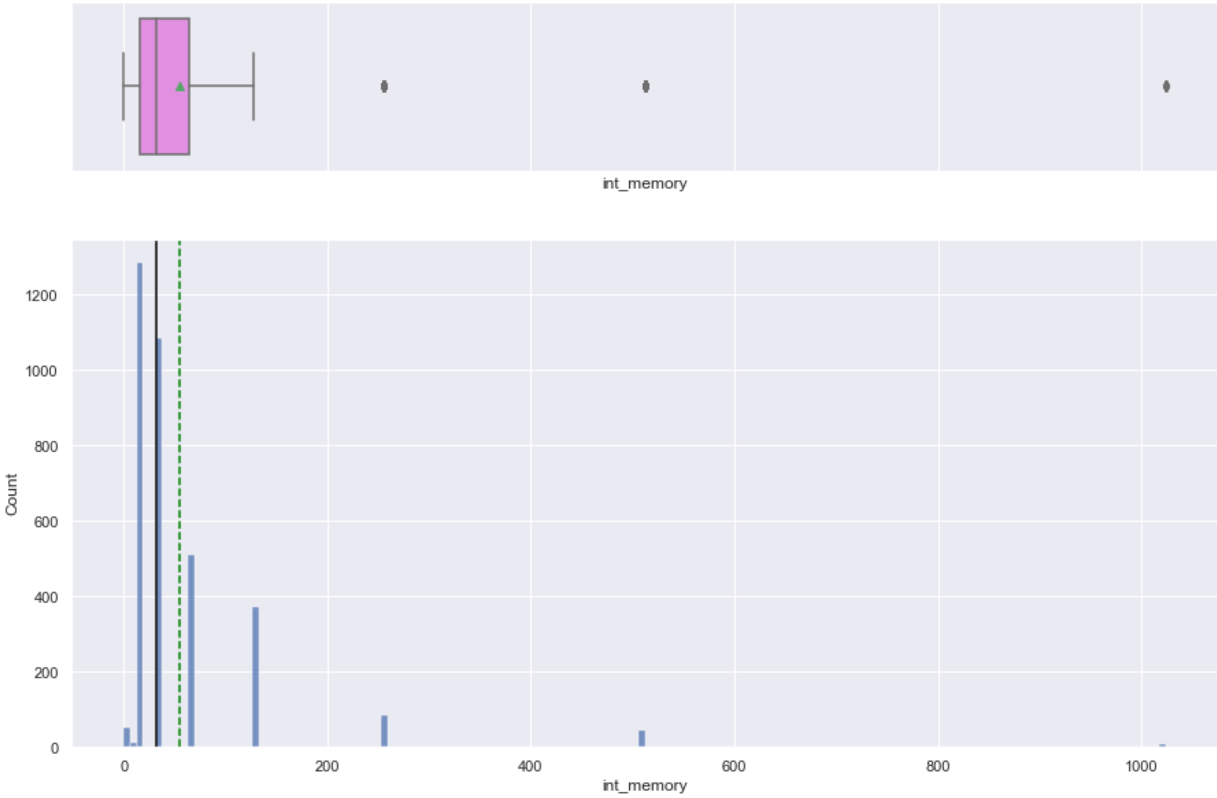
`selfie_camera_mp`

In [17]:
```
histogram_boxplot(df, 'selfie_camera_mp')  ## Complete the code to create
```



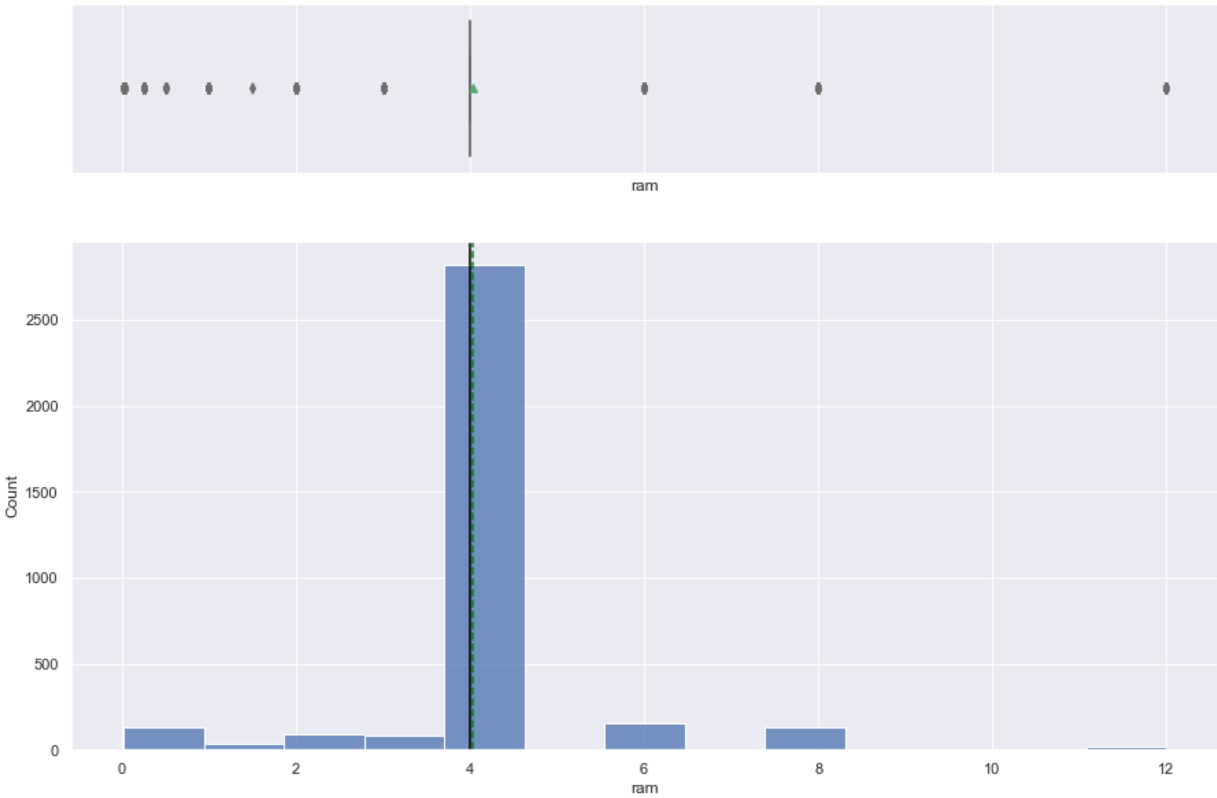
`int_memory`

In [18]:
```
histogram_boxplot(df, 'int_memory')  ## Complete the code to create histo
```



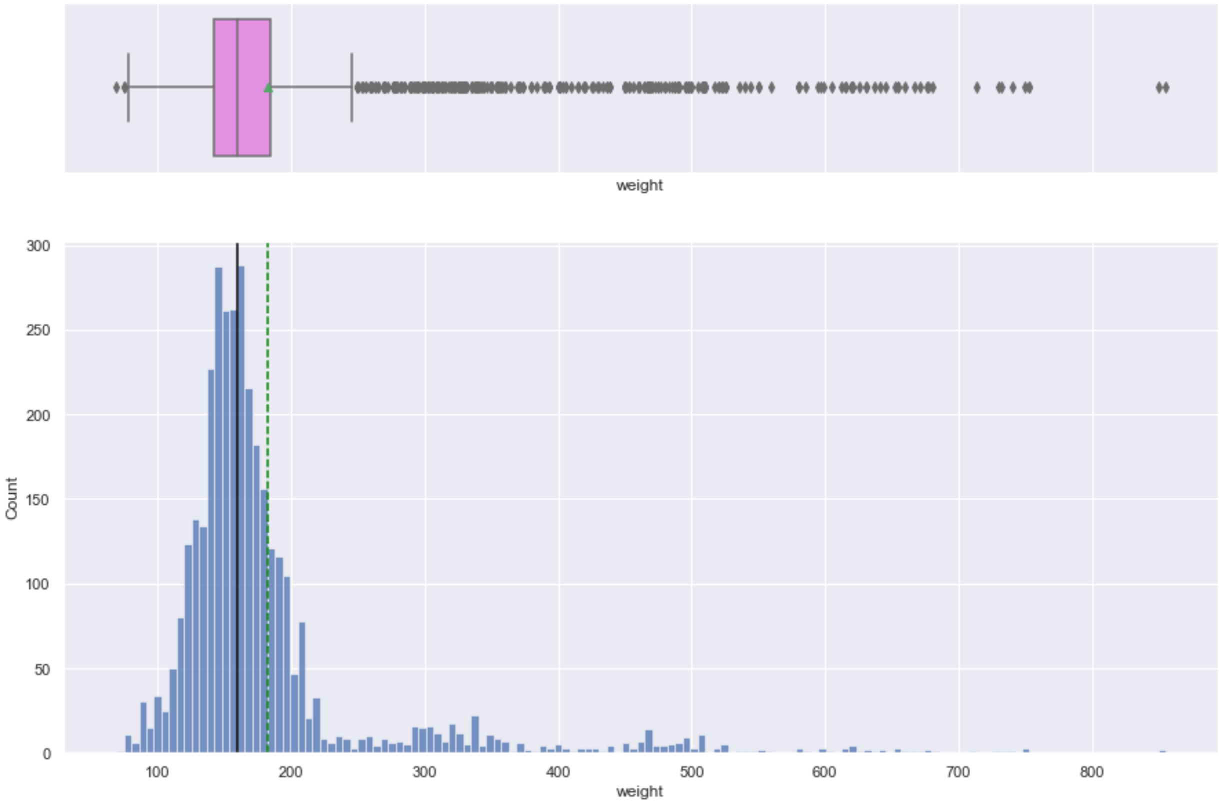
ram

In [19]:
```
histogram_boxplot(df, 'ram')  ## Complete the code to create histogram_bo
```



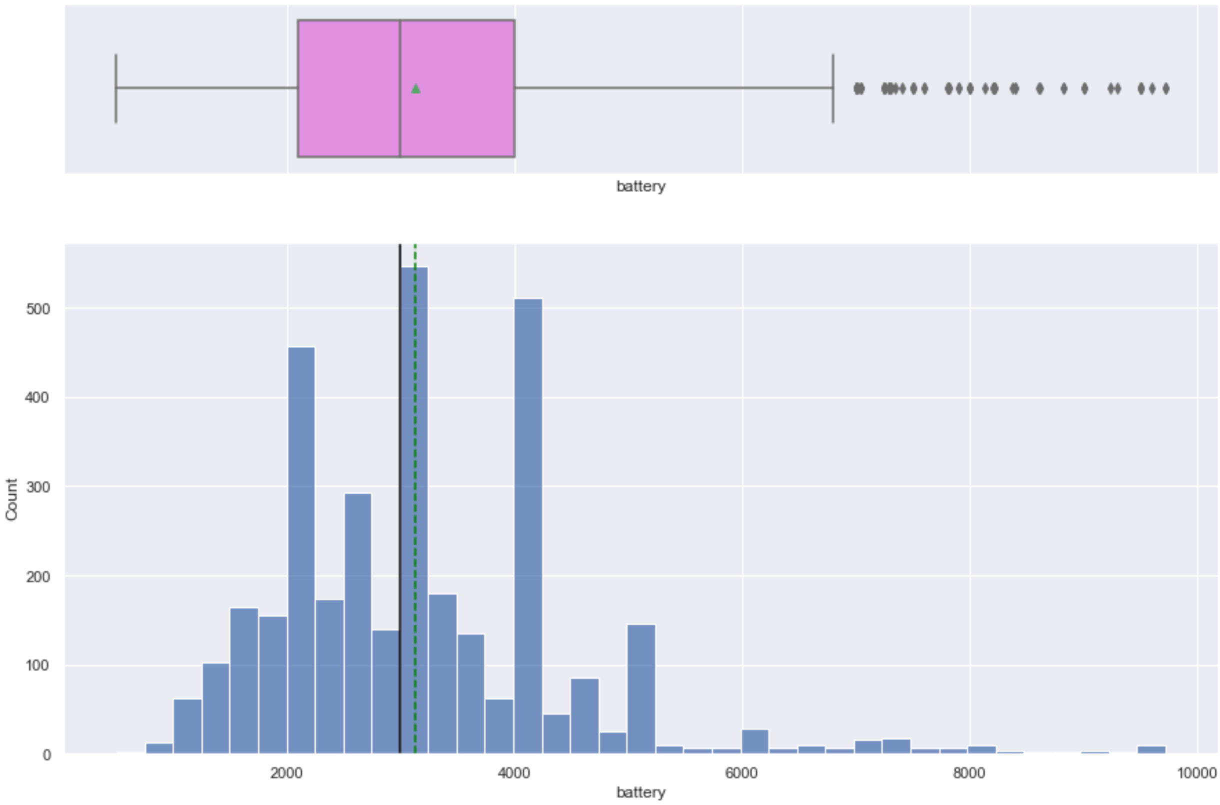
weight

In [20]:
```
histogram_boxplot(df, "weight")  ## Complete the code to create histogram
```



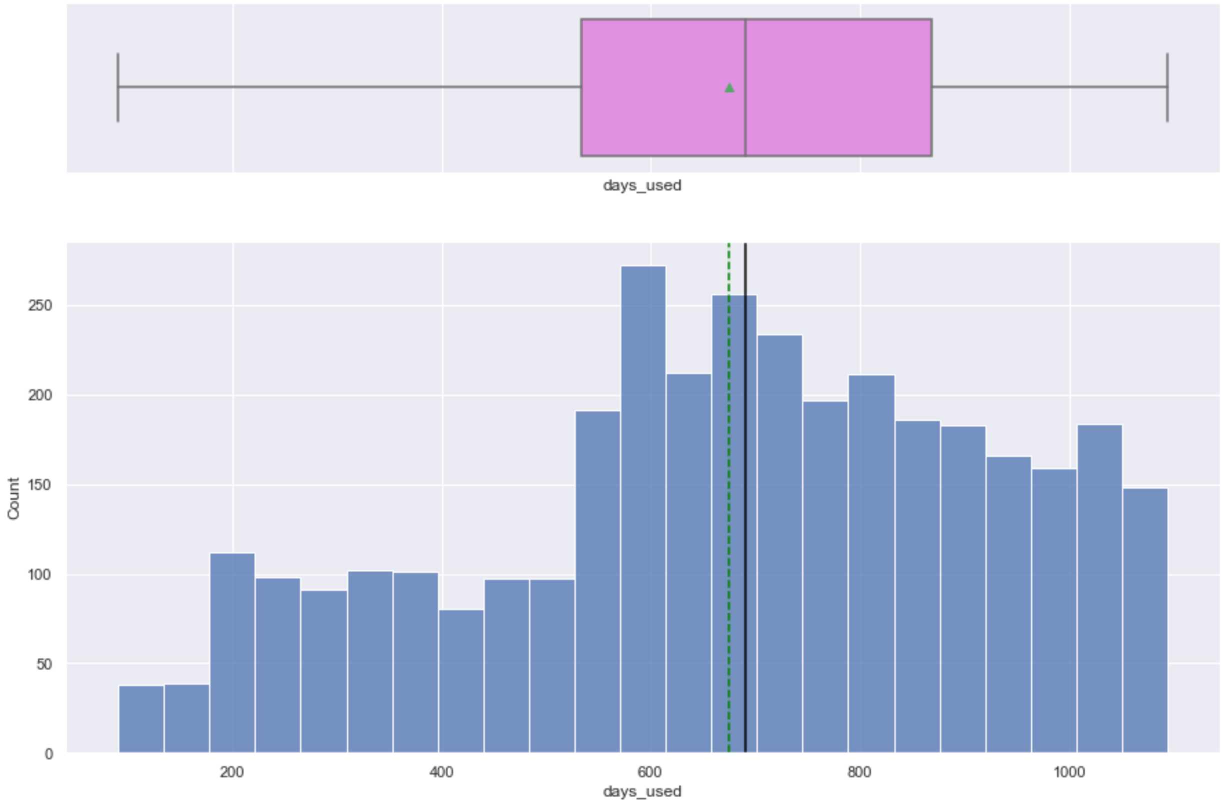
`battery`

In [21]:
```
histogram_boxplot(df,'battery')  ## Complete the code to create histogram
```



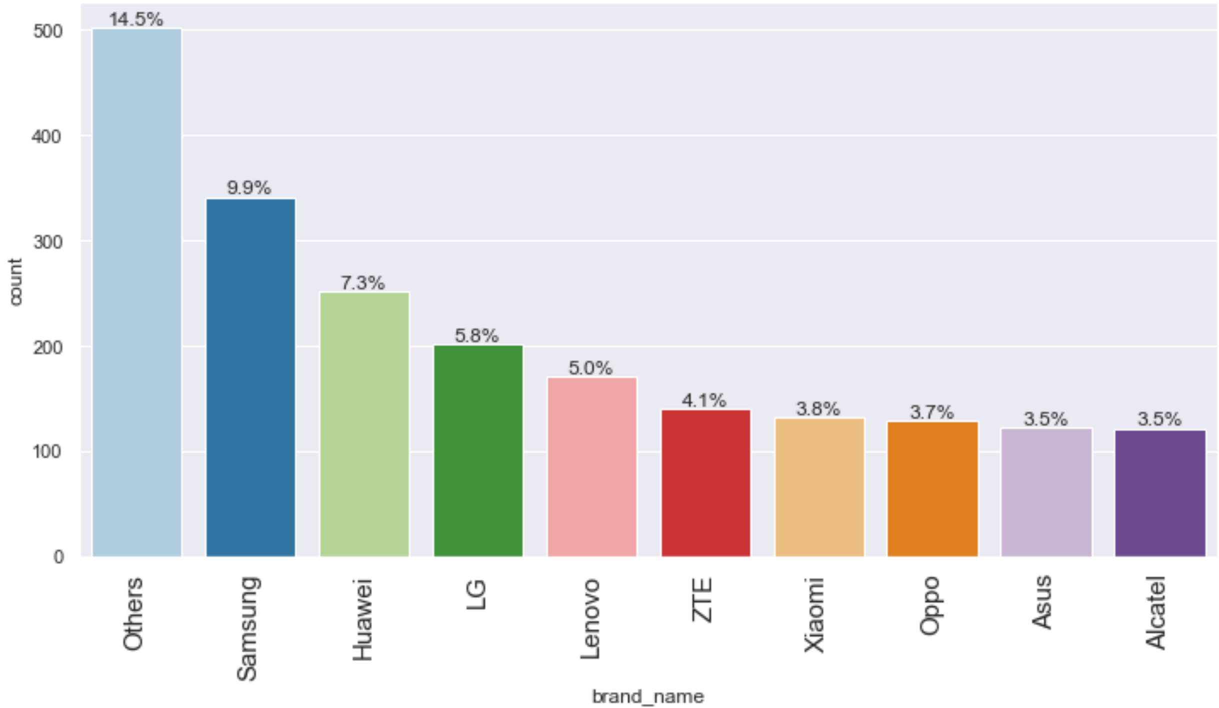
`days_used`

In [22]:
```
histogram_boxplot(df, 'days_used')  ## Complete the code to create histog
```



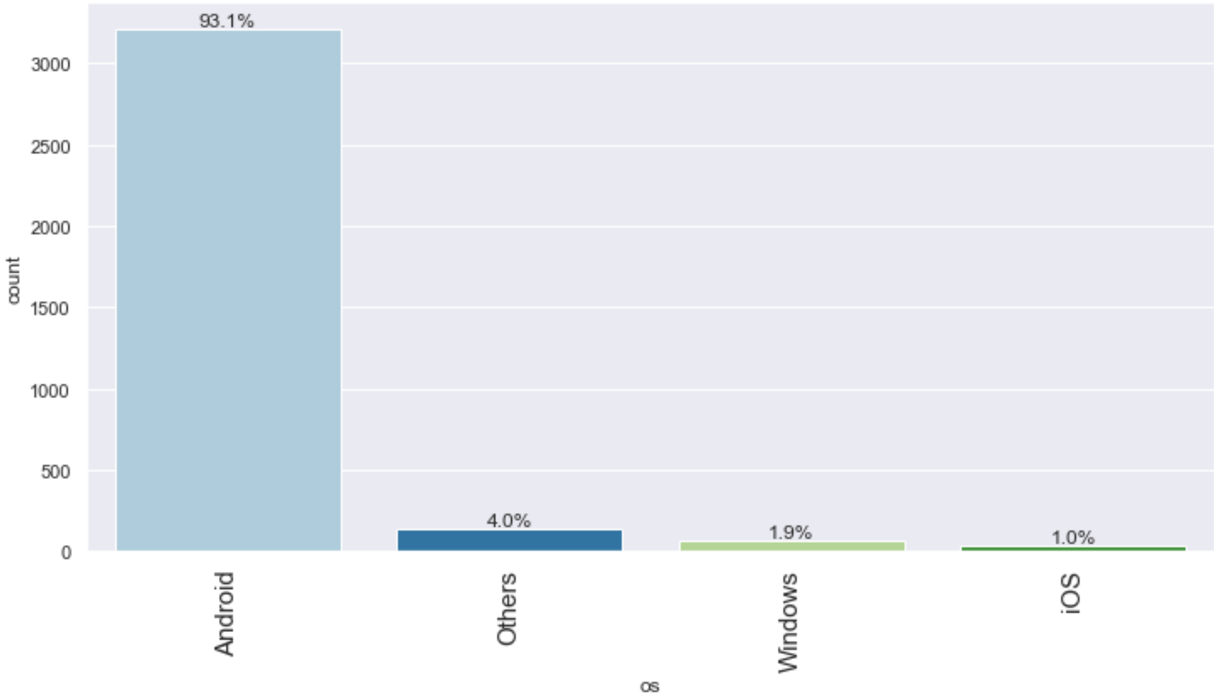
`brand_name`

In [23]:
```
labeled_barplot(df, "brand_name", perc=True, n=10)
```



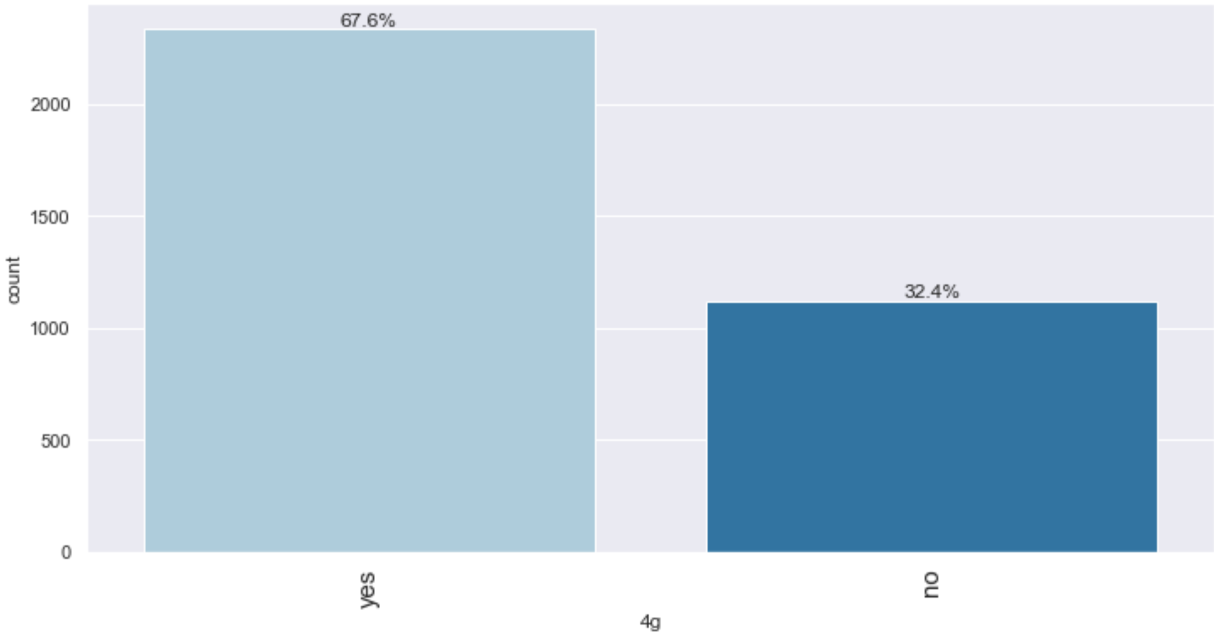
os

In [24]:
```
labeled_barplot(df, "os", perc=True, n=10) ## Complete the code to create
```



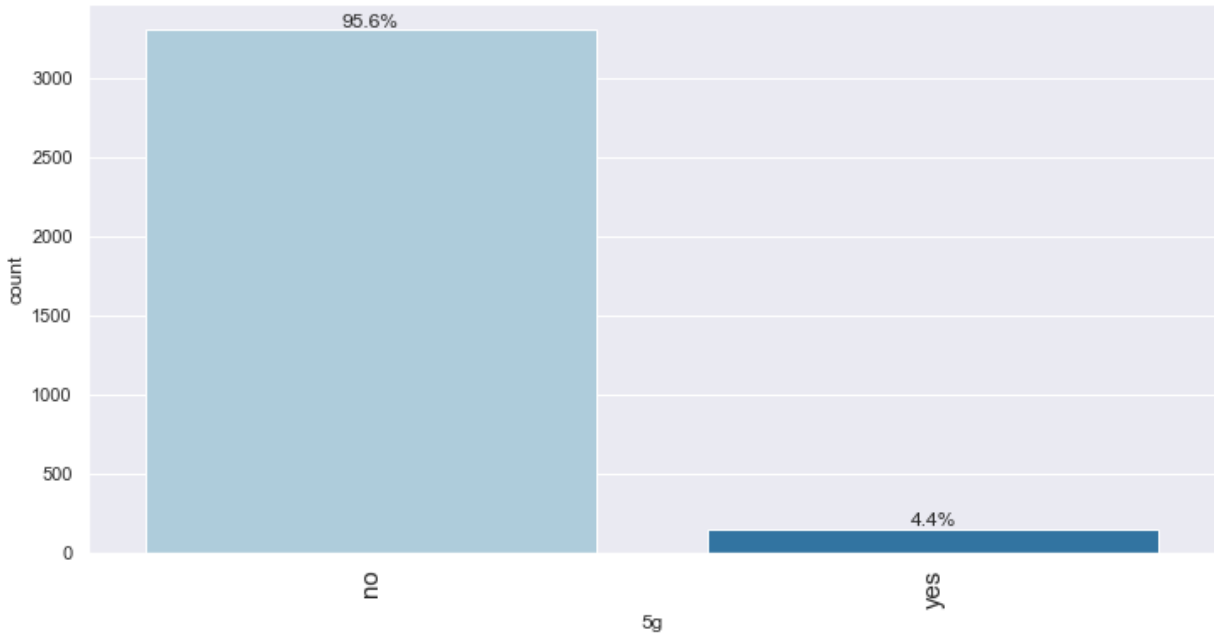
4g

In [25]:
```
labeled_barplot(df, "4g", perc=True, n=10) ## Complete the code to create
```



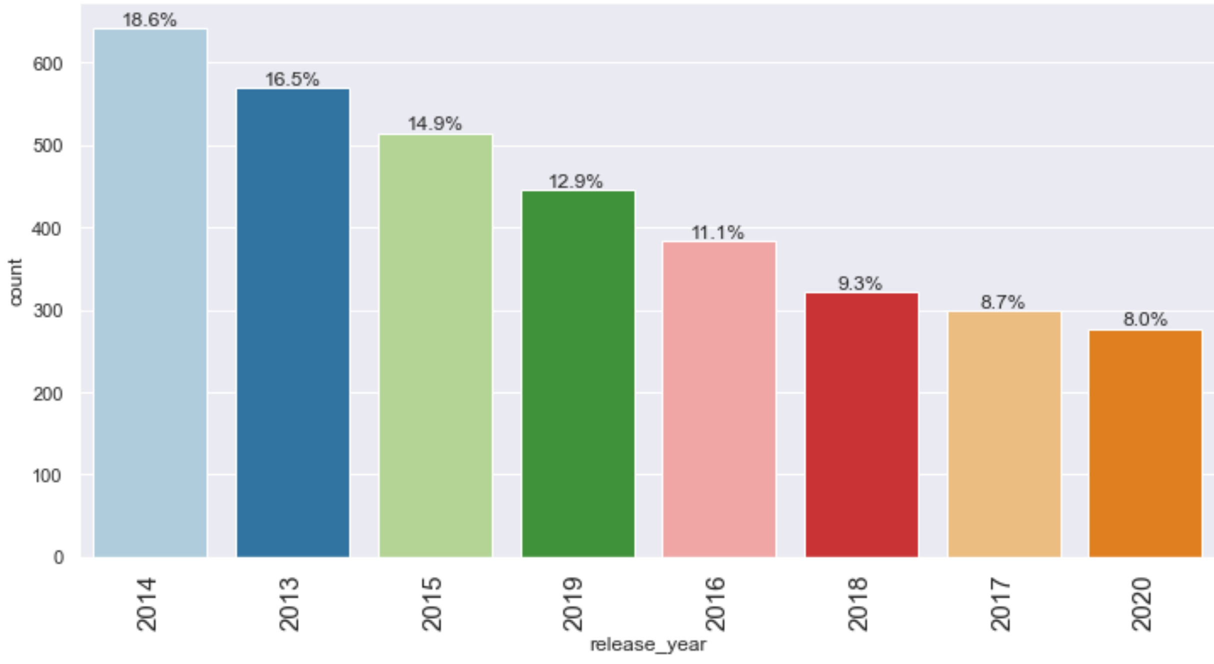
5g

In [26]:
```
labeled_barplot(df, "5g", perc=True, n=10) ## Complete the code to create
```



`release_year`

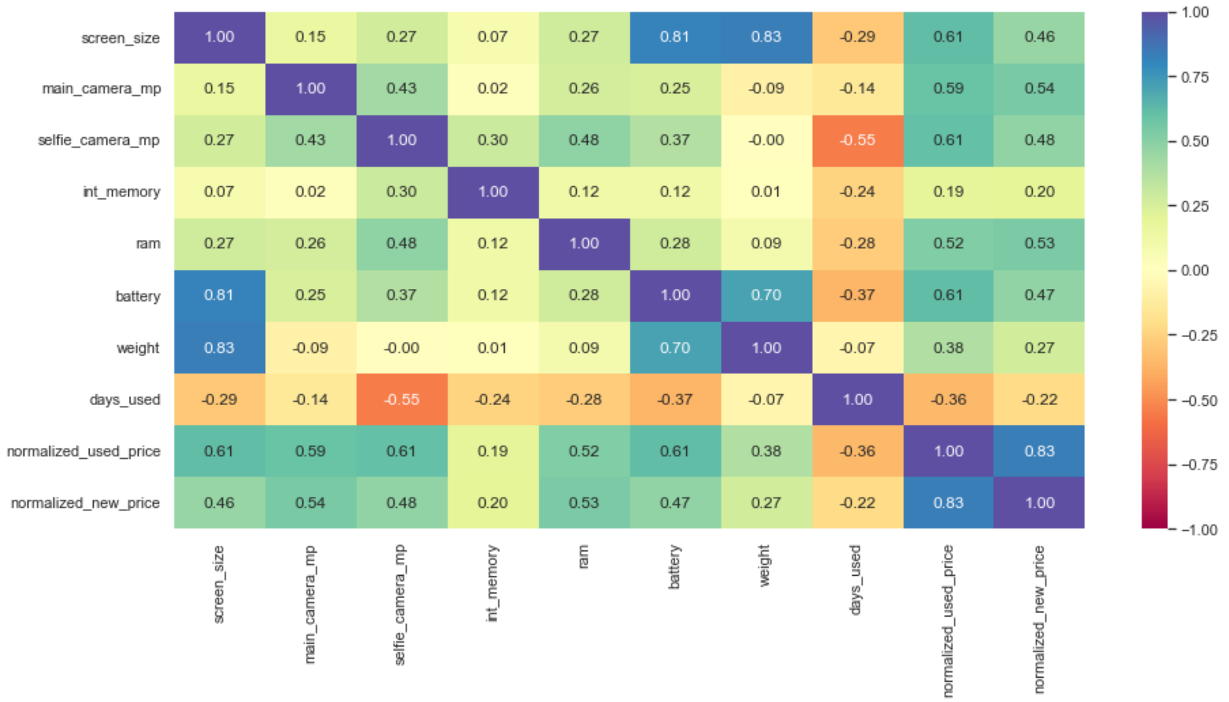
In [27]:
```
labeled_barplot(df, "release_year", perc=True, n=10) ## Complete the code
```



Bivariate Analysis

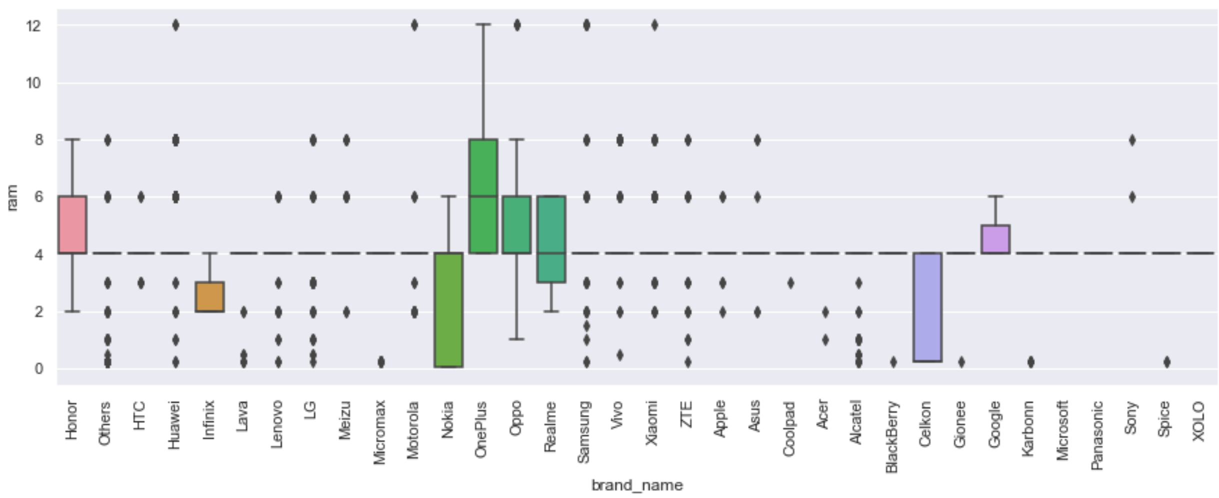
Correlation Check

In [28]:
```
cols_list = df.select_dtypes(include=np.number).columns.tolist()
# dropping release_year as it is a temporal variable
cols_list.remove("release_year")

plt.figure(figsize=(15, 7))
sns.heatmap(
    df[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="S
)
plt.show()
```



The amount of RAM is important for the smooth functioning of a device. Let's see how the amount of RAM varies across brands.

In [29]:
```
plt.figure(figsize=(15, 5))
sns.boxplot(data=df, x="brand_name", y="ram")
plt.xticks(rotation=90)
plt.show()
```

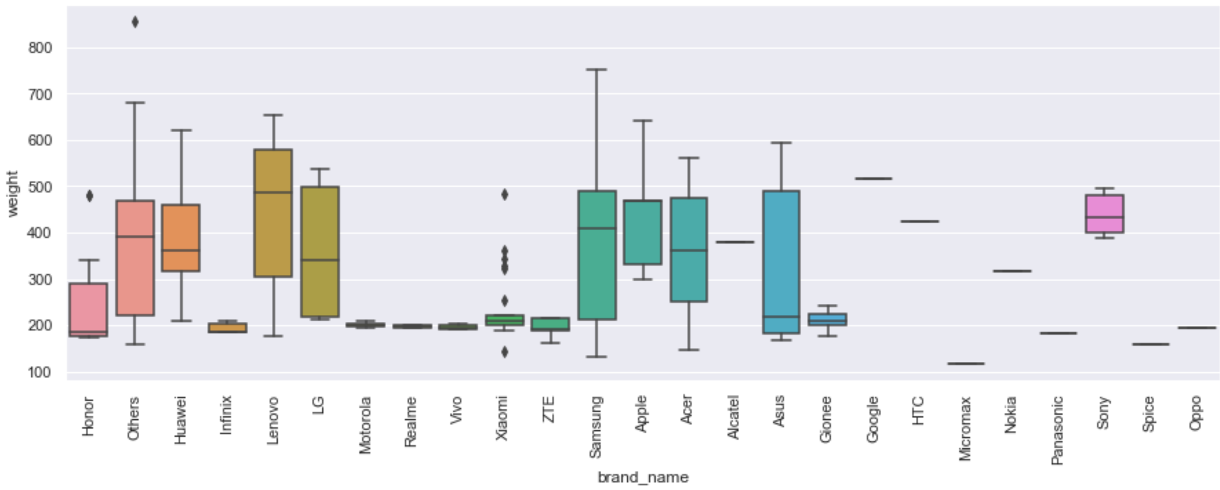


People who travel frequently require devices with large batteries to run through the day. But large battery often increases weight, making it feel uncomfortable in the hands. Let's create a new dataframe of only those devices which offer a large battery and analyze.

In [30]:
```
df_large_battery = df[df.battery > 4500]
df_large_battery.shape
```

Out[30]:
```
(341, 15)
```

In [32]:
```
plt.figure(figsize=(15, 5))
sns.boxplot(data=df_large_battery, x='brand_name', y='weight') ## Complet
plt.xticks(rotation=90)
plt.show()
```

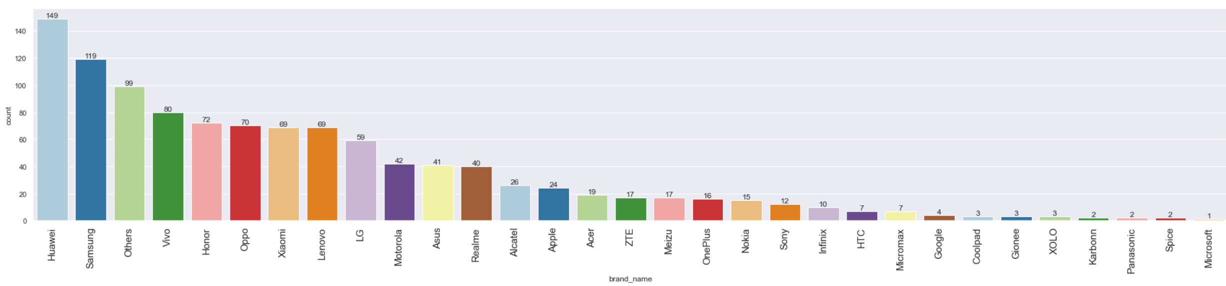


People who buy phones and tablets primarily for entertainment purposes prefer a large screen as they offer a better viewing experience. Let's create a new dataframe of only those devices which are suitable for such people and analyze.

In [33]:
```
df_large_screen = df[df.screen_size > 6 * 2.54]
df_large_screen.shape
```

Out[33]:
```
(1099, 15)
```

In [34]:
```
labeled_barplot(df_large_screen, 'brand_name') ## Complete the code to cr
```

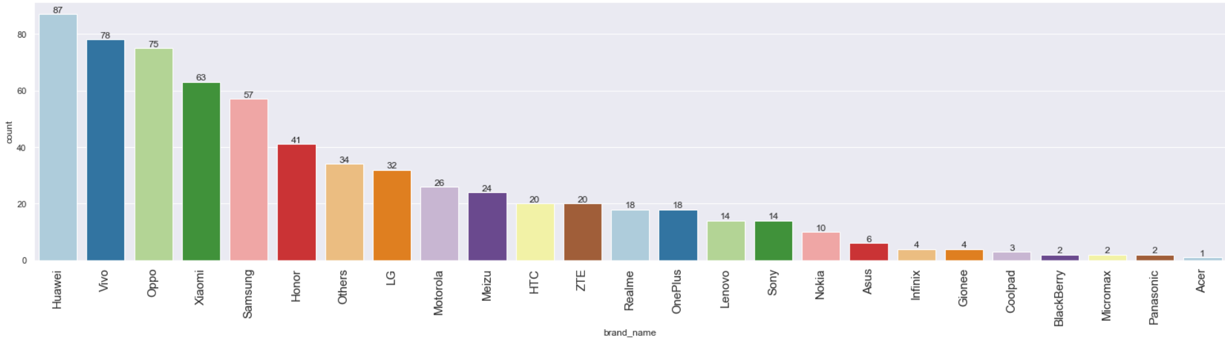


Everyone likes a good camera to capture their favorite moments with loved ones. Some customers specifically look for good front cameras to click cool selfies. Let's create a new dataframe of only those devices which are suitable for this customer segment and analyze.

In [35]:
```
df_selfie_camera = df[df.selfie_camera_mp > 8]
df_selfie_camera.shape
```

Out[35]:
```
(655, 15)
```

In [36]:
```
labeled_barplot(df_selfie_camera, 'brand_name') ## Complete the code to c
```



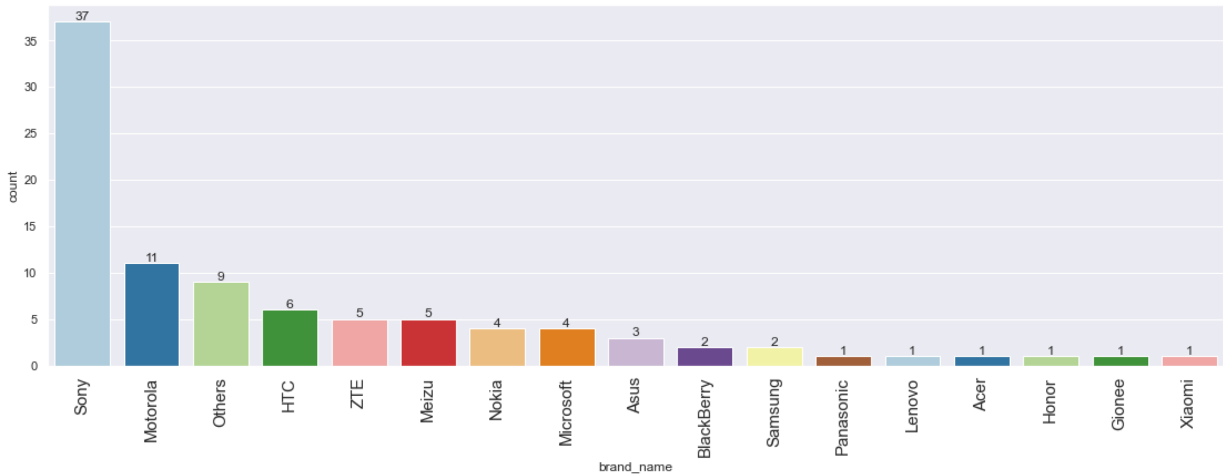
Let's do a similar analysis for rear cameras.

- Rear cameras generally have a better resolution than front cameras, so we set the threshold higher for them at 16MP.

In [37]:
```
df_main_camera = df[df.main_camera_mp > 16]
df_main_camera.shape
```

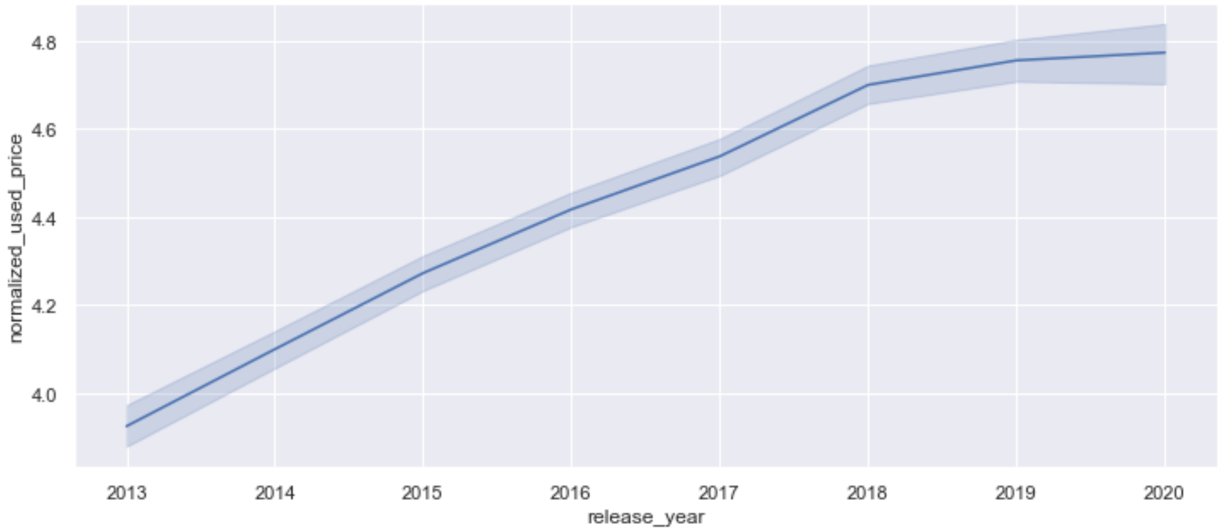
Out[37]:
```
(94, 15)
```

In [38]:
```
labeled_barplot(df_main_camera, 'brand_name') ## Complete the code to cre
```



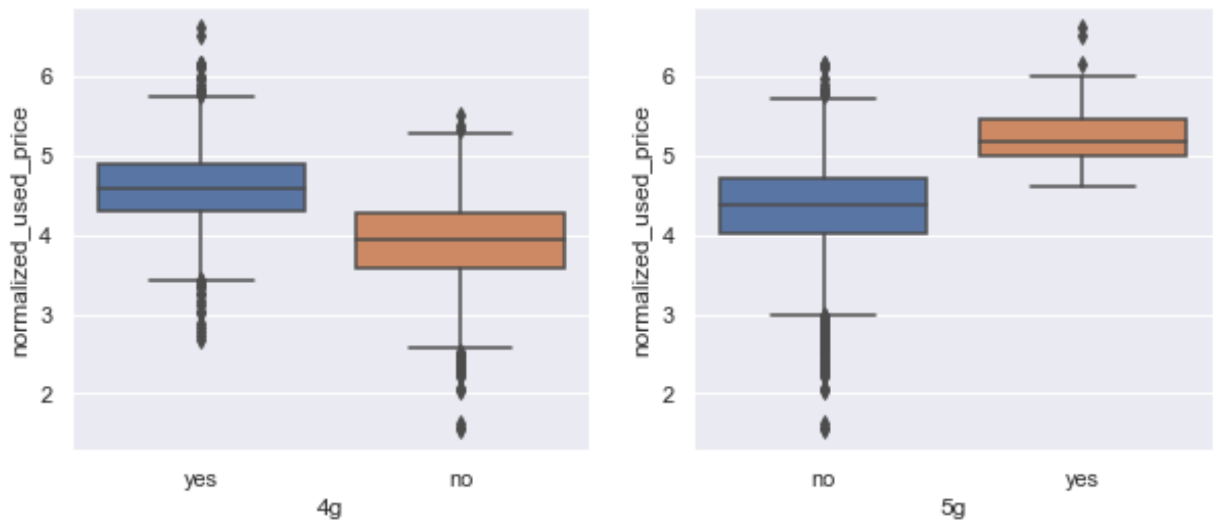
Let's see how the price of used devices varies across the years.

In [40]:
```
plt.figure(figsize=(12, 5))
sns.lineplot(data=df, x="release_year", y='normalized_used_price') ## Com
plt.show()
```



Let's check how the prices vary for used phones and tablets offering 4G and 5G networks.

In [41]:
```
plt.figure(figsize=(10, 4))

plt.subplot(121)
sns.boxplot(data=df, x="4g", y="normalized_used_price")

plt.subplot(122)
sns.boxplot(data=df, x="5g", y="normalized_used_price")

plt.show()
```



Data Preprocessing

Missing Value Imputation

- We will impute the missing values in the data by the column medians grouped by `release_year` and `brand_name`.

In [42]:
```
# let's create a copy of the data
df1 = df.copy()
```

In [43]:
```
# checking for missing values
df1.isnull().sum().sort_values(ascending= False) ## Complete the code to
```

Out[43]:
```
main_camera_mp          179
weight                    7
battery                   6
int_memory                4
ram                       4
selfie_camera_mp          2
brand_name                0
os                        0
screen_size               0
4g                        0
5g                        0
release_year              0
days_used                 0
normalized_used_price     0
normalized_new_price      0
dtype: int64
```

In [44]:
```
cols_impute = [
    "main_camera_mp",
    "selfie_camera_mp",
    "int_memory",
    "ram",
    "battery",
    "weight",
]

for col in cols_impute:
    df1[col] = df1[col].fillna(
        value=df1.groupby(['release_year','brand_name'])[col].transform("
    )   ## Complete the code to impute missing values in cols_impute with

# checking for missing values
df1.isnull().sum().sort_values(ascending=False) ## Complete the code to c
```

Out[44]:

```
main_camera_mp           179
weight                     7
battery                    6
selfie_camera_mp           2
brand_name                 0
os                         0
screen_size                0
4g                         0
5g                         0
int_memory                 0
ram                        0
release_year               0
days_used                  0
normalized_used_price      0
normalized_new_price       0
dtype: int64
```

- We will impute the remaining missing values in the data by the column medians grouped by `brand_name`.

In [45]:

```
cols_impute = [
    "main_camera_mp",
    "selfie_camera_mp",
    "battery",
    "weight",
]

for col in cols_impute:
    df1[col] = df1[col].fillna(
        value=df1.groupby(['brand_name'])[col].transform("median")
    ) ## Complete the code to impute the missing values in cols_impute wi

# checking for missing values
df1.isnull().sum().sort_values(ascending= False) ## Complete the code to
```

Out[45]:

```
main_camera_mp           10
brand_name                0
os                        0
screen_size               0
4g                        0
5g                        0
selfie_camera_mp          0
int_memory                0
ram                       0
battery                   0
weight                    0
release_year              0
days_used                 0
normalized_used_price     0
normalized_new_price      0
dtype: int64
```

- We will fill the remaining missing values in the `main_camera_mp` column by the column median.

In [46]:
```
df1["main_camera_mp"] = df1["main_camera_mp"].fillna(df1["main_camera_mp"

# checking for missing values
df1.isnull().sum().sort_values(ascending= False) ## Complete the code to
```

Out[46]:
```
brand_name               0
os                       0
screen_size              0
4g                       0
5g                       0
main_camera_mp           0
selfie_camera_mp         0
int_memory               0
ram                      0
battery                  0
weight                   0
release_year             0
days_used                0
normalized_used_price    0
normalized_new_price     0
dtype: int64
```

Feature Engineering

- Let's create a new column `years_since_release` from the `release_year` column.
- We will consider the year of data collection, 2021, as the baseline.
- We will drop the `release_year` column.

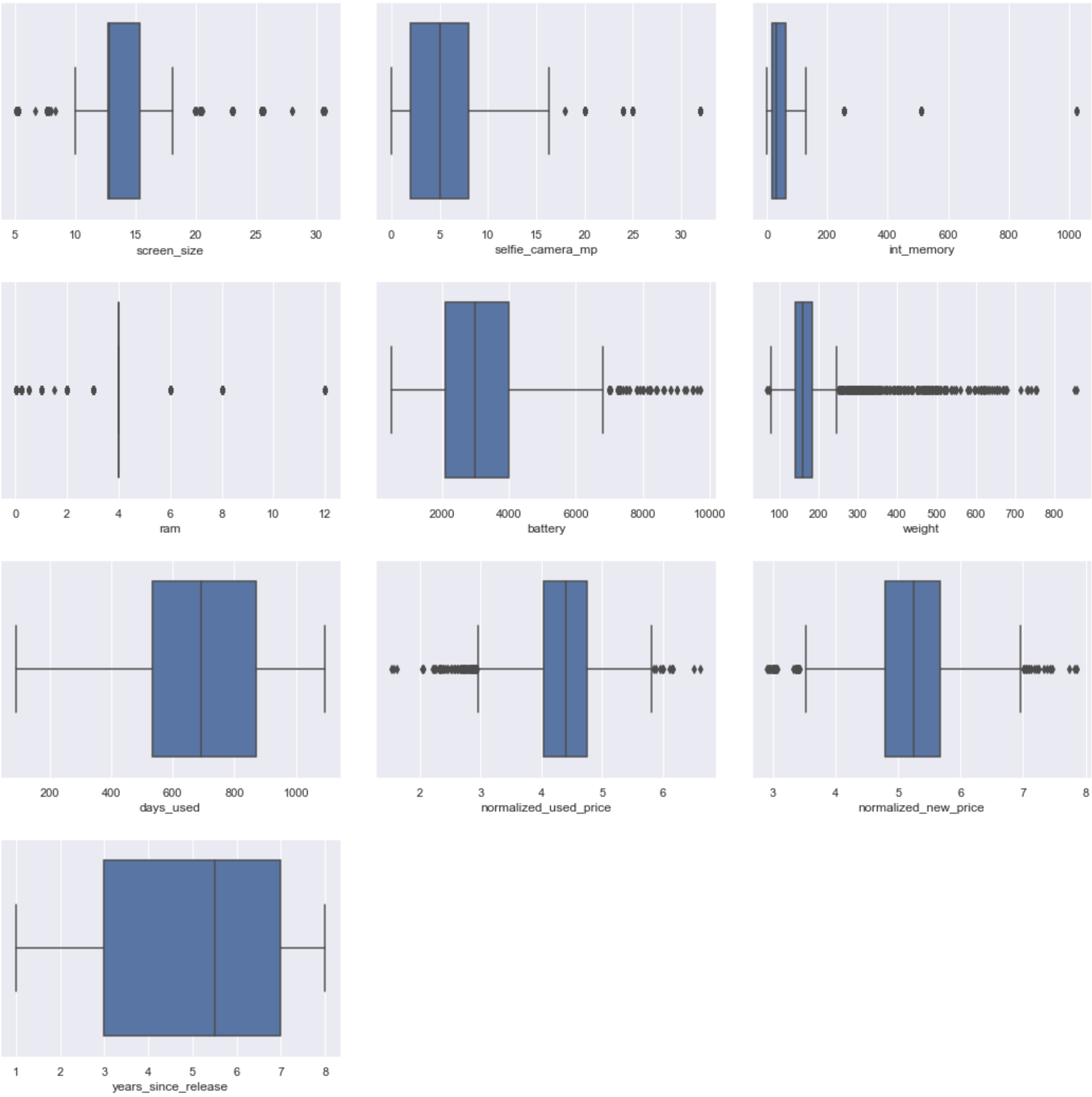
In [47]:
```
df1["years_since_release"] = 2021 - df1["release_year"]
df1.drop("release_year", axis=1, inplace=True)
df1["years_since_release"].describe()
```

Out[47]:
```
count    3454.000000
mean        5.034742
std         2.298455
min         1.000000
25%         3.000000
50%         5.500000
75%         7.000000
max         8.000000
Name: years_since_release, dtype: float64
```

Outlier Check

- Let's check for outliers in the data.

In [48]:
```
# outlier detection using boxplot
num_cols = df1.select_dtypes(include=np.number).columns.tolist()

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

for i, variable in enumerate(num_cols):
    plt.subplot(4, 3, i + 1)
    sns.boxplot(data=df1, x=variable)
    plt.tight_layout(pad=2)

plt.show()
```



Data Preparation for modeling

- We want to predict the normalized price of used devices
- Before we proceed to build a model, we'll have to encode categorical features
- We'll split the data into train and test to be able to evaluate the model that we build on the train data
- We will build a Linear Regression model using the train data and then check it's performance

In [49]:
```
## Complete the code to define the dependent and independent variables
X = df1.drop(['normalized_used_price'],axis=1)
y = df1['normalized_used_price']

print(X.head())
print()
print(y.head())
```

```
  brand_name       os  screen_size   4g   5g main_camera_mp  selfie_camer
a_mp  \
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

   int_memory  ram  battery  weight  days_used  normalized_new_price  \
0        64.0  3.0   3020.0   146.0        127              4.715100
1       128.0  8.0   4300.0   213.0        325              5.519018
2       128.0  8.0   4200.0   213.0        162              5.884631
3        64.0  6.0   7250.0   480.0        345              5.630961
4        64.0  3.0   5000.0   185.0        293              4.947837

   years_since_release
0                    1
1                    1
2                    1
3                    1
4                    1

0    4.307572
1    5.162097
2    5.111084
3    5.135387
4    4.389995
Name: normalized_used_price, dtype: float64
```

In [50]:
```
# let's add the intercept to data
X = sm.add_constant(X)
```

In [51]:
```
# creating dummy variables
X = pd.get_dummies(
    X,
    columns=X.select_dtypes(include=["object", "category"]).columns.tolis
    drop_first=True,
)  ## Complete the code to create dummies for independent features

X.head()
```

Out[51]:

	const	screen_size	selfie_camera_mp	int_memory	ram	battery	weight	days_used
0	1.0	14.50	5.0	64.0	3.0	3020.0	146.0	127
1	1.0	17.30	16.0	128.0	8.0	4300.0	213.0	325
2	1.0	16.69	8.0	128.0	8.0	4200.0	213.0	162
3	1.0	25.50	8.0	64.0	6.0	7250.0	480.0	345
4	1.0	15.32	8.0	64.0	3.0	5000.0	185.0	293

5 rows × 89 columns

In [52]:
```
# splitting the data in 70:30 ratio for train to test data

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

In [53]:
```
print("Number of rows in train data =", x_train.shape[0])
print("Number of rows in test data =", x_test.shape[0])
```

```
Number of rows in train data = 2417
Number of rows in test data = 1037
```

Model Building - Linear Regression

In [54]:
```
olsmodel1 = sm.OLS(y_train,x_train).fit() ## Complete the code to fit OLS
print(olsmodel1.summary())
```

```
                            OLS Regression Results
==========================================================================
========
Dep. Variable:     normalized_used_price   R-squared:
0.850
Model:                               OLS   Adj. R-squared:
0.845
Method:                    Least Squares   F-statistic:
159.2
Date:                   Wed, 15 Feb 2023   Prob (F-statistic):
0.00
Time:                           12:42:22   Log-Likelihood:
163.90
No. Observations:                   2417   AIC:
-159.8
Df Residuals:                       2333   BIC:
326.6
```

```
Df Model:                          83
Covariance Type:            nonrobust
==========================================================================
==========================================================================
==========================================================================
==========================================================================
===================================================

coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
---------------------------------------------------
const
1.6815      0.080     20.959      0.000       1.524       1.839
screen_size
0.0238      0.003      6.865      0.000       0.017       0.031
selfie_camera_mp
0.0137      0.001     12.146      0.000       0.012       0.016
int_memory
0.0002   7.09e-05      2.259      0.024    2.11e-05       0.000
ram
0.0223      0.005      4.274      0.000       0.012       0.032
battery
-1.753e-05   7.39e-06     -2.373      0.018    -3.2e-05   -3.04e-06
weight
0.0011      0.000      7.982      0.000       0.001       0.001
days_used
2.496e-05   3.11e-05      0.803      0.422    -3.6e-05     8.6e-05
normalized_new_price
0.4146      0.013     32.372      0.000       0.390       0.440
years_since_release
-0.0213      0.005     -4.560      0.000      -0.031      -0.012
brand_name_Alcatel
0.0093      0.047      0.196      0.844      -0.084       0.102
brand_name_Apple
-0.1129      0.147     -0.766      0.444      -0.402       0.176
brand_name_Asus
0.0033      0.048      0.070      0.944      -0.090       0.097
brand_name_BlackBerry
-0.0768      0.072     -1.067      0.286      -0.218       0.064
brand_name_Celkon
-0.0499      0.067     -0.747      0.455      -0.181       0.081
brand_name_Coolpad
0.0089      0.073      0.123      0.902      -0.133       0.151
brand_name_Gionee
0.0425      0.058      0.738      0.461      -0.070       0.155
brand_name_Google
0.0562      0.157      0.359      0.720      -0.251       0.364
brand_name_HTC
-0.0112      0.049     -0.230      0.818      -0.107       0.084
brand_name_Honor
0.0201      0.049      0.407      0.684      -0.076       0.117
brand_name_Huawei
-0.0225      0.045     -0.502      0.616      -0.110       0.065
brand_name_Infinix
```

```
0.0192      0.046      0.413      0.680      -0.072       0.110
brand_name_Karbonn
0.0772      0.067      1.155      0.248      -0.054       0.208
brand_name_LG
-0.0206      0.045     -0.455      0.649      -0.110       0.068
brand_name_Lava
0.0197      0.062      0.317      0.752      -0.102       0.142
brand_name_Lenovo
0.0340      0.045      0.751      0.453      -0.055       0.123
brand_name_Meizu
-0.0198      0.056     -0.353      0.724      -0.130       0.090
brand_name_Micromax
-0.0254      0.048     -0.533      0.594      -0.119       0.068
brand_name_Microsoft
0.0642      0.089      0.720      0.472      -0.111       0.239
brand_name_Motorola
-0.0037      0.050     -0.074      0.941      -0.102       0.094
brand_name_Nokia
0.0781      0.052      1.498      0.134      -0.024       0.180
brand_name_OnePlus
0.0680      0.077      0.881      0.378      -0.083       0.219
brand_name_Oppo
0.0051      0.048      0.107      0.915      -0.089       0.099
brand_name_Others
-0.0125      0.042     -0.297      0.767      -0.095       0.070
brand_name_Panasonic
0.0428      0.056      0.768      0.443      -0.067       0.152
brand_name_Realme
0.0156      0.062      0.253      0.800      -0.105       0.136
brand_name_Samsung
-0.0477      0.043     -1.098      0.272      -0.133       0.037
brand_name_Sony
-0.0476      0.055     -0.862      0.389      -0.156       0.061
brand_name_Spice
-0.0184      0.063     -0.291      0.771      -0.142       0.106
brand_name_Vivo
-0.0279      0.049     -0.575      0.565      -0.123       0.067
brand_name_XOLO
-7.906e-05      0.055     -0.001      0.999      -0.107       0.107
brand_name_Xiaomi
0.0650      0.048      1.342      0.180      -0.030       0.160
brand_name_ZTE
-0.0027      0.048     -0.057      0.955      -0.097       0.091
os_Others
0.0241      0.035      0.683      0.495      -0.045       0.093
os_Windows
-2.594e-05      0.048     -0.001      1.000      -0.093       0.093
os_iOS
0.0272      0.146      0.186      0.852      -0.259       0.313
4g_yes
0.0479      0.016      2.975      0.003       0.016       0.080
5g_yes
-0.0665      0.032     -2.065      0.039      -0.130      -0.003
main_camera_mp_8.0
-0.1088      0.015     -7.331      0.000      -0.138      -0.080
main_camera_mp_5.0
-0.1803      0.019     -9.565      0.000      -0.217      -0.143
```

```
main_camera_mp_10.5
0.0275      0.055      0.501      0.616      -0.080       0.135
main_camera_mp_3.15
-0.2440      0.032     -7.688      0.000      -0.306      -0.182
main_camera_mp_<bound method NDFrame._add_numeric_operations.<locals>.med
ian of 0       13.0
1       13.0
2       13.0
3       13.0
4       13.0
        ...
3449    13.0
3450    13.0
3451    13.0
3452    13.0
3453    13.0
Name: main_camera_mp, Length: 3454, dtype: float64>     0.0192      0.046
0.413      0.680      -0.072       0.110
main_camera_mp_2.0
-0.2751      0.029     -9.569      0.000      -0.331      -0.219
main_camera_mp_16.0
0.1076      0.025      4.345      0.000       0.059       0.156
main_camera_mp_0.3
-0.4714      0.044    -10.704      0.000      -0.558      -0.385
main_camera_mp_12.0
0.0076      0.024      0.319      0.750      -0.039       0.055
main_camera_mp_14.5
-0.0145      0.078     -0.187      0.852      -0.167       0.138
main_camera_mp_48.0
0.2757      0.118      2.344      0.019       0.045       0.506
main_camera_mp_3.0
-0.1121      0.136     -0.824      0.410      -0.379       0.155
main_camera_mp_21.0
0.0706      0.069      1.028      0.304      -0.064       0.205
main_camera_mp_1.3
-0.4807      0.066     -7.237      0.000      -0.611      -0.350
main_camera_mp_13.1
0.1410      0.165      0.854      0.393      -0.183       0.465
main_camera_mp_24.0
0.0398      0.134      0.296      0.767      -0.224       0.303
main_camera_mp_0.08
-0.4712      0.234     -2.010      0.045      -0.931      -0.011
main_camera_mp_20.7
0.1023      0.083      1.237      0.216      -0.060       0.264
main_camera_mp_23.0
0.2705      0.073      3.720      0.000       0.128       0.413
main_camera_mp_1.0
-1.589e-16   9.33e-17     -1.703      0.089   -3.42e-16     2.4e-17
main_camera_mp_18.0
0.0682      0.238      0.287      0.774      -0.398       0.535
main_camera_mp_12.2
-0.1628      0.175     -0.928      0.354      -0.507       0.181
main_camera_mp_12.3
0.0540      0.108      0.500      0.617      -0.158       0.266
main_camera_mp_20.0
0.1257      0.096      1.303      0.193      -0.064       0.315
main_camera_mp_20.2
```

```
0.0289      0.232      0.125      0.901      -0.426       0.484
main_camera_mp_4.0
-0.2934      0.098     -2.981      0.003      -0.486      -0.100
main_camera_mp_12.5
0.1250      0.117      1.067      0.286      -0.105       0.355
main_camera_mp_10.0
-0.2221      0.119     -1.868      0.062      -0.455       0.011
main_camera_mp_6.5
-0.0930      0.164     -0.566      0.572      -0.415       0.229
main_camera_mp_6.7
-0.2939      0.128     -2.303      0.021      -0.544      -0.044
main_camera_mp_41.0
-1.544e-17   2.62e-17     -0.589      0.556   -6.68e-17    3.59e-17
main_camera_mp_20.1
2.805e-17   4.81e-17      0.583      0.560   -6.63e-17    1.22e-16
main_camera_mp_12.6
-2.974e-17   3.99e-17     -0.745      0.456   -1.08e-16    4.85e-17
main_camera_mp_16.3
0.3742      0.233      1.608      0.108      -0.082       0.830
main_camera_mp_22.6
-0.1479      0.232     -0.639      0.523      -0.602       0.306
main_camera_mp_19.0
-0.0679      0.102     -0.666      0.506      -0.268       0.132
main_camera_mp_21.5
0.1234      0.234      0.527      0.598      -0.335       0.582
main_camera_mp_21.2
0.2065      0.168      1.233      0.218      -0.122       0.535
main_camera_mp_8.1
-0.1930      0.123     -1.572      0.116      -0.434       0.048
main_camera_mp_1.2
-0.1055      0.235     -0.449      0.654      -0.567       0.356
main_camera_mp_22.5
0.1759      0.232      0.759      0.448      -0.279       0.630
========================================================================
=====
Omnibus:                      205.856   Durbin-Watson:
1.906
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            38
9.471
Skew:                          -0.578   Prob(JB):                    2.6
8e-85
Kurtosis:                       4.591   Cond. No.                    4.2
1e+19
========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The smallest eigenvalue is 1.63e-29. This might indicate that there a
re
strong multicollinearity problems or that the design matrix is singular.
```

In [ ]:
```
The value for adj. R-squared is 0.84, which is good.
The y-intercept is equal to the value of the const coeffiecient which is
The coefficients of the different predictor variables is well listed in t
normalized_new_price is equal to 0.420.
```

Model Performance Check

Let's check the performance of the model using different metrics.

- We will be using metric functions defined in sklearn for RMSE, MAE, and R^2.
- We will define a function to calculate MAPE and adjusted R^2.
- We will create a function which will print out all the above metrics in one go.

In [55]:
```
# function to compute adjusted R-squared
def adj_r2_score(predictors, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = predictors.shape[0]
    k = predictors.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))


# function to compute MAPE
def mape_score(targets, predictions):
    return np.mean(np.abs(targets - predictions) / targets) * 100


# function to compute different metrics to check performance of a regress
def model_performance_regression(model, predictors, target):
    """
    Function to compute different metrics to check regression model perfo

    model: regressor
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    r2 = r2_score(target, pred)  # to compute R-squared
    adjr2 = adj_r2_score(predictors, target, pred)  # to compute adjusted
    rmse = np.sqrt(mean_squared_error(target, pred))  # to compute RMSE
    mae = mean_absolute_error(target, pred)  # to compute MAE
    mape = mape_score(target, pred)  # to compute MAPE

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "RMSE": rmse,
            "MAE": mae,
            "R-squared": r2,
            "Adj. R-squared": adjr2,
            "MAPE": mape,
        },
        index=[0],
    )

    return df_perf
```

In [56]:
```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel1_train_perf = model_performance_regression(olsmodel1, x_train, y
olsmodel1_train_perf
```

```
Training Performance
```

Out[56]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.226107	0.17736	0.849942	0.844202	4.247918

In [57]:

```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmodel1_test_perf = model_performance_regression(olsmodel1,x_test,y_tes
olsmodel1_test_perf
```

```
Test Performance
```

Out[57]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.239404	0.185272	0.841094	0.82616	4.495925

Checking Linear Regression Assumptions

We will be checking the following Linear Regression assumptions:

1. **No Multicollinearity**
2. **Linearity of variables**
3. **Independence of error terms**
4. **Normality of error terms**
5. **No Heteroscedasticity**

TEST FOR MULTICOLLINEARITY

- We will test for multicollinearity using VIF.
- **General Rule of thumb**:
  - If VIF is 1 then there is no correlation between the kth predictor and the remaining predictor variables.
  - If VIF exceeds 5 or is close to exceeding 5, we say there is moderate multicollinearity.
  - If VIF is 10 or exceeding 10, it shows signs of high multicollinearity.

Let's define a function to check VIF.

In [58]:
```
#Let's define a function to check VIF.

def checking_vif(predictors):
    vif = pd.DataFrame()
    vif["feature"] = predictors.columns

    # calculating VIF for each feature
    vif["VIF"] = [
        variance_inflation_factor(predictors.values, i)
        for i in range(len(predictors.columns))
    ]
    return vif
```

In [61]:
```
checking_vif(x_train).ascending  ## Complete the code to check VIF on tra
```

```
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/statsmodels/sta
ts/outliers_influence.py:195: RuntimeWarning: divide by zero encountered
in double_scalars
  vif = 1. / (1. - r_squared_i)
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/statsmodels/reg
ression/linear_model.py:1736: RuntimeWarning: invalid value encountered i
n double_scalars
  return 1 - self.ssr/self.centered_tss
```

Out[61]:

	feature	VIF
0	const	293.713519
1	screen_size	8.072753
2	selfie_camera_mp	2.897750
3	int_memory	1.434004
4	ram	2.375248
5	battery	4.285870
6	weight	6.972076
7	days_used	2.751173
8	normalized_new_price	3.464667
9	years_since_release	5.260309
10	brand_name_Alcatel	3.436328
11	brand_name_Apple	13.339101
12	brand_name_Asus	3.377646
13	brand_name_BlackBerry	1.748348
14	brand_name_Celkon	1.835217

Removing Multicollinearity (if needed)

To remove multicollinearity

1. Drop every column one by one that has a VIF score greater than 5.
2. Look at the adjusted R-squared and RMSE of all these models.
3. Drop the variable that makes the least change in adjusted R-squared.
4. Check the VIF scores again.
5. Continue till you get all VIF scores under 5.

Let's define a function that will help us do this.

In [62]:
```
def treating_multicollinearity(predictors, target, high_vif_columns):
    """
    Checking the effect of dropping the columns showing high multicolline
    on model performance (adj. R-squared and RMSE)

    predictors: independent variables
    target: dependent variable
    high_vif_columns: columns having high VIF
    """
    # empty lists to store adj. R-squared and RMSE values
    adj_r2 = []
    rmse = []

    # build ols models by dropping one of the high VIF columns at a time
    # store the adjusted R-squared and RMSE in the lists defined previous
    for cols in high_vif_columns:
        # defining the new train set
        train = predictors.loc[:, ~predictors.columns.str.startswith(cols

        # create the model
        olsmodel = sm.OLS(target, train).fit()

        # adding adj. R-squared and RMSE to the lists
        adj_r2.append(olsmodel.rsquared_adj)
        rmse.append(np.sqrt(olsmodel.mse_resid))

    # creating a dataframe for the results
    temp = pd.DataFrame(
        {
            "col": high_vif_columns,
            "Adj. R-squared after_dropping col": adj_r2,
            "RMSE after dropping col": rmse,
        }
    ).sort_values(by="Adj. R-squared after_dropping col", ascending=False
    temp.reset_index(drop=True, inplace=True)

    return temp
```

In [63]:
```
col_list = ['screen_size','weight'] ## Complete the code to specify the c

res = treating_multicollinearity(x_train, y_train, col_list) ## Complete
res
```

Out[63]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	screen_size	0.841532	0.232404
1	weight	0.840428	0.233212

In [65]:

```
col_to_drop = 'weight' ## Complete the code to specify the column to drop
x_train2 = x_train.loc[:, ~x_train.columns.str.startswith(col_to_drop)] #
x_test2 = x_test.loc[:, ~x_test.columns.str.startswith(col_to_drop)] ## C

# Check VIF now
vif = checking_vif(x_train2)
print("VIF after dropping ", col_to_drop)
vif.head(20)
```

```
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/statsmodels/sta
ts/outliers_influence.py:195: RuntimeWarning: divide by zero encountered
in double_scalars
  vif = 1. / (1. - r_squared_i)
VIF after dropping  weight
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/statsmodels/reg
ression/linear_model.py:1736: RuntimeWarning: invalid value encountered i
n double_scalars
  return 1 - self.ssr/self.centered_tss
```

Out[65]:

	feature	VIF
0	const	256.414019
1	screen_size	3.791503
2	selfie_camera_mp	2.866614
3	int_memory	1.431794
4	ram	2.374720
5	battery	3.810056
6	days_used	2.738370
7	normalized_new_price	3.449802
8	years_since_release	5.096289
9	brand_name_Alcatel	3.436116
10	brand_name_Apple	13.337119
11	brand_name_Asus	3.373382
12	brand_name_BlackBerry	1.747677
13	brand_name_Celkon	1.833790
14	brand_name_Coolpad	1.481306
15	brand_name_Gionee	1.978886
16	brand_name_Google	4.619000
17	brand_name_HTC	3.556834
18	brand_name_Honor	3.410058
19	brand_name_Huawei	6.198473

Dropping high p-value variables (if needed)

- We will drop the predictor variables having a p-value greater than 0.05 as they do not significantly impact the target variable.
- But sometimes p-values change after dropping a variable. So, we'll not drop all variables at once.
- Instead, we will do the following:
    - Build a model, check the p-values of the variables, and drop the column with the highest p-value.
    - Create a new model without the dropped feature, check the p-values of the variables, and drop the column with the highest p-value.
    - Repeat the above two steps till there are no columns with p-value > 0.05.

The above process can also be done manually by picking one variable at a time that has a high p-value, dropping it, and building a model again. But that might be a little tedious and using a loop will be more efficient.

In [66]:
```
# initial list of columns
predictors = x_train2.copy()  ## Complete the code to check for p-values
cols = predictors.columns.tolist()

# setting an initial max p-value
max_p_value = 1

while len(cols) > 0:
    # defining the train set
    x_train_aux = predictors[cols]

    # fitting the model
    model = sm.OLS(y_train, x_train_aux).fit()

    # getting the p-values and the maximum p-value
    p_values = model.pvalues
    max_p_value = max(p_values)

    # name of the variable with maximum p-value
    feature_with_p_max = p_values.idxmax()

    if max_p_value > 0.05:
        cols.remove(feature_with_p_max)
    else:
        break

selected_features = cols
print(selected_features)
```

```
['const', 'screen_size', 'selfie_camera_mp', 'ram', 'normalized_new_price
', 'years_since_release', 'brand_name_Nokia', 'brand_name_Samsung', 'bran
d_name_Xiaomi', '4g_yes', 'main_camera_mp_8.0', 'main_camera_mp_5.0', 'ma
in_camera_mp_3.15', 'main_camera_mp_2.0', 'main_camera_mp_16.0', 'main_ca
mera_mp_0.3', 'main_camera_mp_48.0', 'main_camera_mp_1.3', 'main_camera_m
p_23.0', 'main_camera_mp_4.0', 'main_camera_mp_10.0', 'main_camera_mp_6.7
', 'main_camera_mp_16.3', 'main_camera_mp_8.1']
```

In [67]:
```
x_train3 = x_train2[selected_features]  ## Complete the code to specify t
x_test3 = x_test2[selected_features]  ## Complete the code to specify the
```

In [68]:
```
olsmodel2 = sm.OLS(y_train,x_train3).fit() ## Complete the code fit OLS()
print(olsmodel2.summary())
```

```
                            OLS Regression Results
==========================================================================
========
Dep. Variable:     normalized_used_price   R-squared:
0.843
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
557.8
Date:                   Wed, 15 Feb 2023   Prob (F-statistic):
0.00
Time:                           12:45:14   Log-Likelihood:
107.69
No. Observations:                   2417   AIC:
-167.4
Df Residuals:                       2393   BIC:
-28.41
Df Model:                             23
Covariance Type:               nonrobust
==========================================================================
===============
                           coef    std err          t      P>|t|      [0.
025      0.975]
--------------------------------------------------------------------------
---------------
const                    1.5113      0.052     28.786      0.000       1.
408       1.614
screen_size              0.0439      0.002     28.724      0.000       0.
041       0.047
selfie_camera_mp         0.0131      0.001     12.330      0.000       0.
011       0.015
ram                      0.0156      0.004      3.480      0.001       0.
007       0.024
normalized_new_price     0.4195      0.011     38.112      0.000       0.
398       0.441
years_since_release     -0.0098      0.004     -2.780      0.005      -0.
017      -0.003
brand_name_Nokia         0.1008      0.031      3.235      0.001       0.
040       0.162
brand_name_Samsung      -0.0395      0.016     -2.402      0.016      -0.
072      -0.007
brand_name_Xiaomi        0.0723      0.026      2.826      0.005       0.
022       0.122
```

```
4g_yes                     0.0422      0.015      2.800      0.005       0.
013       0.072
main_camera_mp_8.0        -0.1024      0.014     -7.493      0.000      -0.
129      -0.076
main_camera_mp_5.0        -0.1589      0.017     -9.200      0.000      -0.
193      -0.125
main_camera_mp_3.15       -0.2054      0.030     -6.901      0.000      -0.
264      -0.147
main_camera_mp_2.0        -0.2168      0.026     -8.342      0.000      -0.
268      -0.166
main_camera_mp_16.0        0.1019      0.024      4.197      0.000       0.
054       0.149
main_camera_mp_0.3        -0.3934      0.039    -10.037      0.000      -0.
470      -0.317
main_camera_mp_48.0        0.2836      0.117      2.427      0.015       0.
054       0.513
main_camera_mp_1.3        -0.3872      0.060     -6.431      0.000      -0.
505      -0.269
main_camera_mp_23.0        0.2564      0.068      3.789      0.000       0.
124       0.389
main_camera_mp_4.0        -0.3067      0.096     -3.204      0.001      -0.
494      -0.119
main_camera_mp_10.0       -0.2505      0.117     -2.133      0.033      -0.
481      -0.020
main_camera_mp_6.7        -0.2403      0.121     -1.980      0.048      -0.
478      -0.002
main_camera_mp_16.3        0.5478      0.234      2.344      0.019       0.
089       1.006
main_camera_mp_8.1        -0.2956      0.118     -2.510      0.012      -0.
527      -0.065
=========================================================================
=====
Omnibus:                      193.234   Durbin-Watson:
1.911
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              36
3.597
Skew:                          -0.550   Prob(JB):                     1.1
1e-79
Kurtosis:                       4.549   Cond. No.
889.
=========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
```

In [69]:

```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel2_train_perf = model_performance_regression(olsmodel2, x_train3,
olsmodel2_train_perf
```

```
Training Performance
```

Out[69]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.231426	0.181502	0.842797	0.84122	4.344607

In [70]:

```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmodel2_test_perf = model_performance_regression(olsmodel2, x_test3, y_
olsmodel2_test_perf
```

```
Test Performance
```

Out[70]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.240476	0.186263	0.839667	0.835864	4.511603

Now we'll check the rest of the assumptions on *olsmod2*.

1. **Linearity of variables**
2. **Independence of error terms**
3. **Normality of error terms**
4. **No Heteroscedasticity**

TEST FOR LINEARITY AND INDEPENDENCE

- We will test for linearity and independence by making a plot of fitted values vs residuals and checking for patterns.
- If there is no pattern, then we say the model is linear and residuals are independent.
- Otherwise, the model is showing signs of non-linearity and residuals are not independent.

In [71]:

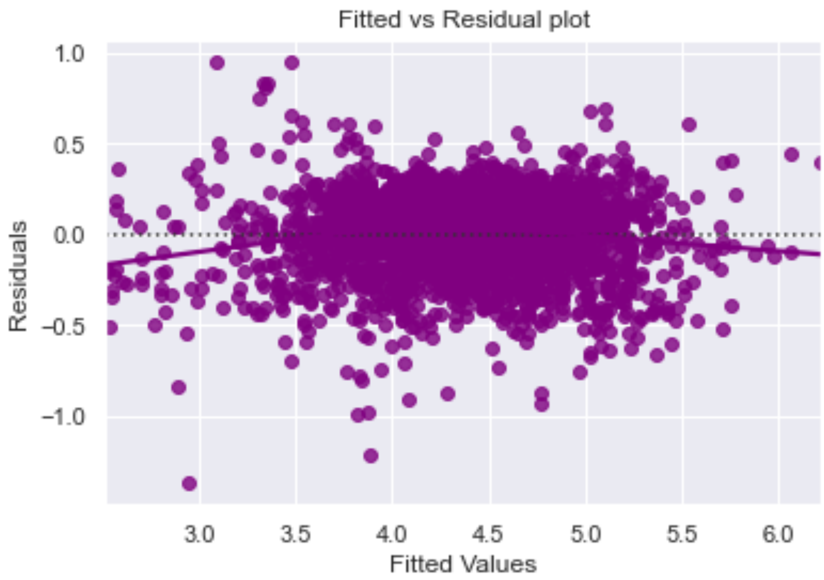
```
# let us create a dataframe with actual, fitted and residual values
df_pred = pd.DataFrame()

df_pred["Actual Values"] = y_train  # actual values
df_pred["Fitted Values"] = olsmodel2.fittedvalues  # predicted values
df_pred["Residuals"] = olsmodel2.resid  # residuals

df_pred.head()
```

Out[71]:

	Actual Values	Fitted Values	Residuals
3026	4.087488	3.870343	0.217145
1525	4.448399	4.585538	-0.137139
1128	4.315353	4.293276	0.022077
3003	4.282068	4.260293	0.021775
2907	4.456438	4.456145	0.000293

In [72]:

```
# let's plot the fitted values vs residuals

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

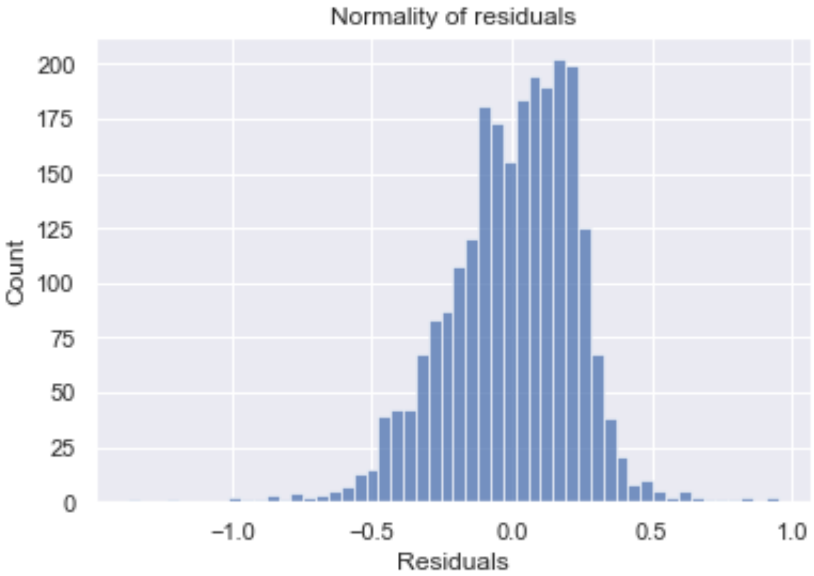


TEST FOR NORMALITY

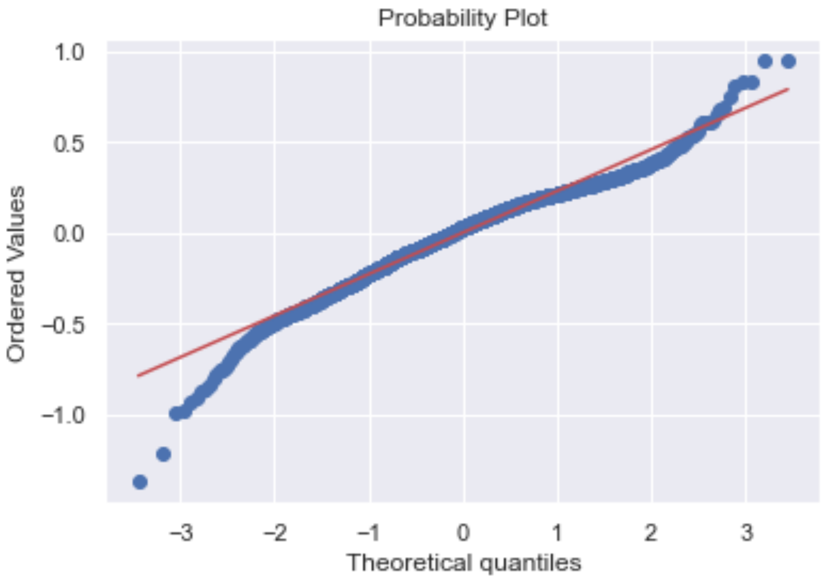
- We will test for normality by checking the distribution of residuals, by checking the Q-Q plot of residuals, and by using the Shapiro-Wilk test.
- If the residuals follow a normal distribution, they will make a straight line plot, otherwise not.
- If the p-value of the Shapiro-Wilk test is greater than 0.05, we can say the residuals are normally distributed.

In [73]:

```
sns.histplot(data=df_pred, x='Residuals') ## Complete the code to plot th
plt.title("Normality of residuals")
plt.show()
```



In [74]:
```
import pylab
import scipy.stats as stats

stats.probplot(df_pred['Residuals'], dist="norm", plot=pylab) ## Complete
plt.show()
```



In [75]:
```
stats.shapiro(df_pred['Residuals']) ## Complete the code to apply the Sha
```

Out[75]:
```
ShapiroResult(statistic=0.974733293056488, pvalue=2.8776883265335313e-20)
```

TEST FOR HOMOSCEDASTICITY

- We will test for homoscedasticity by using the goldfeldquandt test.
- If we get a p-value greater than 0.05, we can say that the residuals are homoscedastic. Otherwise, they are heteroscedastic.

In [76]:
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(df_pred["Residuals"], x_train3) ## Complete
lzip(name, test)
```

Out[76]:
```
[('F statistic', 1.0622319654059906), ('p-value', 0.14943725564900745)]
```

Final Model Summary

In [77]:
```
x_train_final= x_train3.copy()
x_test_final = x_test3.copy()
```

In [78]:
```
olsmodel_final = sm.OLS(y_train, x_train_final).fit()
print(olsmodel_final.summary())
```

```
                            OLS Regression Results
==========================================================================
========
Dep. Variable:     normalized_used_price   R-squared:
0.843
Model:                               OLS   Adj. R-squared:
0.841
Method:                    Least Squares   F-statistic:
557.8
Date:                   Wed, 15 Feb 2023   Prob (F-statistic):
0.00
Time:                           12:46:17   Log-Likelihood:
107.69
No. Observations:                   2417   AIC:
-167.4
Df Residuals:                       2393   BIC:
-28.41
Df Model:                             23
Covariance Type:               nonrobust
==========================================================================
===============
                           coef    std err          t      P>|t|      [0.
025      0.975]
--------------------------------------------------------------------------
---------------
const                    1.5113      0.052     28.786      0.000       1.
408       1.614
screen_size              0.0439      0.002     28.724      0.000       0.
041       0.047
selfie_camera_mp         0.0131      0.001     12.330      0.000       0.
011       0.015
ram                      0.0156      0.004      3.480      0.001       0.
007       0.024
normalized_new_price     0.4195      0.011     38.112      0.000       0.
398       0.441
years_since_release     -0.0098      0.004     -2.780      0.005      -0.
017      -0.003
```

```
brand_name_Nokia          0.1008      0.031      3.235      0.001       0.
040       0.162
brand_name_Samsung       -0.0395      0.016     -2.402      0.016      -0.
072      -0.007
brand_name_Xiaomi         0.0723      0.026      2.826      0.005       0.
022       0.122
4g_yes                    0.0422      0.015      2.800      0.005       0.
013       0.072
main_camera_mp_8.0       -0.1024      0.014     -7.493      0.000      -0.
129      -0.076
main_camera_mp_5.0       -0.1589      0.017     -9.200      0.000      -0.
193      -0.125
main_camera_mp_3.15      -0.2054      0.030     -6.901      0.000      -0.
264      -0.147
main_camera_mp_2.0       -0.2168      0.026     -8.342      0.000      -0.
268      -0.166
main_camera_mp_16.0       0.1019      0.024      4.197      0.000       0.
054       0.149
main_camera_mp_0.3       -0.3934      0.039    -10.037      0.000      -0.
470      -0.317
main_camera_mp_48.0       0.2836      0.117      2.427      0.015       0.
054       0.513
main_camera_mp_1.3       -0.3872      0.060     -6.431      0.000      -0.
505      -0.269
main_camera_mp_23.0       0.2564      0.068      3.789      0.000       0.
124       0.389
main_camera_mp_4.0       -0.3067      0.096     -3.204      0.001      -0.
494      -0.119
main_camera_mp_10.0      -0.2505      0.117     -2.133      0.033      -0.
481      -0.020
main_camera_mp_6.7       -0.2403      0.121     -1.980      0.048      -0.
478      -0.002
main_camera_mp_16.3       0.5478      0.234      2.344      0.019       0.
089       1.006
main_camera_mp_8.1       -0.2956      0.118     -2.510      0.012      -0.
527      -0.065
==========================================================================
=====
Omnibus:                      193.234   Durbin-Watson:
1.911
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               36
3.597
Skew:                          -0.550   Prob(JB):                      1.1
1e-79
Kurtosis:                       4.549   Cond. No.
889.
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
```

In [79]:
```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel_final_train_perf = model_performance_regression(olsmodel_final,
olsmodel_final_train_perf
```

```
Training Performance
```

Out[79]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.231426	0.181502	0.842797	0.84122	4.344607

In [80]:
```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmodel_final_test_perf = model_performance_regression(olsmodel_final, x
olsmodel_final_test_perf
```

```
Test Performance
```

Out[80]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.240476	0.186263	0.839667	0.835864	4.511603

Actionable Insights and Recommendations

-
