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

Importing necessary libraries

In [1]:
```
%load_ext nb_black

import pandas as pd
import numpy as np

# for visualizing data
import matplotlib.pyplot as plt
import seaborn as sns

# For randomized data splitting
from sklearn.model_selection import train_test_split

# To build linear regression_model
import statsmodels.api as sm

# To check model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Loading the dataset

In [2]:
```
data = pd.read_csv("used_device_data.csv")
```

Data Overview

- Observations
- Sanity checks

In [3]:
```
data.shape
```

Out[3]:
```
(3454, 15)
```

In [4]:
```
data.head()
```

Out[4]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_us
0	Honor	Android	14.50	yes	no	13.0	5.0	64.0	3.0	3020.0	146.0	2020	1
1	Honor	Android	17.30	yes	yes	13.0	16.0	128.0	8.0	4300.0	213.0	2020	3
2	Honor	Android	16.69	yes	yes	13.0	8.0	128.0	8.0	4200.0	213.0	2020	1
3	Honor	Android	25.50	yes	yes	13.0	8.0	64.0	6.0	7250.0	480.0	2020	3
4	Honor	Android	15.32	yes	no	13.0	8.0	64.0	3.0	5000.0	185.0	2020	2

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
```

- several variables are of 'object' type that we will change to 'category'

In [6]:
```
# change 'object' data type to 'category'
for col in data.columns[data.dtypes == "object"]:
    data[col] = data[col].astype("category")
```

In [7]:
```
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   category
 1   os                     3454 non-null   category
 2   screen_size            3454 non-null   float64
 3   4g                     3454 non-null   category
 4   5g                     3454 non-null   category
 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: category(4), float64(9), int64(2)
memory usage: 312.2 KB
```

In [8]:
```
# check for missing values
data.isnull().sum()
```

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

- a check for missing values shows several variables that need to be addressed.
- no duplicated values.

In [9]:
```
# check for duplicates
data.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
# let's view a sample of the data
data.sample(n=10, random_state=1)
```

Out[10]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_
866	Others	Android	15.24	no	no	8.00	2.0	16.0	4.00	3000.0	206.0	2014	
957	Celkon	Android	10.16	no	no	3.15	0.3	512.0	0.25	1400.0	140.0	2013	
280	Infinix	Android	15.39	yes	no	NaN	8.0	32.0	2.00	5000.0	185.0	2020	
2150	Oppo	Android	12.83	yes	no	13.00	16.0	64.0	4.00	3200.0	148.0	2017	
93	LG	Android	15.29	yes	no	13.00	5.0	32.0	3.00	3500.0	179.0	2019	
1040	Gionee	Android	12.83	yes	no	13.00	8.0	32.0	4.00	3150.0	166.0	2016	
3170	ZTE	Others	10.16	no	no	3.15	5.0	16.0	4.00	1400.0	125.0	2014	
2742	Sony	Android	12.70	yes	no	20.70	2.0	16.0	4.00	3000.0	170.0	2013	
102	Meizu	Android	15.29	yes	no	NaN	20.0	128.0	6.00	3600.0	165.0	2019	
1195	HTC	Android	10.29	no	no	8.00	2.0	32.0	4.00	2000.0	146.0	2015	

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.

- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Questions:

1. What does the distribution of normalized used device prices look like?
2. What percentage of the used device market is dominated by Android devices?
3. The amount of RAM is important for the smooth functioning of a device. How does the amount of RAM vary with the brand?
4. A large battery often increases a device's weight, making it feel uncomfortable in the hands. How does the weight vary for phones and tablets offering large batteries (more than 4500 mAh)?
5. Bigger screens are desirable for entertainment purposes as they offer a better viewing experience. How many phones and tablets are available across different brands with a screen size larger than 6 inches?
6. A lot of devices nowadays offer great selfie cameras, allowing us to capture our favorite moments with loved ones. What is the distribution of devices offering greater than 8MP selfie cameras across brands?
7. Which attributes are highly correlated with the normalized price of a used device?

In [11]:
```
data.describe(include="all").T
```

Out[11]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
brand_name	3454	34	Others	502	NaN	NaN	NaN	NaN	NaN	NaN	NaN
os	3454	4	Android	3214	NaN	NaN	NaN	NaN	NaN	NaN	NaN
screen_size	3454.0	NaN	NaN	NaN	13.713115	3.80528	5.08	12.7	12.83	15.34	30.71
4g	3454	2	yes	2335	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5g	3454	2	no	3302	NaN	NaN	NaN	NaN	NaN	NaN	NaN
main_camera_mp	3275.0	NaN	NaN	NaN	9.460208	4.815461	0.08	5.0	8.0	13.0	48.0
selfie_camera_mp	3452.0	NaN	NaN	NaN	6.554229	6.970372	0.0	2.0	5.0	8.0	32.0
int_memory	3450.0	NaN	NaN	NaN	54.573099	84.972371	0.01	16.0	32.0	64.0	1024.0
ram	3450.0	NaN	NaN	NaN	4.036122	1.365105	0.02	4.0	4.0	4.0	12.0
battery	3448.0	NaN	NaN	NaN	3133.402697	1299.682844	500.0	2100.0	3000.0	4000.0	9720.0
weight	3447.0	NaN	NaN	NaN	182.751871	88.413228	69.0	142.0	160.0	185.0	855.0
release_year	3454.0	NaN	NaN	NaN	2015.965258	2.298455	2013.0	2014.0	2015.5	2018.0	2020.0
days_used	3454.0	NaN	NaN	NaN	674.869716	248.580166	91.0	533.5	690.5	868.75	1094.0
normalized_used_price	3454.0	NaN	NaN	NaN	4.364712	0.588914	1.536867	4.033931	4.405133	4.7557	6.619433
normalized_new_price	3454.0	NaN	NaN	NaN	5.233107	0.683637	2.901422	4.790342	5.245892	5.673718	7.847841

In [12]:
```
# fix the missing values
# we first create a copy of the original data to avoid changes to it
data = data.copy()
```

In [13]:
```
# dropping missing values in the target
data.dropna(subset=["normalized_used_price"], inplace=True)
```

In [14]:
```
# filling missing values using the column median for the predictor variables
medianFiller = lambda x: x.fillna(x.median())
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()
data[numeric_columns] = data[numeric_columns].apply(medianFiller, axis=0)
```

In [15]:
```
data.isnull().sum()
```

Out[15]:
```
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
```

- used the median to fill in missing values for each variable respectively.

In [16]:
```
# check the number of unique values in each column of the dataframe
data.nunique()
```

Out[16]:
```
brand_name              34
os                       4
screen_size            142
4g                       2
5g                       2
main_camera_mp          41
selfie_camera_mp        37
int_memory              15
ram                     12
battery                324
weight                 555
release_year             8
days_used              924
normalized_used_price 3094
normalized_new_price  2988
dtype: int64
```

In [17]:
```
data.brand_name.value_counts()
```

Out[17]:
```
Others        502
Samsung       341
Huawei        251
LG            201
Lenovo        171
ZTE           140
Xiaomi        132
Oppo          129
Asus          122
Alcatel       121
Vivo          117
Micromax      117
Honor         116
HTC           110
Nokia         106
Motorola      106
Sony           86
Meizu          62
Gionee         56
Acer           51
XOLO           49
Panasonic      47
Realme         41
Apple          39
Lava           36
Celkon         33
Spice          30
Karbonn        29
Microsoft      22
OnePlus        22
Coolpad        22
BlackBerry     22
Google         15
Infinix        10
Name: brand_name, dtype: int64
```

- there are significant number of brands that make cell phones

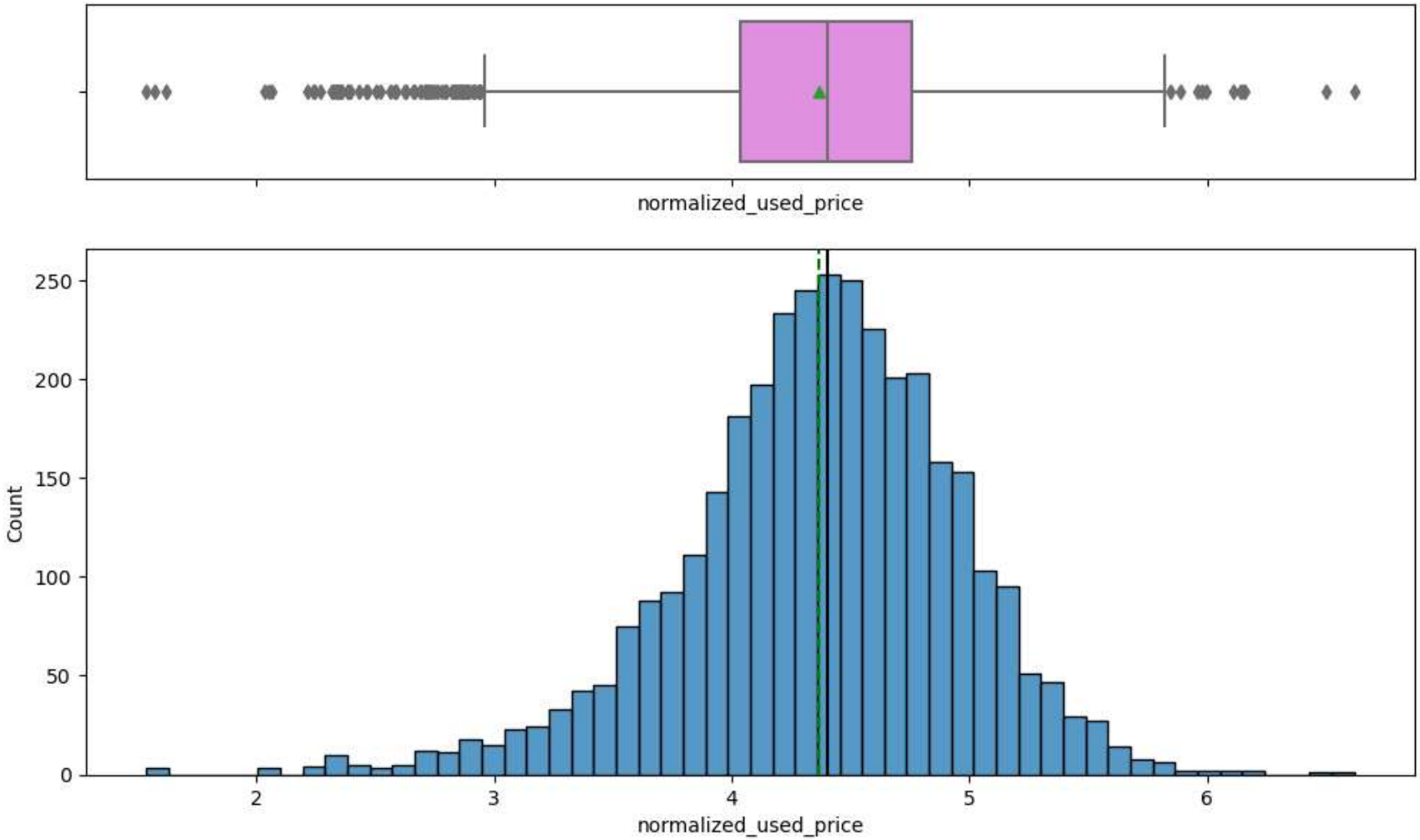
In [18]:
```
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined
```

```
    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    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 star will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) 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

# 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 False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
```

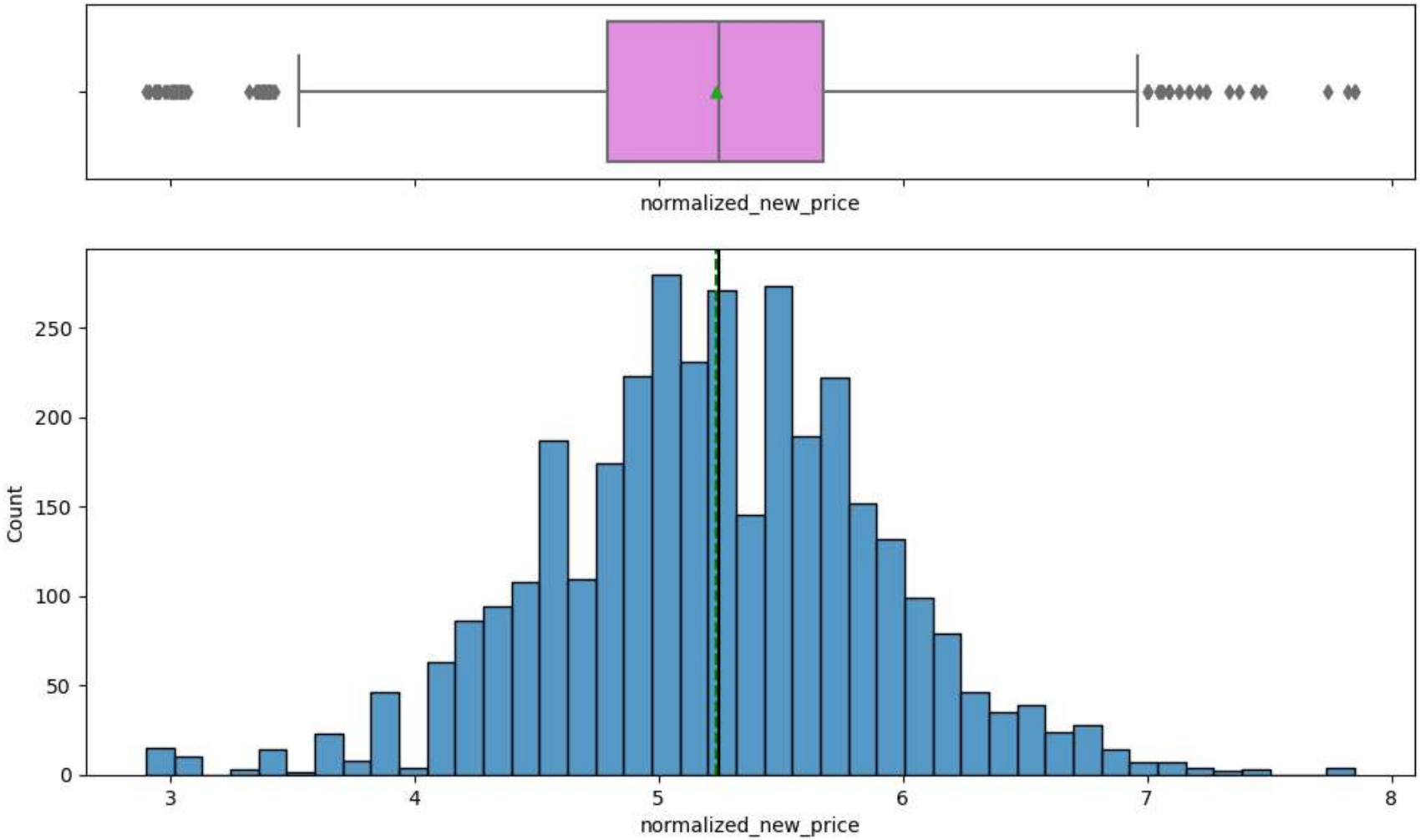
```
    else:
        plt.figure(figsize=(n + 1, 5))

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

    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
```

1. Distribution normalized used device prices

In [19]:
```
# no. 1 distribution of normalized used device prices


histogram_boxplot(data, "normalized_used_price")
```

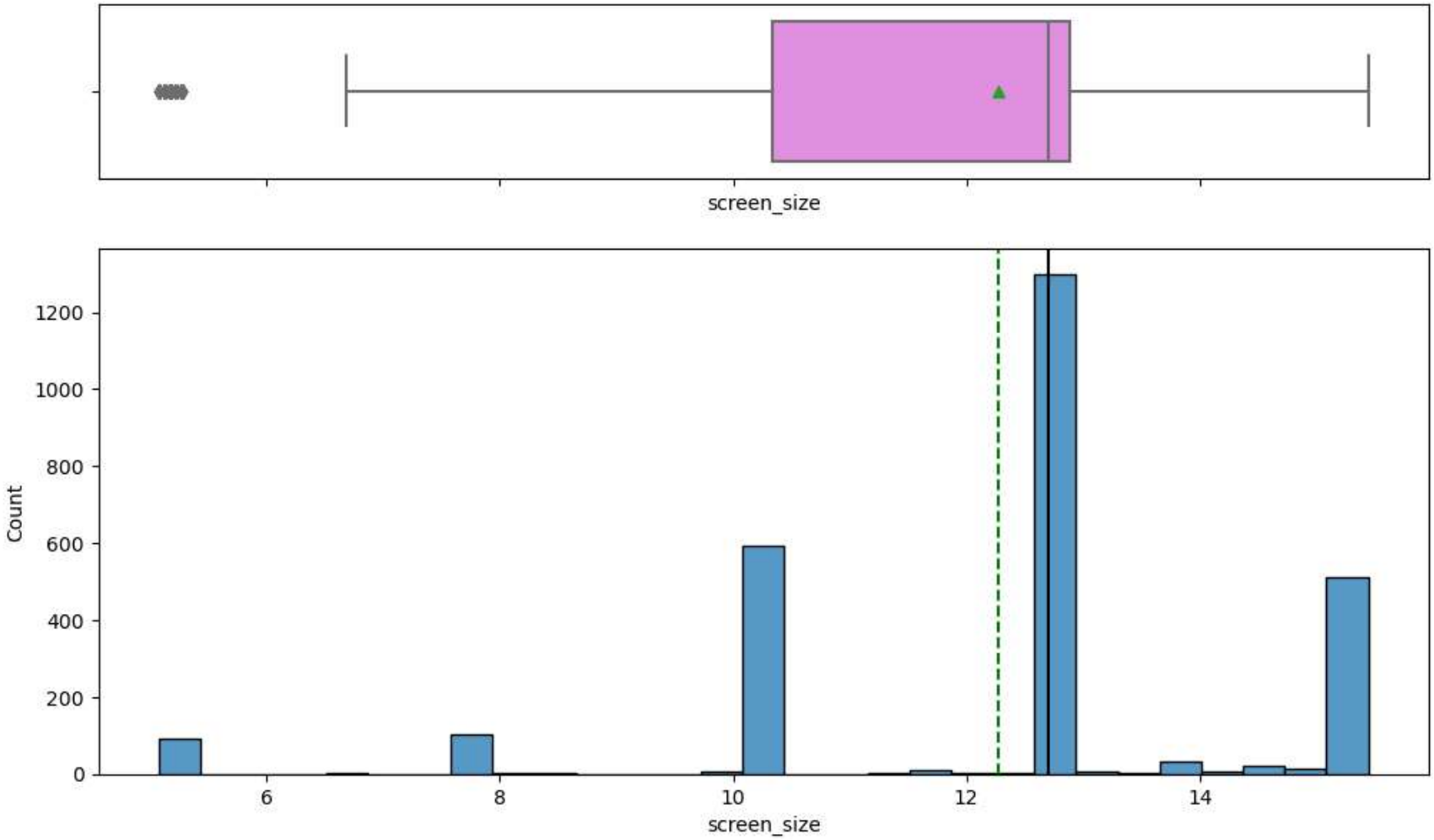


In [20]:
```
histogram_boxplot(data, "normalized_new_price")
```



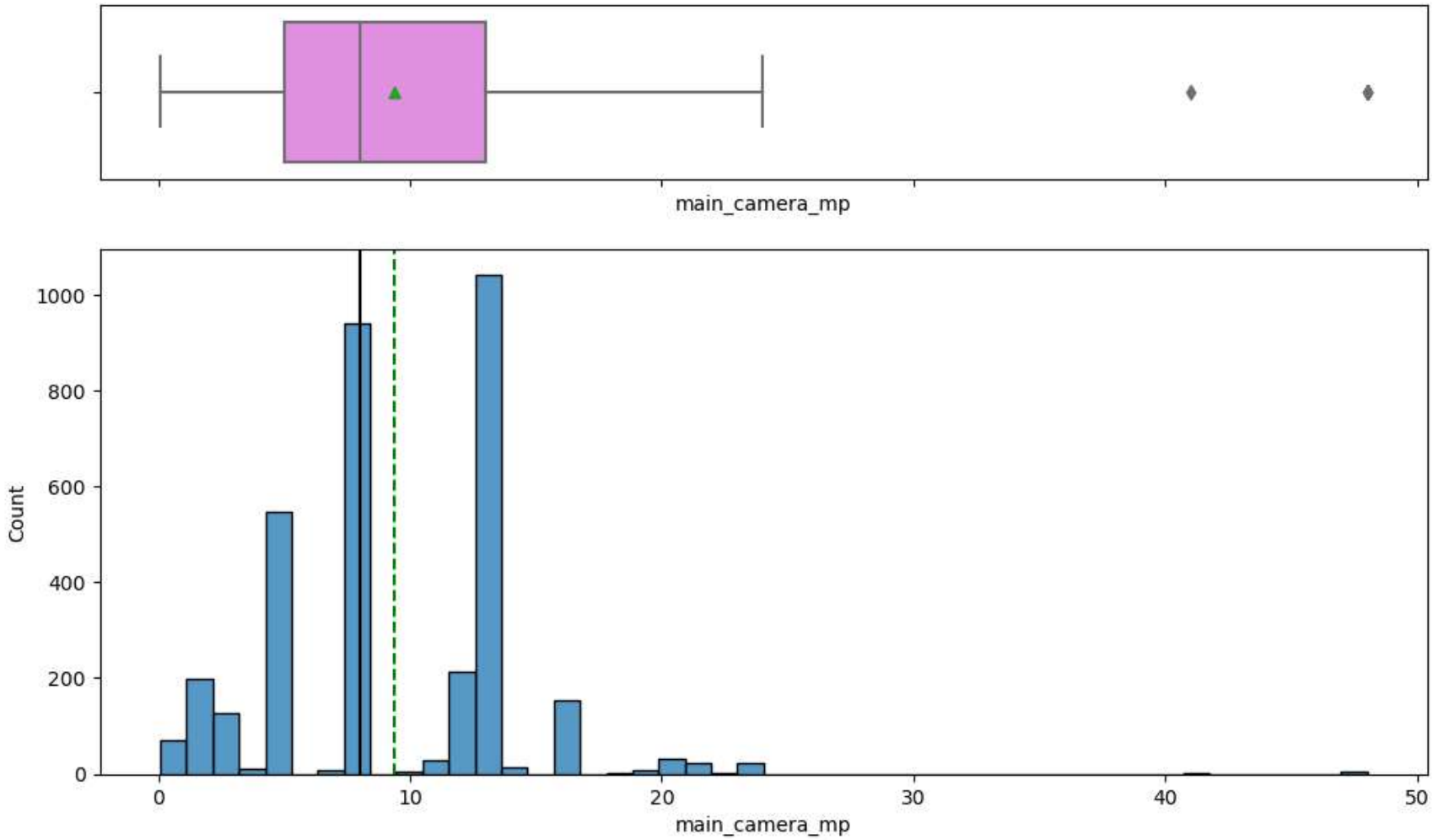
- both new and used prices are approximately normally distributed

In [21]:
```
screen_size_under_1545 = data[data.screen_size < 15.45]
histogram_boxplot(screen_size_under_1545, "screen_size")
```

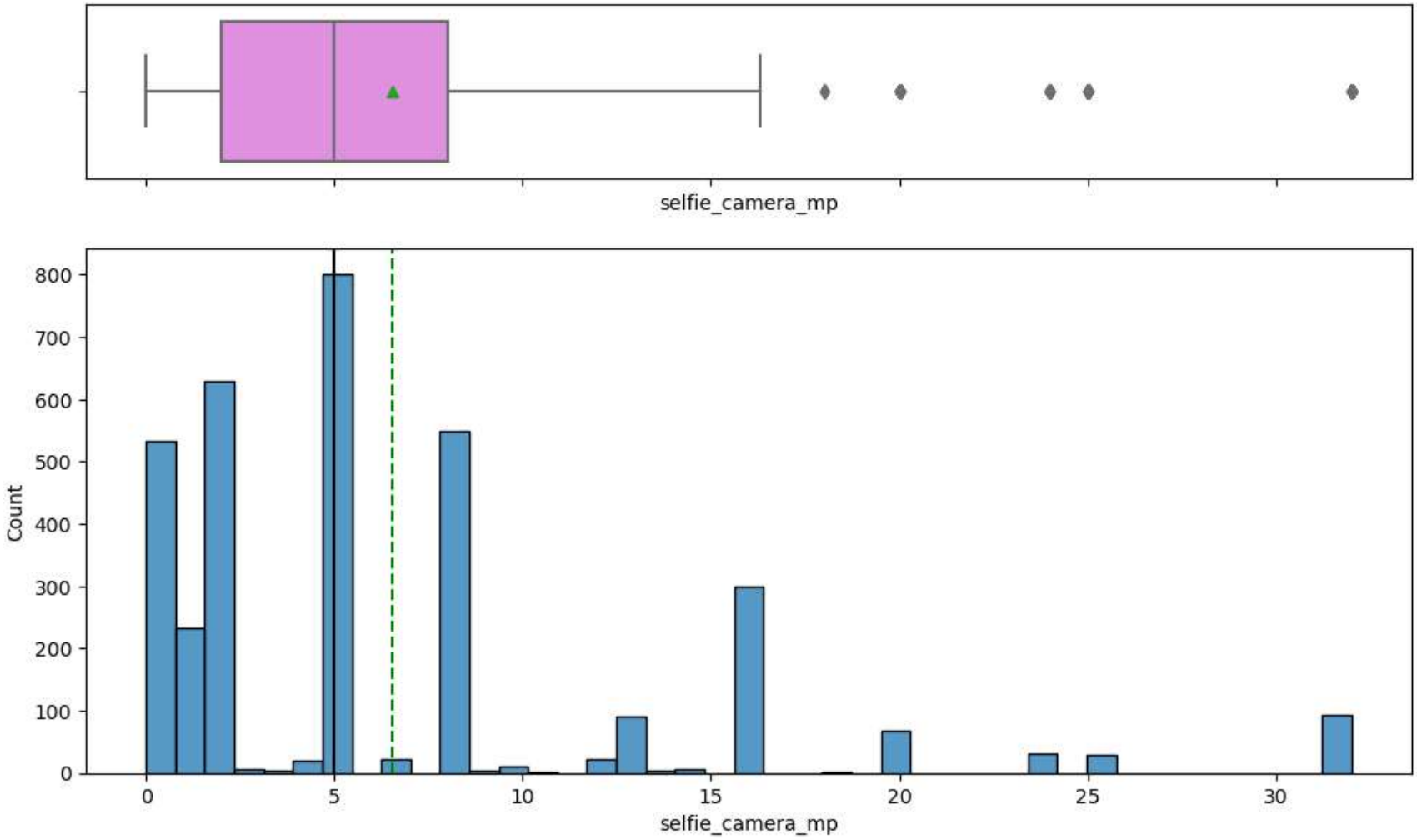


- some very small screen sizes create left skew in the screen size data

In [22]:
```
histogram_boxplot(data, "main_camera_mp")
```

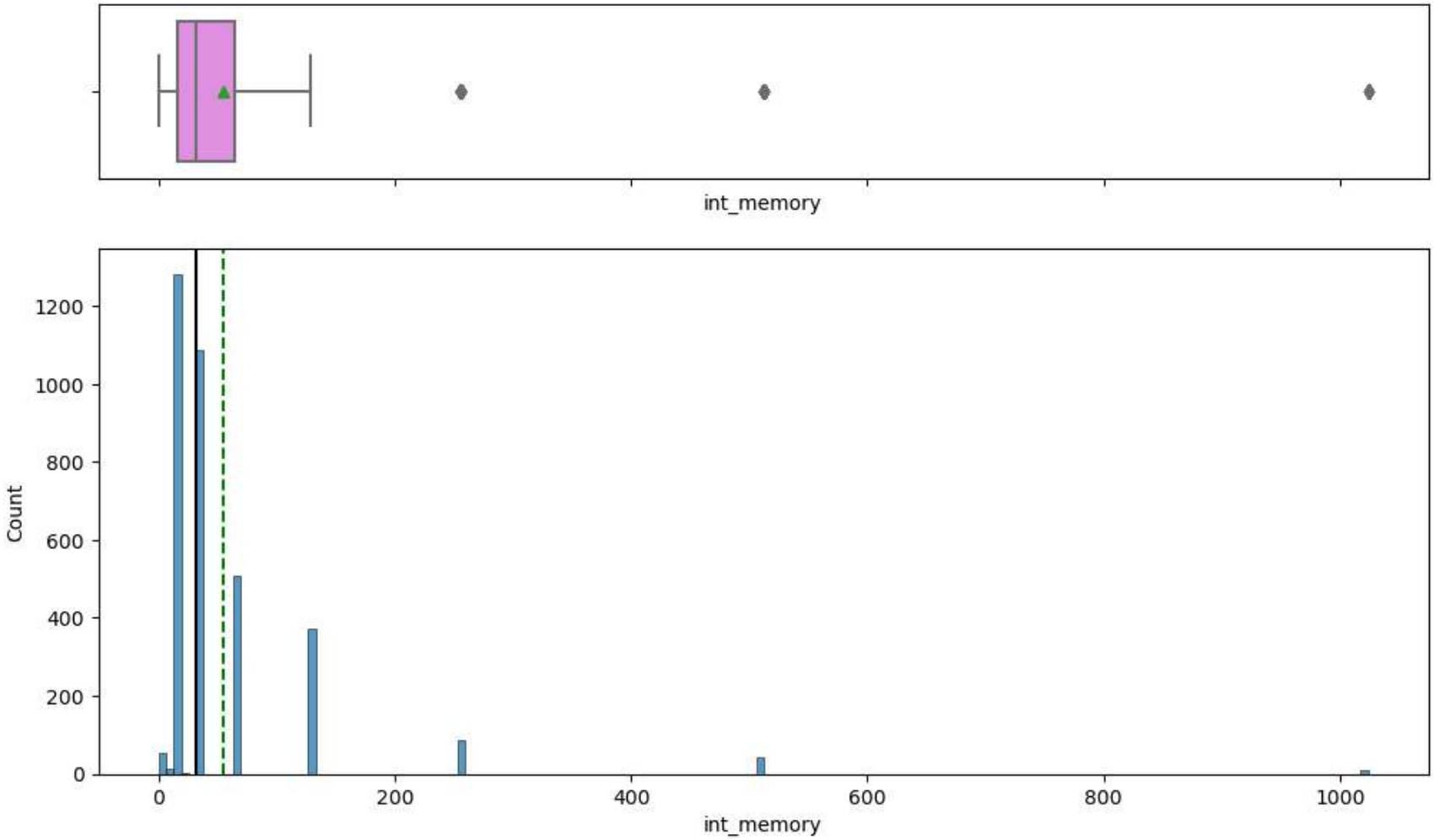


In [23]:
```
histogram_boxplot(data, "selfie_camera_mp")
```



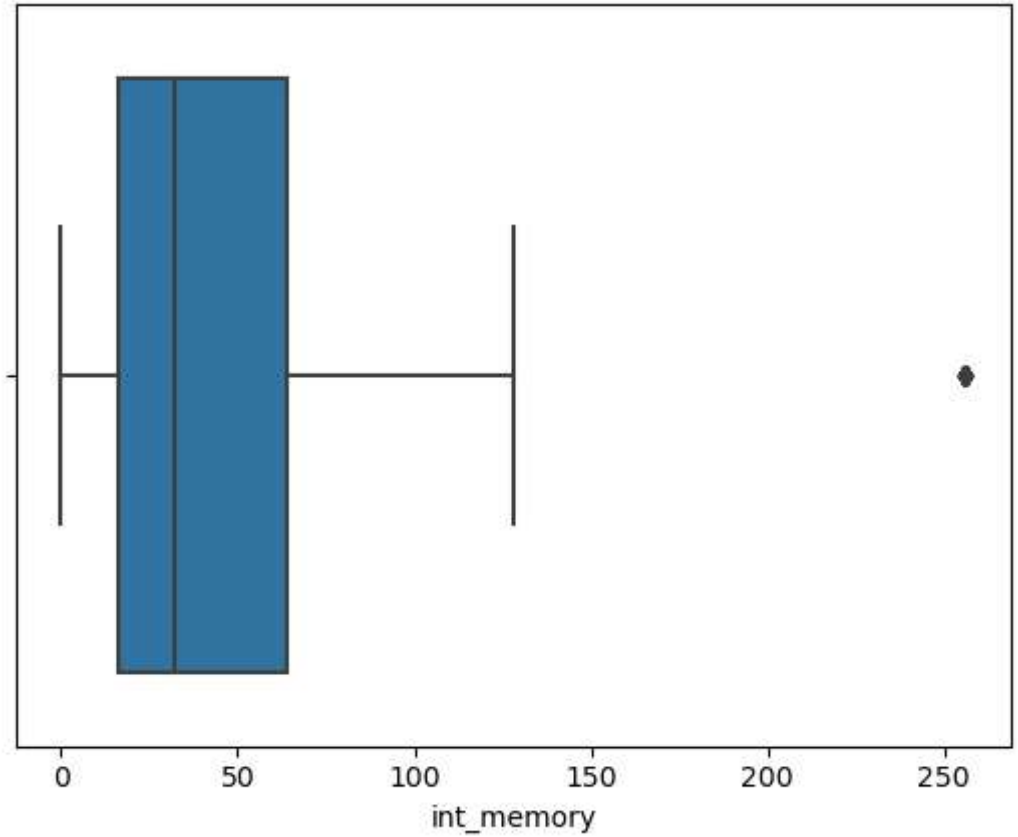
- both the main and selfie camera data is a little skewed to the right due to some very large megapixel models.

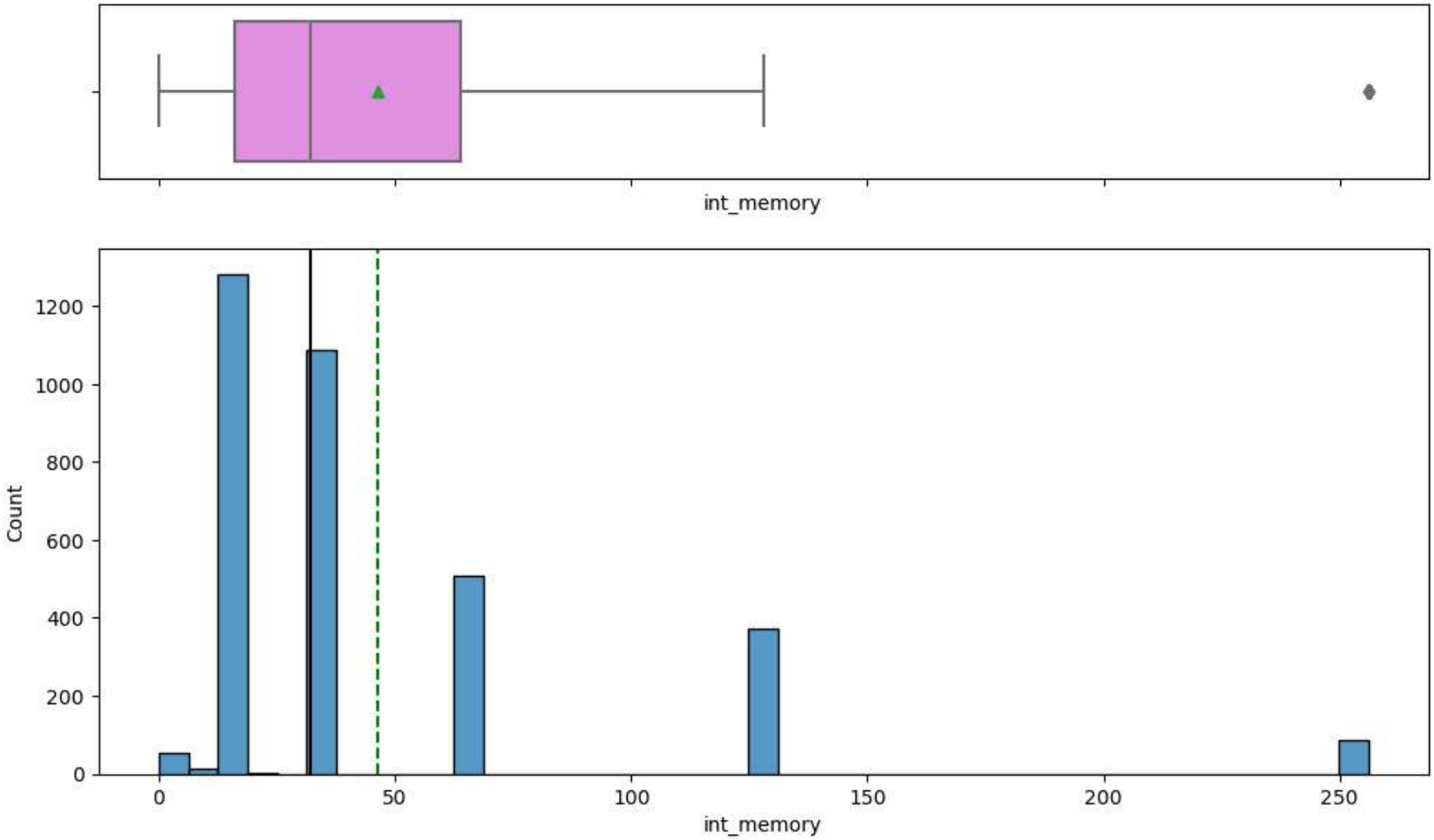
In [24]:
```
histogram_boxplot(data, "int_memory")
```



- internal memory skewed right due to a few large outliers

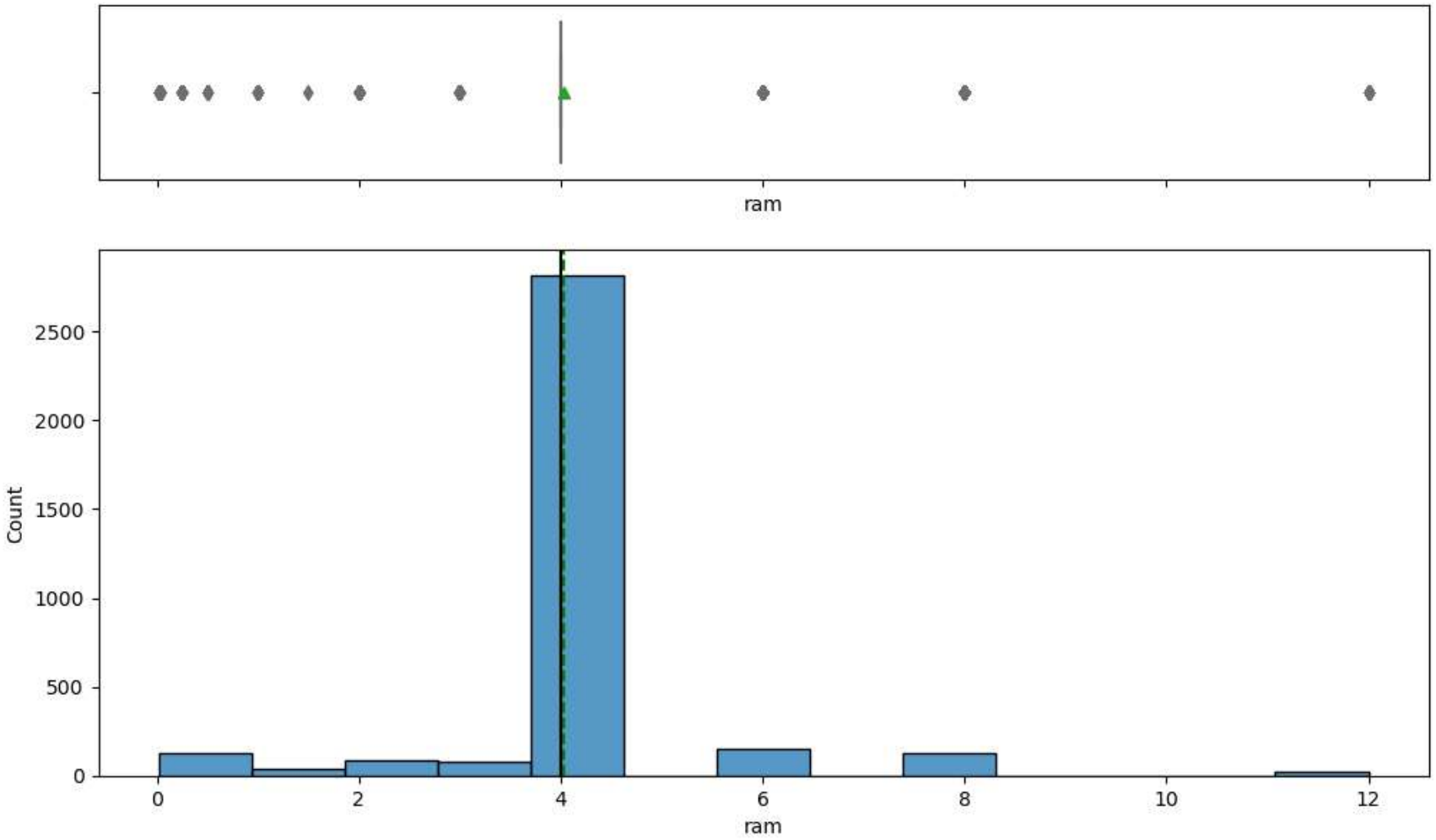
In [25]:
```
# drop int_memory greater than 400 - outlier treatment
data_mem = data.copy()
data_mem = data_mem[data.int_memory < 400]
sns.boxplot(data=data_mem, x="int_memory")


histogram_boxplot(data_mem, "int_memory")
```


0
50
100
150
200
250
int_memory

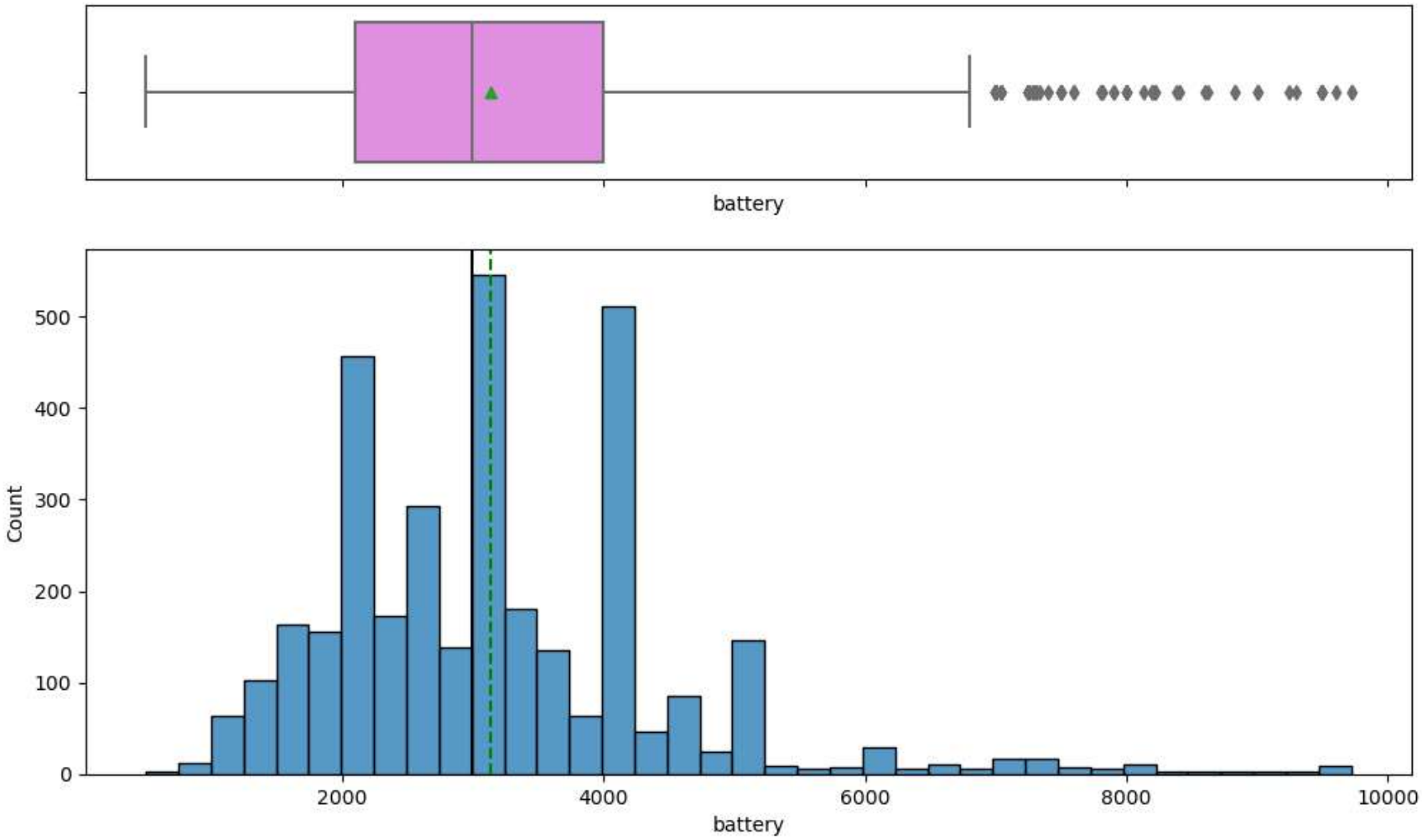


- to deal with the skew for internal memory, we treat outliers by dropping data points that are greater than 400 GB.

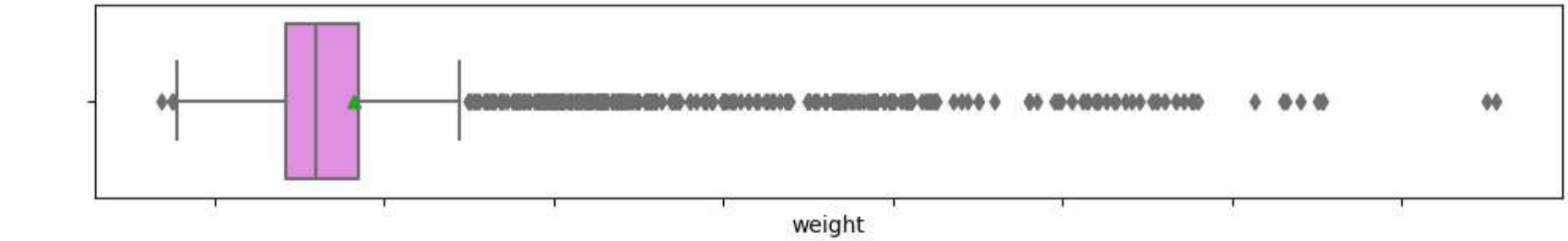
In [26]:
```
histogram_boxplot(data, "ram")
```

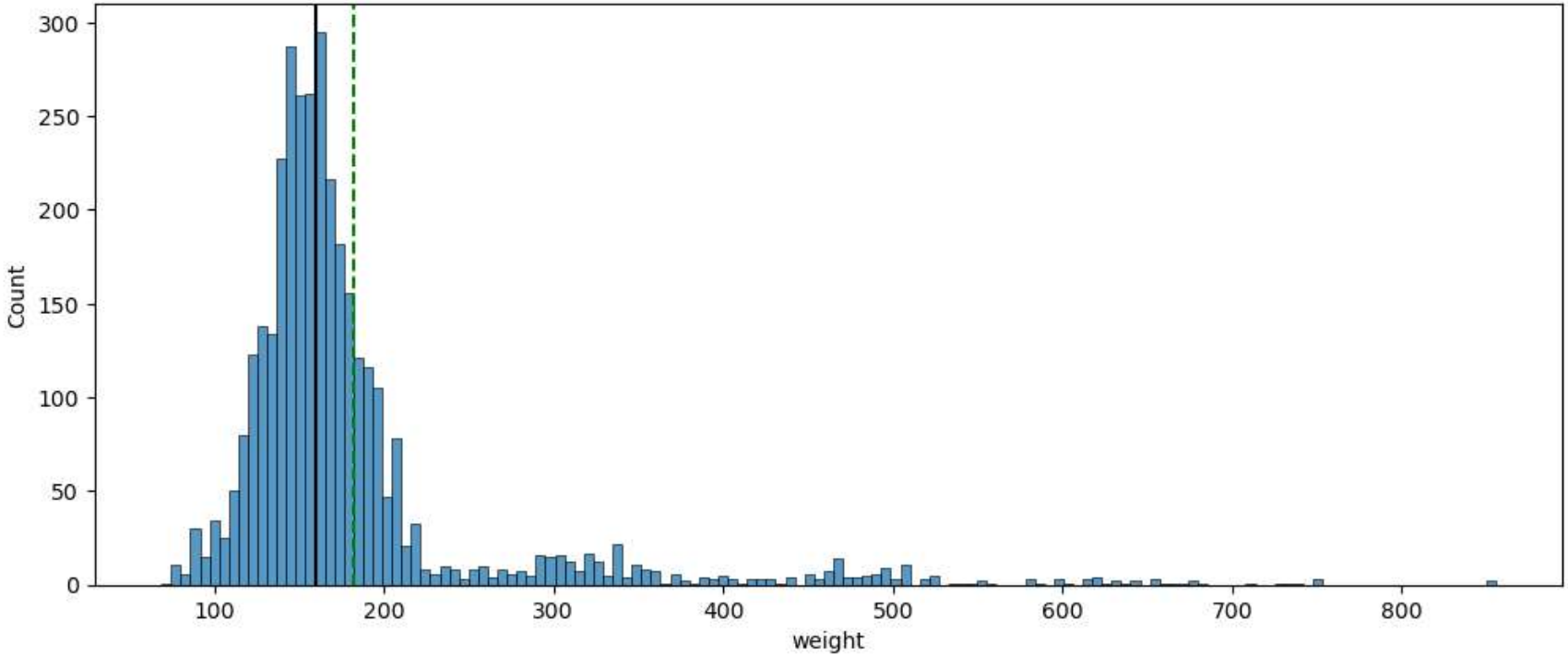


In [97]:
```
histogram_boxplot(data, "battery")
```

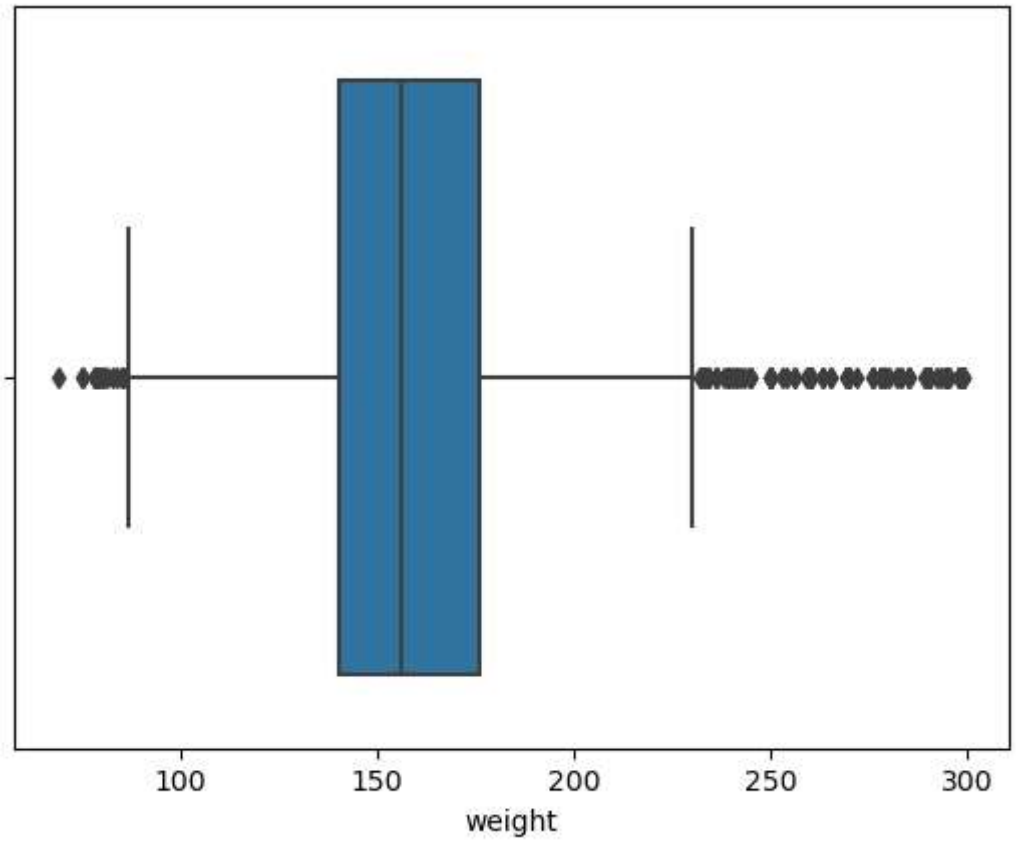


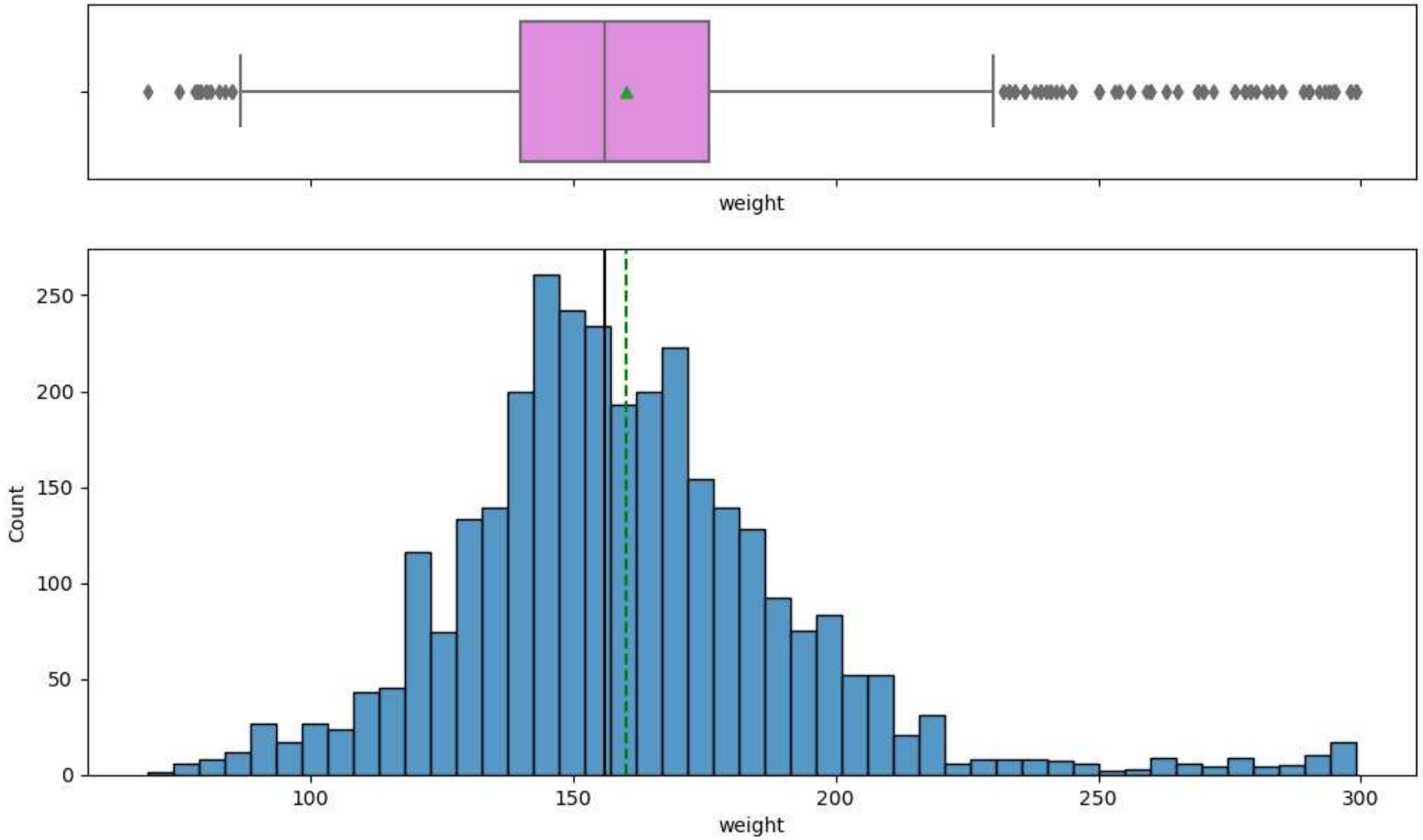
In [27]:
```
histogram_boxplot(data, "weight")
```





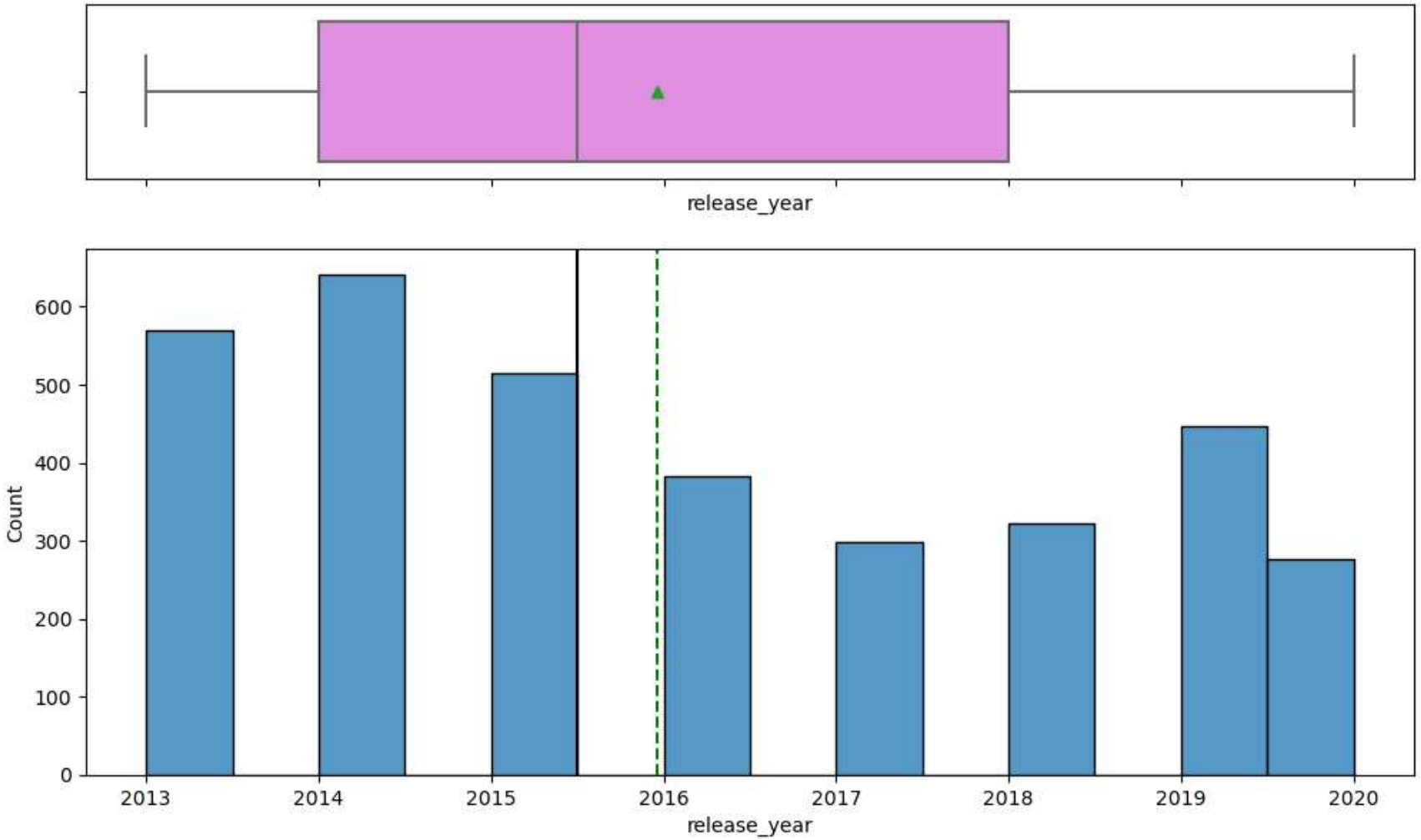
In [28]:
```
# drop weight greater than 300 - outlier treatment
data_weight = data.copy()
data_weight = data_weight[data.weight < 300]
sns.boxplot(data=data_weight, x="weight")


histogram_boxplot(data_weight, "weight")
```


100
150
200
250
300
weight



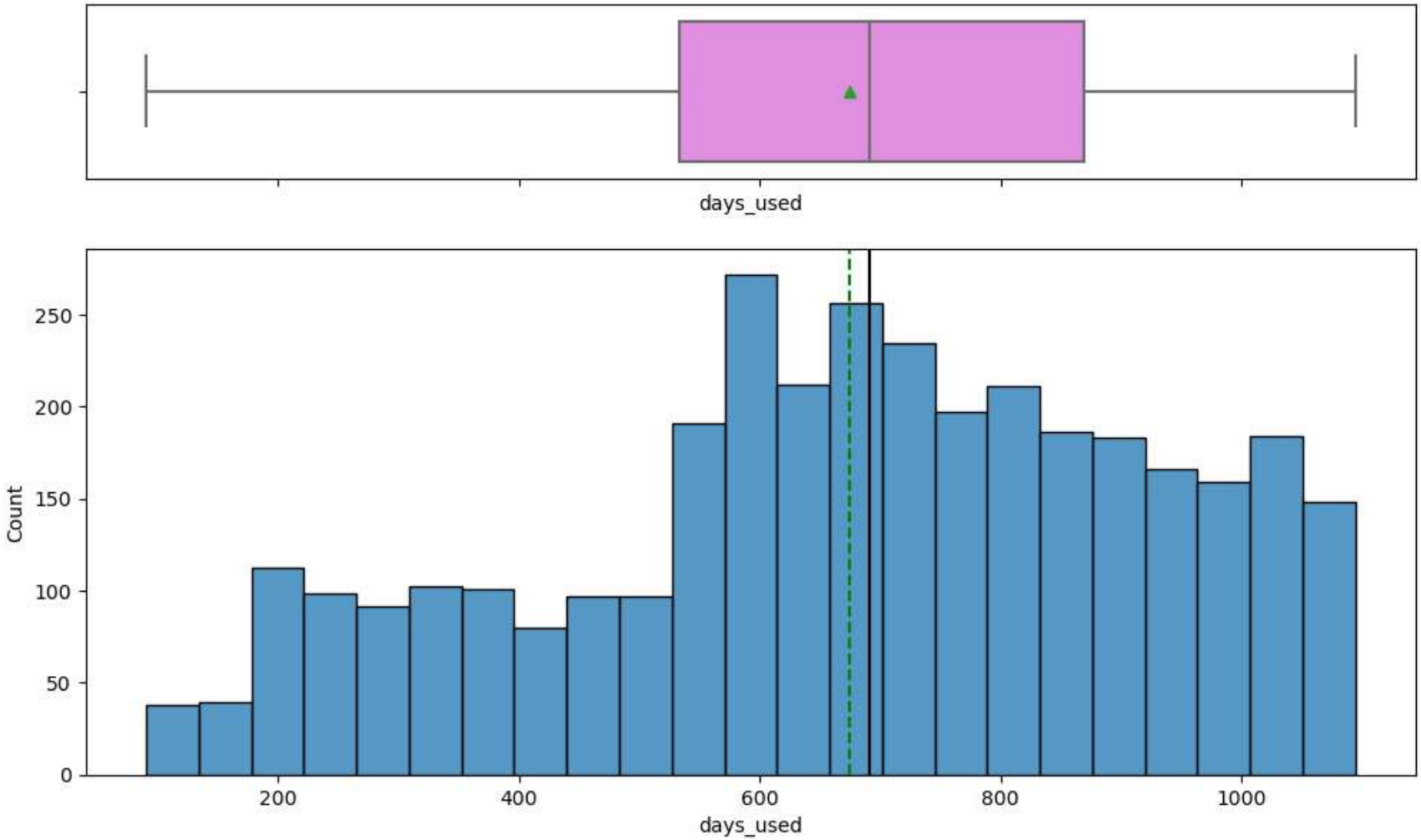
- to deal with outliers that are causing skew in the distribution of weight, we will drop those values that are greater than 300 grams.

In [29]:
```
histogram_boxplot(data, "release_year")
```



- as we would probably guess, the older models are the ones most available on the resale market.

In [30]:
```
histogram_boxplot(data, "days_used")
```



- the number of days used could be difficult to interpret without making too many assumption. Looking at days used with the year released would be helpful. As it is, the assumption would be that the older the phone, the more days used it will have on it.

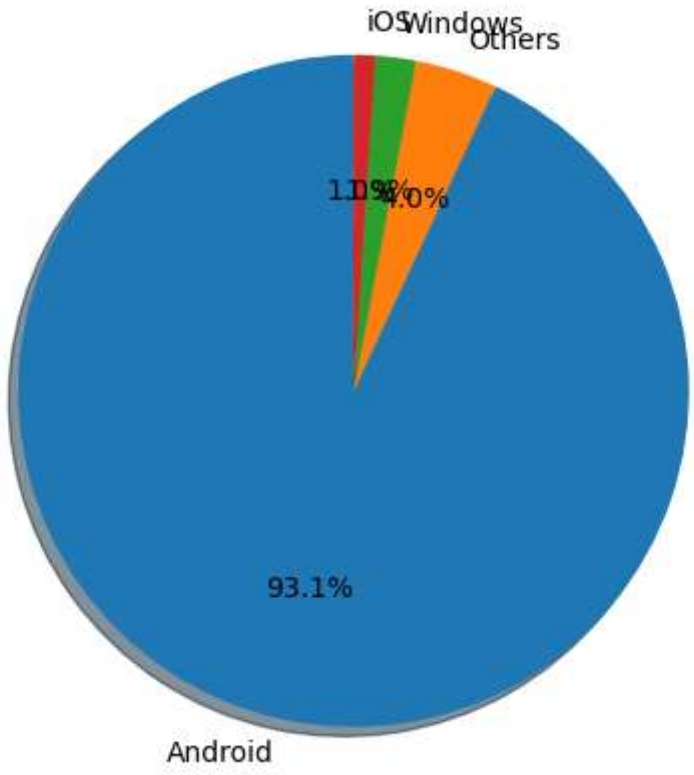
2. A look at the percentage of the device market for Android

In [31]:
```
from collections import Counter
count_of_os = Counter(data['os'])
```

```
print(count_of_os)
```

```
Counter({'Android': 3214, 'Others': 137, 'Windows': 67, 'iOS': 36})
```

In [10]:
```
# 2 percentage of device market for Android
labels = 'Android', 'Others', "Windows", 'iOS'
sizes = [3214, 137, 67, 36]

fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal')

plt.show()
```

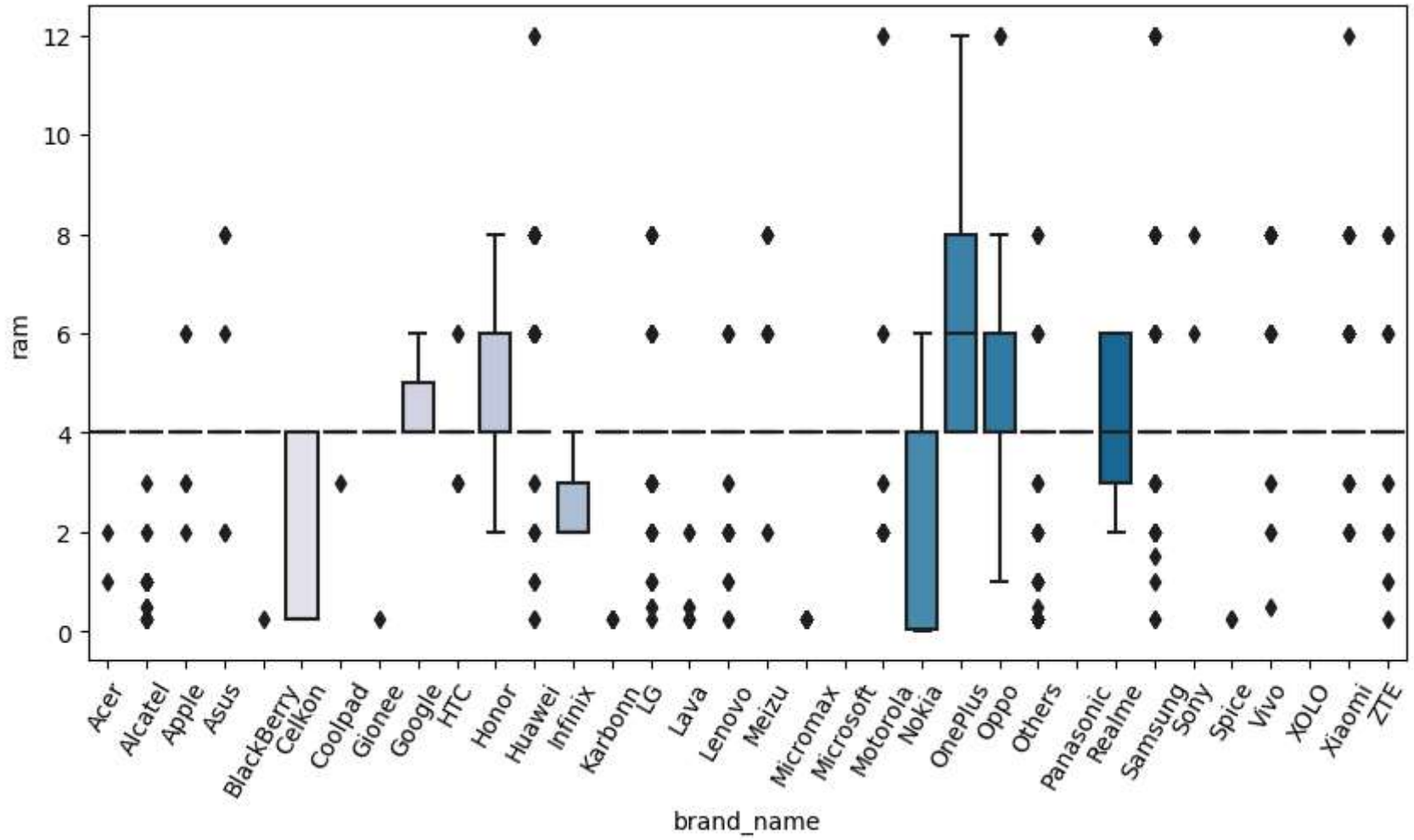


- Missing value treatment
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)

- Preparing data for modeling
- Any other preprocessing steps (if needed)

3. RAM by brand

In [32]:
```
# 3 RAM by brand

plt.figure(figsize=(10,5))
sns.boxplot(x = "brand_name", y = "ram", data = data, palette = 'PuBu')
plt.xticks(rotation = 60)
plt.show()
```



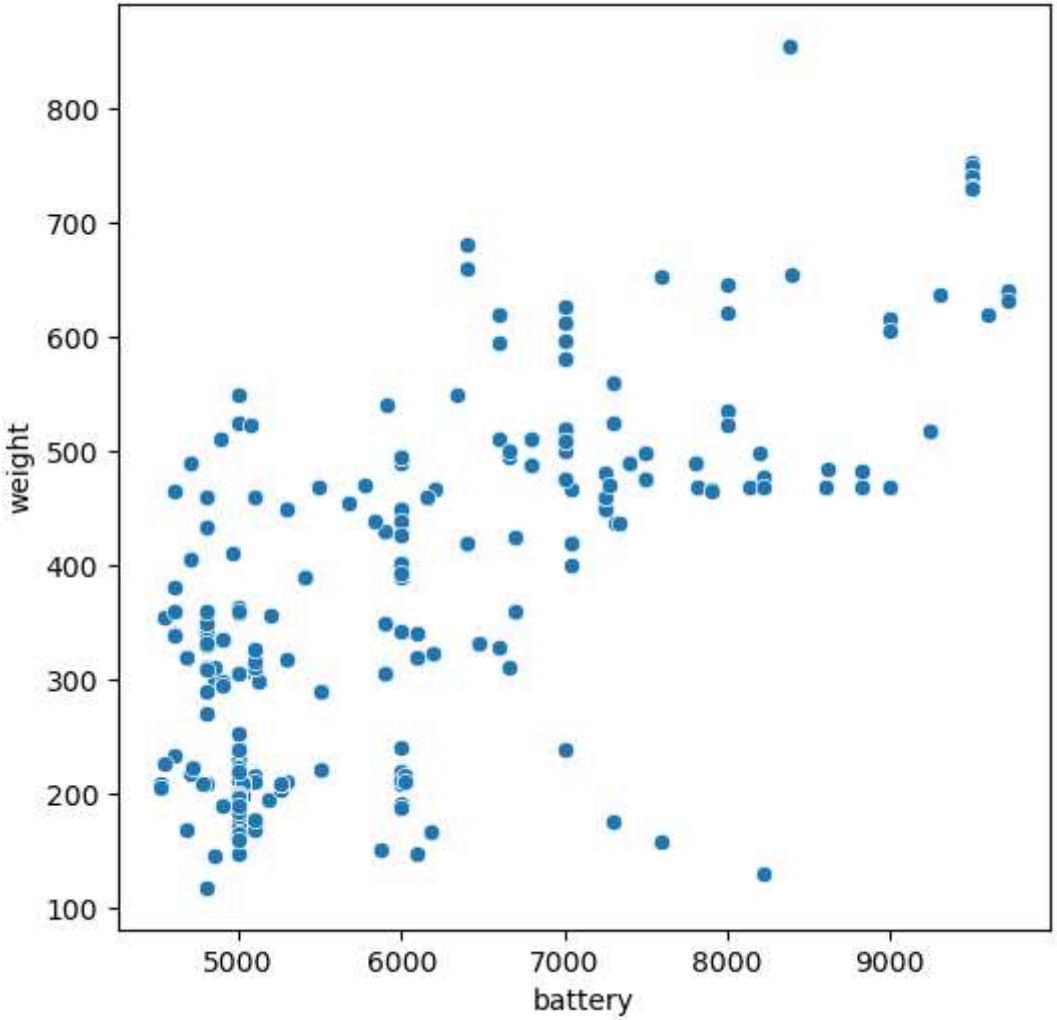
- brands tend to vary quite a bit by what they offer in terms of RAM. Some offer from very little to over 8 GB, while others stick to a tighter range such as 3-6 GB.

4. How weight varies by battery sizes greater than 4500 mAh

In [33]:
```
q = data[data.battery > 4500]
q.count()
```

Out[33]:
```
brand_name              341
os                      341
screen_size             341
4g                      341
5g                      341
main_camera_mp          341
selfie_camera_mp        341
int_memory              341
ram                     341
battery                 341
weight                  341
release_year            341
days_used               341
normalized_used_price   341
normalized_new_price    341
dtype: int64
```

In [34]:
```
# 4 weight vary by battery sizes greater than 4500 mAh

q = data[data.battery > 4500]
plt.figure(figsize=(6, 6))
sns.scatterplot(data=q, x= 'battery', y="weight")
plt.show()
```

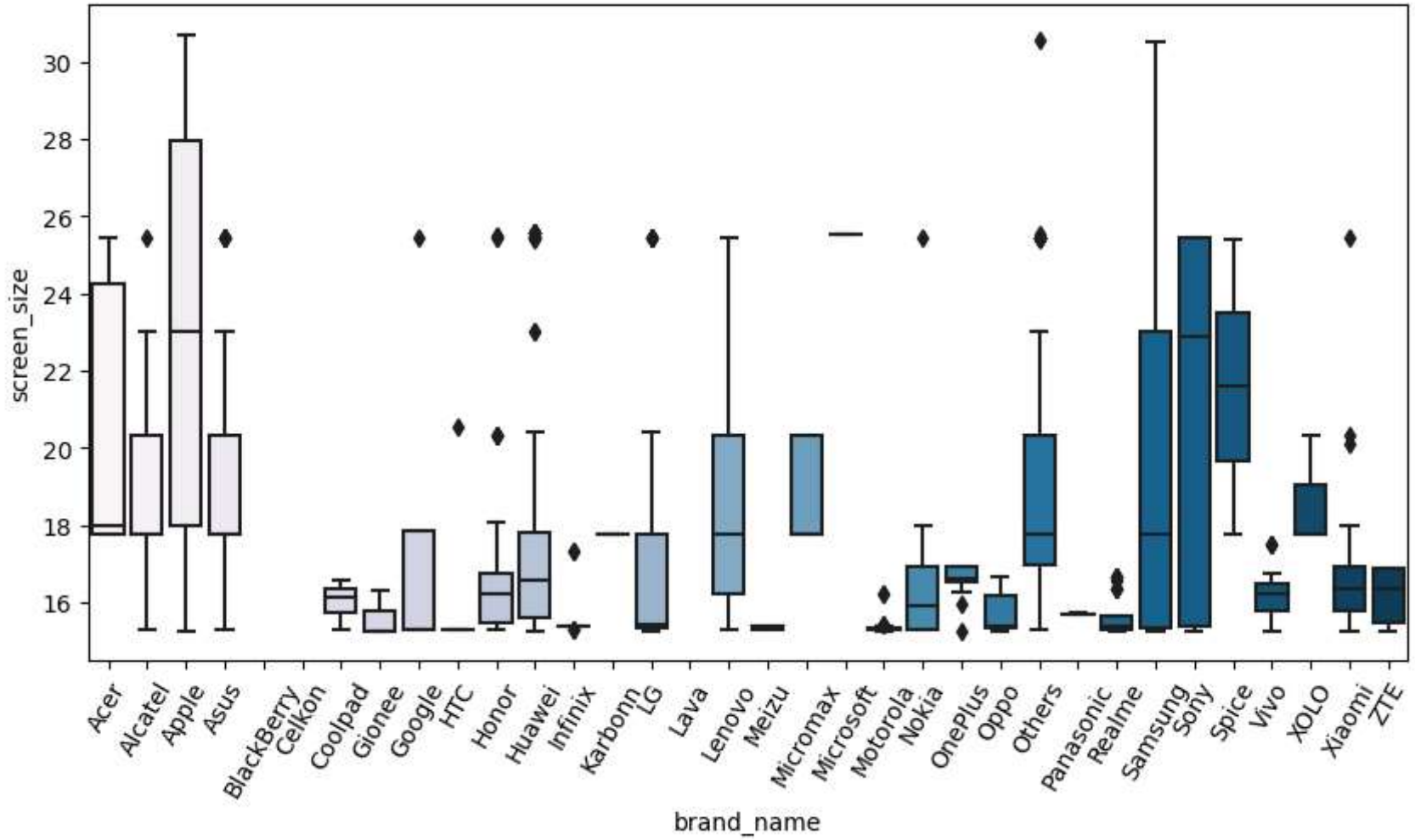


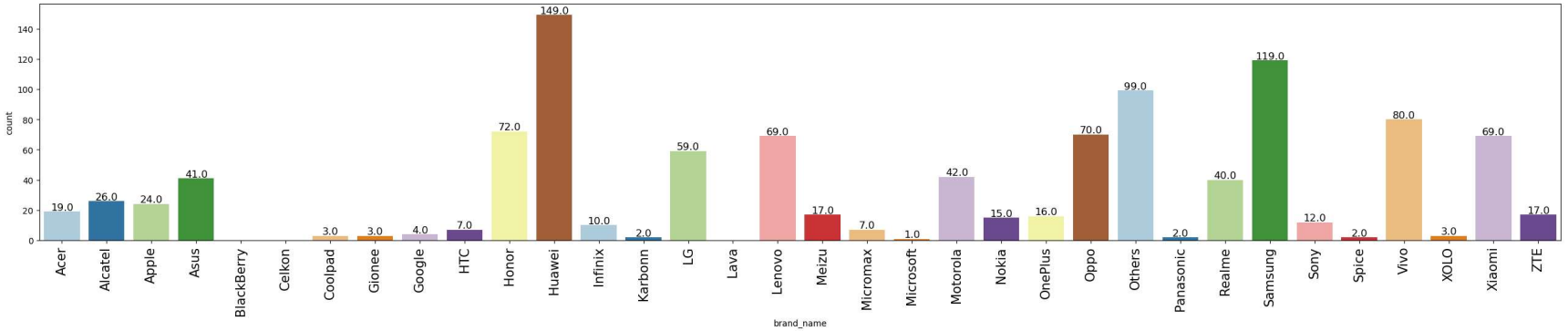
- there is a general positive correlation where the more capacity of the battery, the heavier the batter will be.

5. How many devices with screen size > 6 inches (15.24 cm) across brands

In [35]:
```
# 5 how many devices with screen size > 6 inches (15.24 cm) across brands

w = data[data.screen_size > 15.24]
```

```
plt.figure(figsize=(10, 5))
sns.boxplot(x="brand_name", y="screen_size", data=w, palette="PuBu")
plt.xticks(rotation=60)
plt.show()

labeled_barplot(w, "brand_name")
```

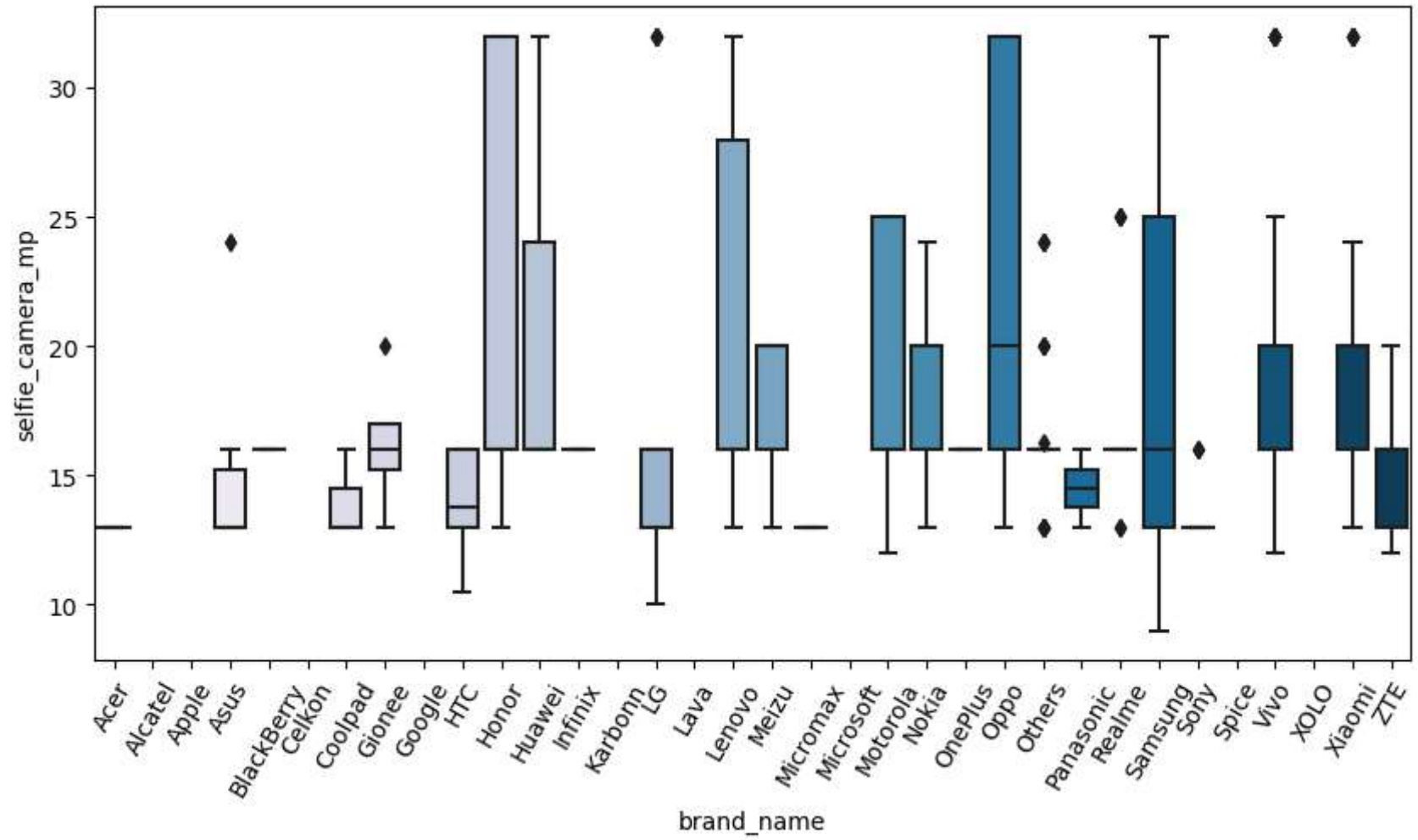


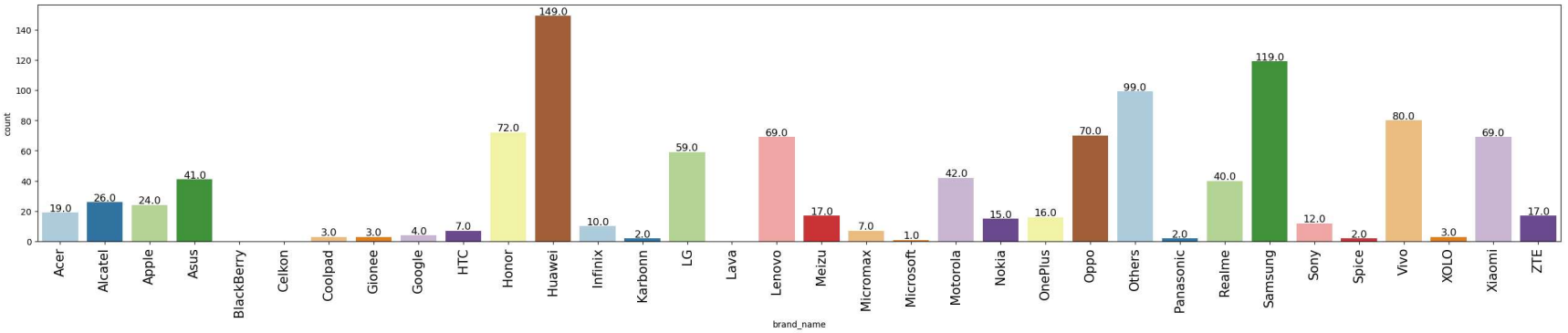


- many brands offer screen sizes greater than 6 inches, but Huawei and Samsung offer the most.

6. Devices offering greater than 8 mp selfie feature across brands

In [36]:
```
# 6 devices offering > 8MP selfie feature across brands

t = data[data.selfie_camera_mp > 8]

plt.figure(figsize=(10,5))
sns.boxplot(x = "brand_name", y = "selfie_camera_mp", data = t, palette = 'PuBu')
plt.xticks(rotation = 60)
plt.show()

labeled_barplot(w, 'brand_name')
```


selfie_camera_mp
30
25
20
15
10
Acer
Alcatel
Apple
Asus
BlackBerry
Celkon
Coolpad
Gionee
Google
HTC
Honor
Huawei
Infinix
Karbonn
LG
Lava
Lenovo
Meizu
Micromax
Microsoft
Motorola
Nokia
OnePlus
Oppo
Others
Panasonic
Realme
Samsung
Sony
Spice
Vivo
XOLO
Xiaomi
ZTE
brand_name


count
brand_name
19.0
26.0
24.0
41.0
3.0
3.0
4.0
7.0
72.0
149.0
10.0
2.0
59.0
69.0
17.0
7.0
1.0
42.0
15.0
16.0
70.0
99.0
2.0
40.0
119.0
12.0
2.0
80.0
3.0
69.0
17.0

- similar to the screen size observations, many brands offer a selfie cam with at least 8 mp. Huawei and Samsung are the leaders here as well.

EDA

- It is a good idea to explore the data once again after manipulating it.

In [37]:
```
data.describe()
```

Out[37]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	norm
count	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	
mean	13.713115	9.384534	6.553329	54.546957	4.036080	3133.170961	182.705761	2015.965258	674.869716	
std	3.805280	4.700148	6.968453	84.926611	1.364314	1298.565062	88.329494	2.298455	248.580166	
min	5.080000	0.080000	0.000000	0.010000	0.020000	500.000000	69.000000	2013.000000	91.000000	
25%	12.700000	5.000000	2.000000	16.000000	4.000000	2100.000000	142.000000	2014.000000	533.500000	
50%	12.830000	8.000000	5.000000	32.000000	4.000000	3000.000000	160.000000	2015.500000	690.500000	
75%	15.340000	13.000000	8.000000	64.000000	4.000000	4000.000000	185.000000	2018.000000	868.750000	
max	30.710000	48.000000	32.000000	1024.000000	12.000000	9720.000000	855.000000	2020.000000	1094.000000	

In [38]:
```
data.shape
```

Out[38]:
```
(3454, 15)
```

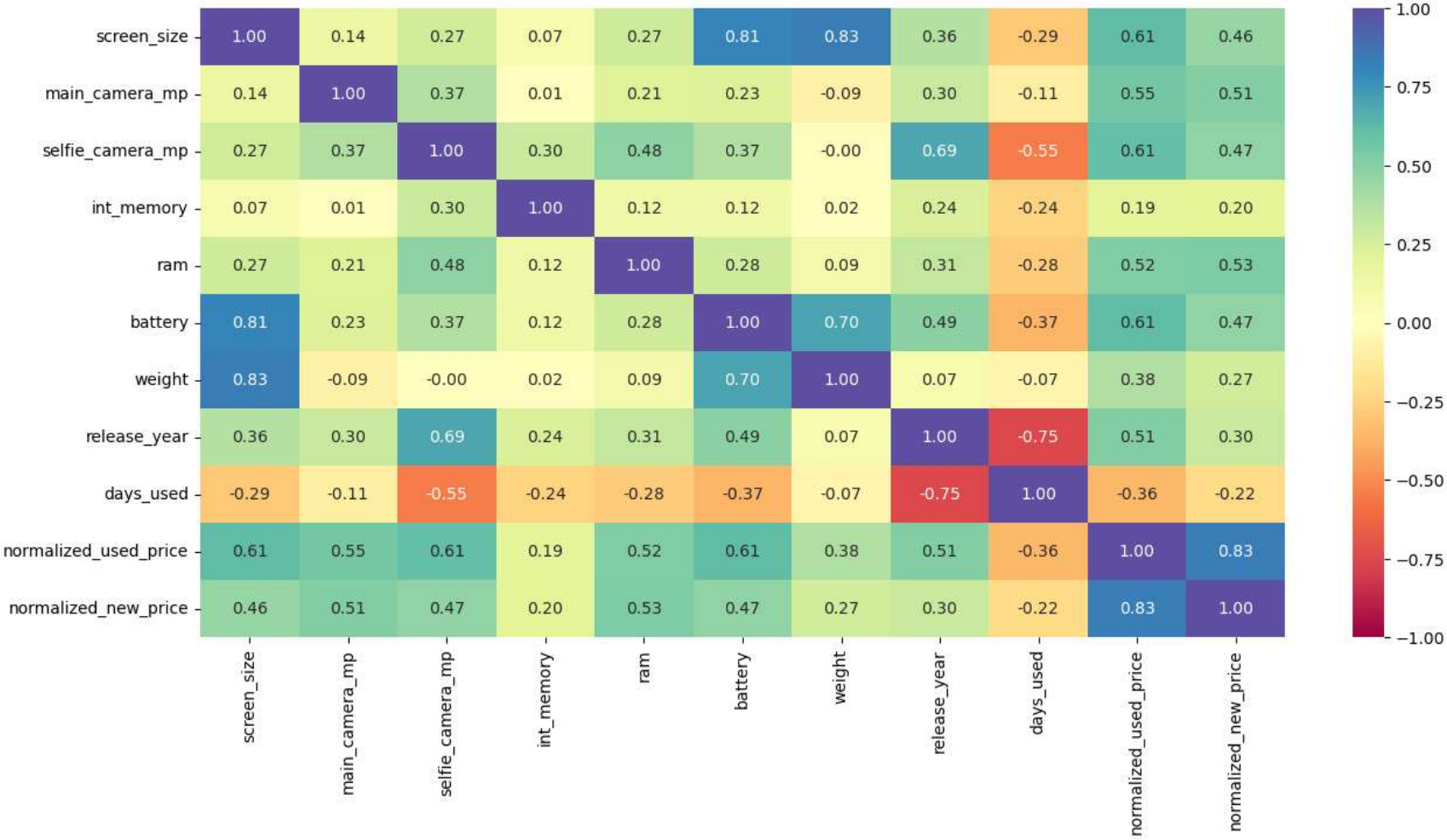
In [39]:
```
data1 = data.copy()
```

Bivariate Analysis

In [40]:
```
# correlation of all attributes with normalized_used_price
data[data.columns[:]].corr()["normalized_used_price"][:]
```

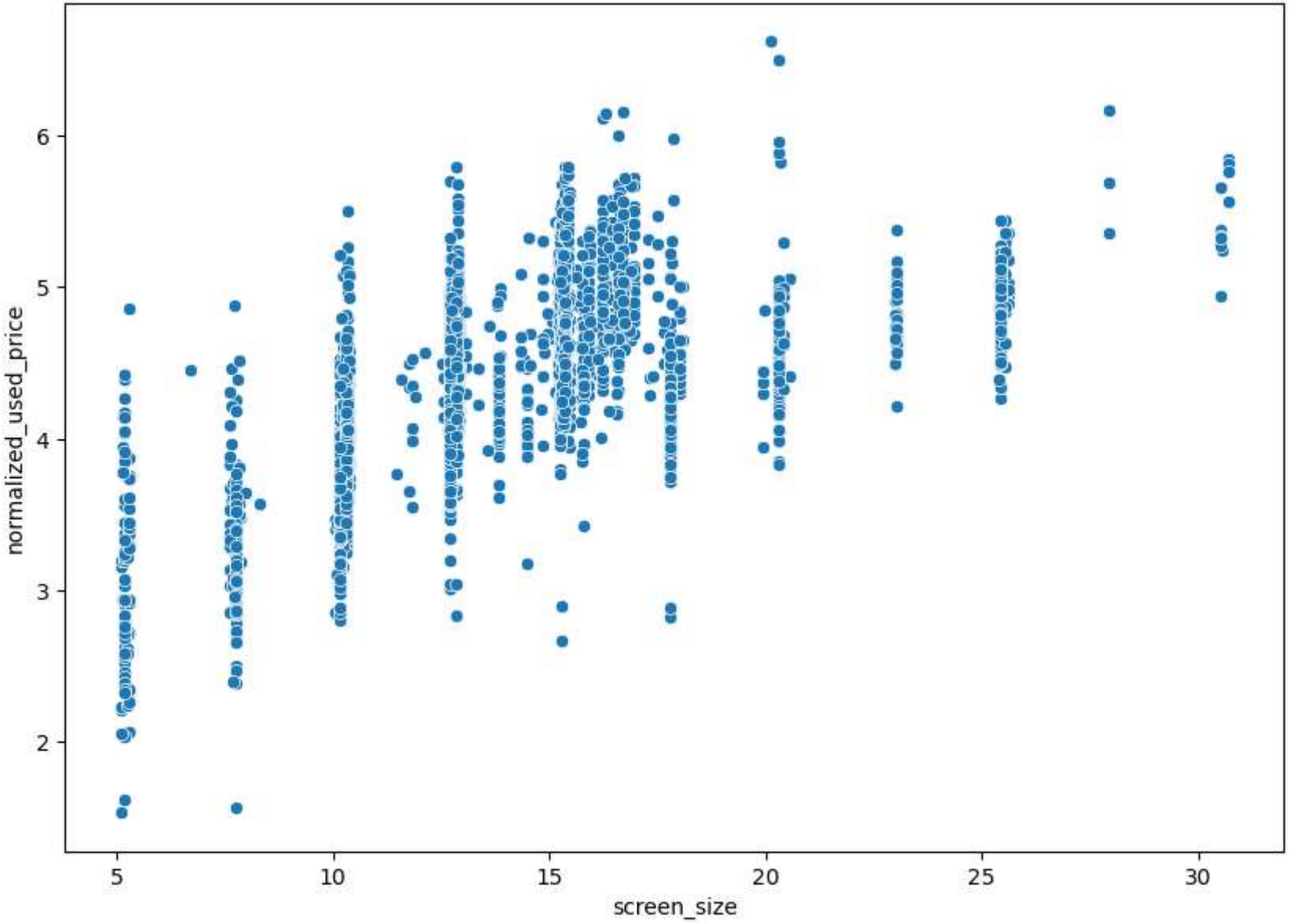
Out[40]:
```
screen_size              0.614785
main_camera_mp           0.552477
selfie_camera_mp         0.607548
int_memory               0.190954
ram                      0.518783
battery                  0.612041
weight                   0.382456
release_year             0.509790
days_used               -0.358264
normalized_used_price    1.000000
normalized_new_price     0.834496
Name: normalized_used_price, dtype: float64
```

In [41]:
```
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()

# correlation heatmap
plt.figure(figsize=(15, 7))
sns.heatmap(
    data[numeric_columns].corr(),
    annot=True,
    vmin=-1,
    vmax=1,
    fmt=".2f",
    cmap="Spectral",
)
plt.show()
```



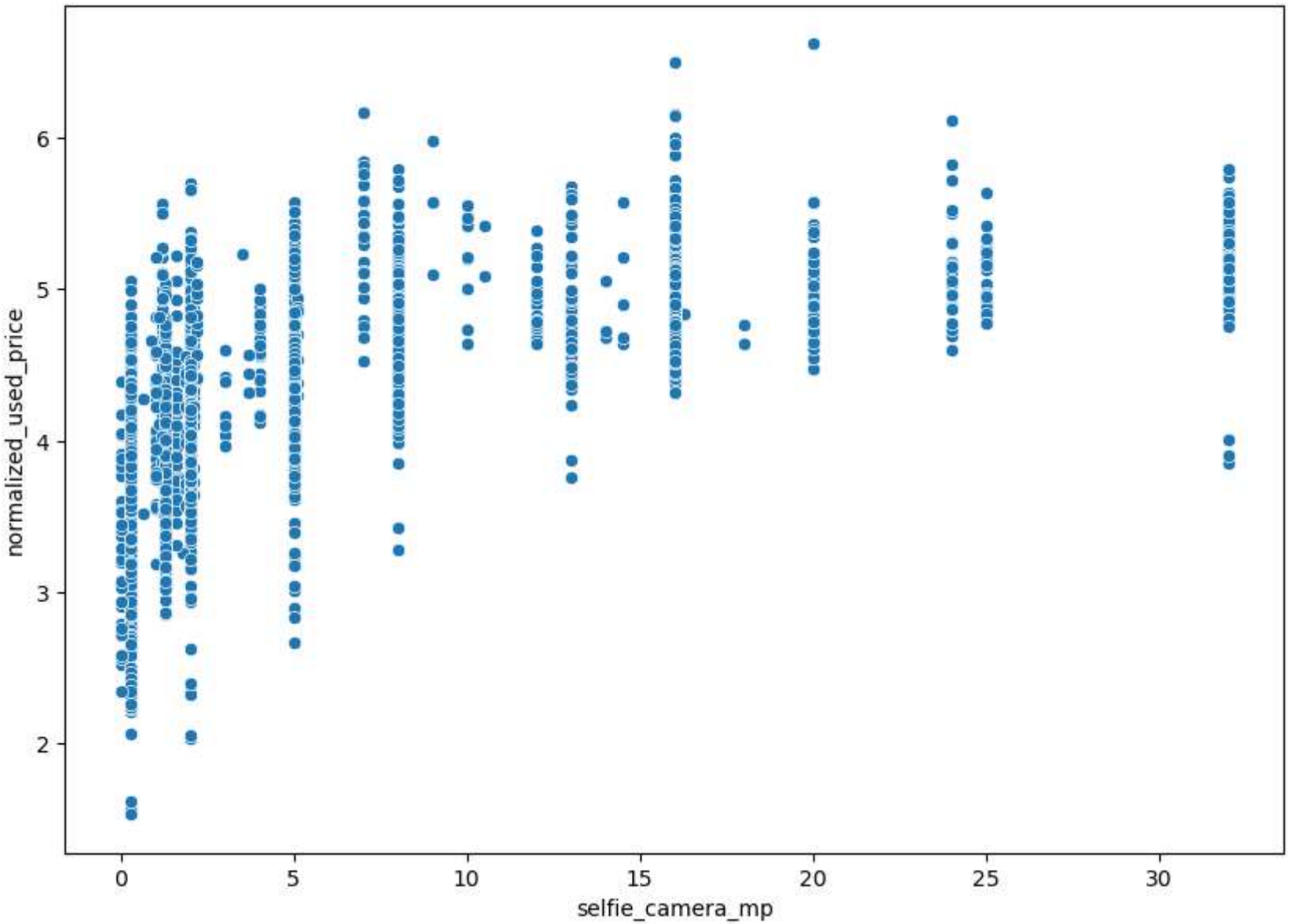
- screen size, battery and weight all have high correlations

In [43]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="normalized_used_price", x="screen_size",  data=data)
plt.show()
```



- screen size appears to be a driver of used price

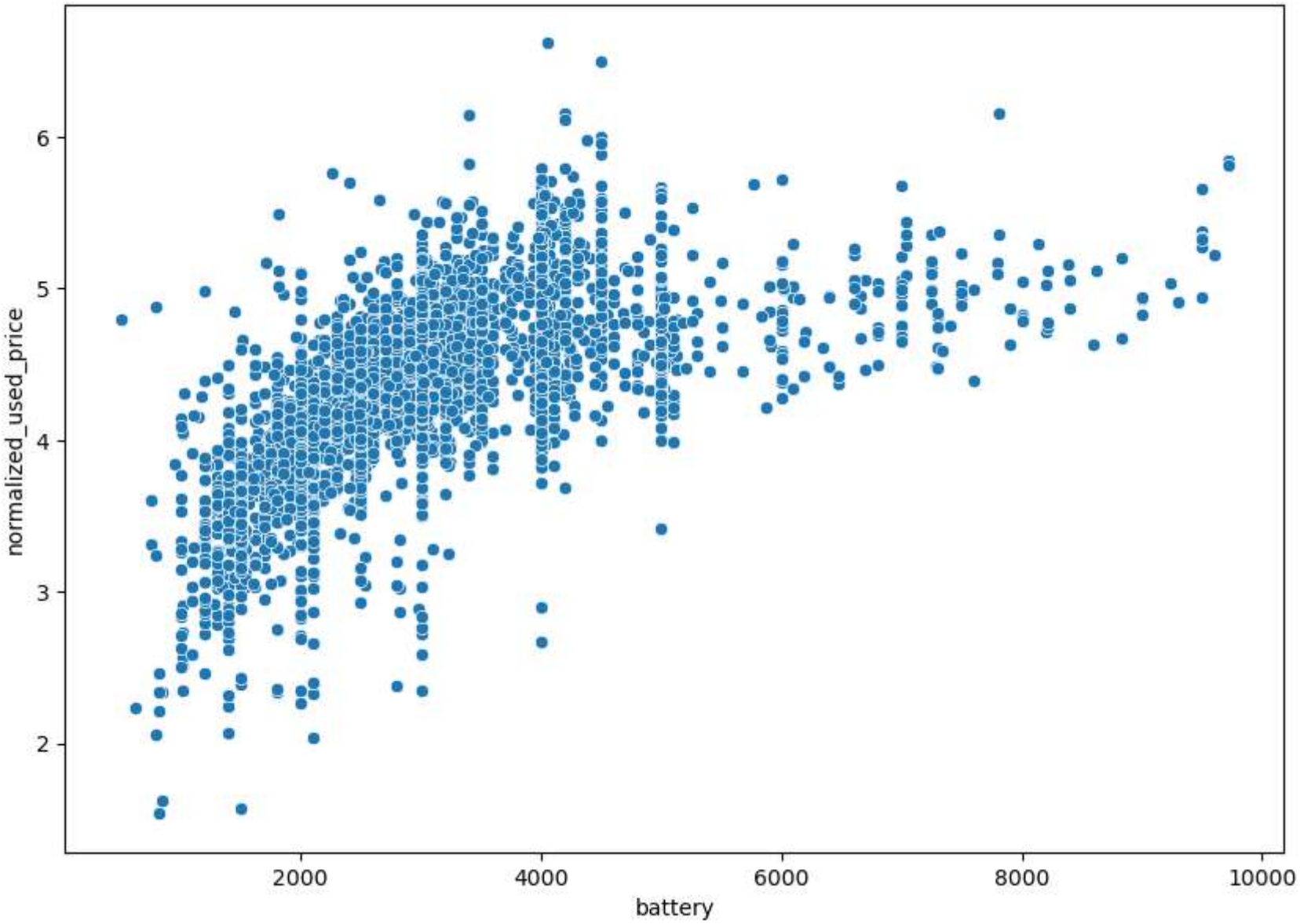
In [44]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="normalized_used_price", x="selfie_camera_mp",  data=data)
plt.show()
```



- the mega pixel offering for a selfie camera feature is a positive driver of used price.

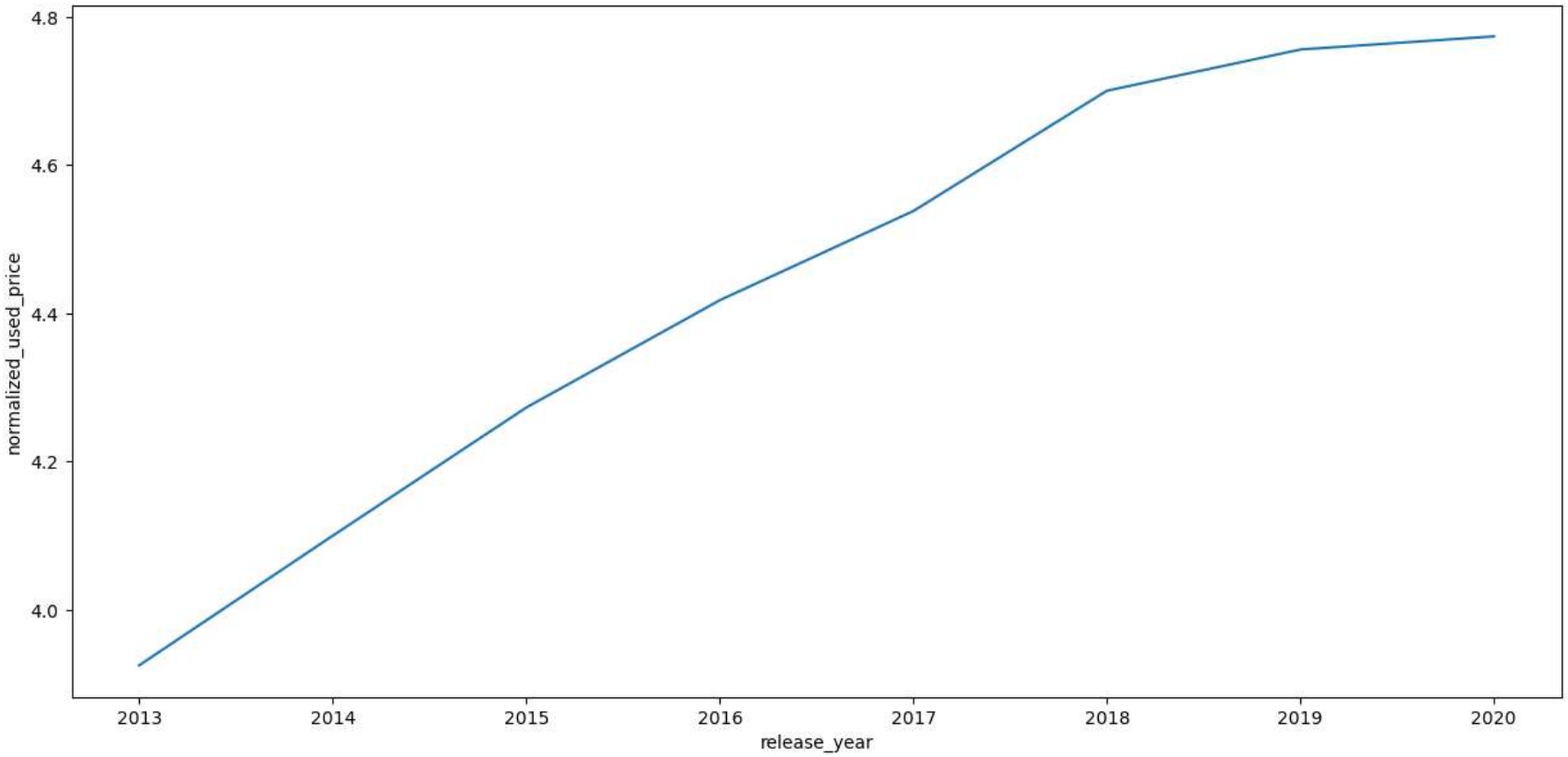
In [45]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(y="normalized_used_price", x="battery",  data=data)
```

```
plt.show()
```



- the more battery capacity leads to a higher used price, generally

In [46]:
```
# average life expectancy over the years
plt.figure(figsize=(15, 7))
sns.lineplot(x="release_year", y="normalized_used_price", data=data, ci=None)
plt.show()
```



- a more recently released phone will command higher used prices

Model Building - Linear Regression

1. We want to predict the used cell phone prices.

2. Before we proceed to build a model, we'll have to encode categorical features.

3. We'll split the data into train and test to be able to evaluate the model that we build on the train data.

4. We will build a Linear Regression model using the train data and then check it's performance.

In [48]:
```
# define the X ad y variables.
X = data.drop(["normalized_used_price"], axis=1)
y = data["normalized_used_price"]

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

```
  brand_name       os  screen_size   4g   5g  main_camera_mp  \
0      Honor  Android        14.50  yes   no            13.0
1      Honor  Android        17.30  yes  yes            13.0
2      Honor  Android        16.69  yes  yes            13.0
3      Honor  Android        25.50  yes  yes            13.0
4      Honor  Android        15.32  yes   no            13.0

   selfie_camera_mp  int_memory  ram  battery  weight  release_year  \
0               5.0        64.0  3.0   3020.0   146.0          2020
1              16.0       128.0  8.0   4300.0   213.0          2020
2               8.0       128.0  8.0   4200.0   213.0          2020
3               8.0        64.0  6.0   7250.0   480.0          2020
4               8.0        64.0  3.0   5000.0   185.0          2020

   days_used  normalized_new_price
0        127              4.715100
1        325              5.519018
2        162              5.884631
3        345              5.630961
4        293              4.947837
0    4.307572
1    5.162097
2    5.111084
3    5.135387
4    4.389995
Name: normalized_used_price, dtype: float64
```

In [49]:
```
# add constant which is the y-intercept when X-values are all 0.
X = sm.add_constant(X)
```

In [50]:
```
# get dummy values for categorical variables
X = pd.get_dummies(
    X,
    columns=X.select_dtypes(include=["category"]).columns.tolist(),
    drop_first=True,
)
X.head()
```

Out[50]:

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	...	brand_name_Spice
0	1.0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	2020	127	...	0
1	1.0	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	2020	325	...	0
2	1.0	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	2020	162	...	0
3	1.0	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	2020	345	...	0
4	1.0	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	2020	293	...	0

5 rows × 49 columns

In [51]:
```
# 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, random_state=1)
```

In [52]:
```
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
```

In [53]:
```
olsmodel = sm.OLS(y_train, x_train).fit()
print(olsmodel.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:	Fri, 10 Feb 2023	Prob (F-statistic):	0.00
Time:	18:13:29	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	-66.707	-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	-2.92e-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	-3.01e-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	-2.58e-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):              409.702
Skew:                          -0.607   Prob(JB):                     1.08e-89
Kurtosis:                       4.611   Cond. No.                     7.69e+06
==============================================================================

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

Adjusted R-squared = .842 which is a very good value.

Const coefficient = -48.6977 This is the y-intercept

To check performance of the model using different metrics:

- We will be using metric functions defined in sklearn for RMSE, MAE, and R^2.

In [123…

```
x_train1 = sm.add_constant(x_train)
# adding constant to the test data
x_test1 = sm.add_constant(x_test)
```

```
olsmod0 = sm.OLS(y_train, x_train1).fit()
print(olsmod0.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.840
Model:                               OLS   Adj. R-squared:                  0.839
Method:                    Least Squares   F-statistic:                     1264.
Date:                   Thu, 09 Feb 2023   Prob (F-statistic):               0.00
Time:                           08:52:11   Log-Likelihood:                 87.447
No. Observations:                   2417   AIC:                            -152.9
Df Residuals:                       2406   BIC:                            -89.20
Df Model:                             10
Covariance Type:               nonrobust
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                -61.9333      8.123     -7.624      0.000     -77.862     -46.004
screen_size            0.0277      0.003      9.065      0.000       0.022       0.034
main_camera_mp         0.0219      0.001     16.484      0.000       0.019       0.025
selfie_camera_mp       0.0138      0.001     12.865      0.000       0.012       0.016
int_memory          5.542e-05   6.61e-05      0.839      0.402   -7.41e-05       0.000
ram                    0.0214      0.004      4.944      0.000       0.013       0.030
battery             -8.689e-06   7.02e-06     -1.238      0.216   -2.25e-05    5.07e-06
weight                 0.0008      0.000      6.377      0.000       0.001       0.001
release_year           0.0313      0.004      7.783      0.000       0.023       0.039
days_used           5.525e-05   3.02e-05      1.830      0.067   -3.96e-06       0.000
normalized_new_price   0.4263      0.011     39.852      0.000       0.405       0.447
==============================================================================
Omnibus:                      228.894   Durbin-Watson:                   1.909
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              409.880
Skew:                          -0.649   Prob(JB):                     9.90e-90
Kurtosis:                       4.544   Cond. No.                     6.75e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.75e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Model Performance Check

In [54]:
```
# 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 regression model
def model_performance_regression(model, predictors, target):

    """
    Function to compute different metrics to check regression model performance
    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 R-squared
    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 [55]:

```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
```

```
olsmodel_train_perf = model_performance_regression(olsmodel, x_train, y_train)
olsmodel_train_perf
```

```
Training Performance
```

Out[55]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.229849	0.180336	0.844933	0.841723	4.326958

In [56]:

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

```
Test Performance
```

Out[56]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.238306	0.184064	0.842547	0.834731	4.488006

- RMSE and MAE are comparable, so model is not overfitting.
- The training R^2 is .84 indicating that the model is not underfitting.
- MAE suggests the model can predict used prices withing a mean error of .18 Euros on the test data.
- MAPE of 4.48 on the test data means that we can predict within 4.3% of the used price.

Checking Linear Regression Assumptions

In order to make statistical inferences from a linear regression model, it is important to ensure that the assumptions of linear regression are satisfied.

1. No Multicollinearity
2. Linearity of variables
3. Independence of error terms

4. Normality of error terms
5. No Heteroscedasticity

In [57]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
# we will 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 [58]:
```
checking_vif(x_train)
```

Out[58]:

	feature	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

	feature	VIF
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
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

- weight and screen size have VIF's above 5.
- several brands have VIF's above 5 also, so we will consider those as well.

Removing Multicolinearity

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.

In [70]:
```
def treating_multicollinearity(predictors, target, high_vif_columns):

    # 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 previously
    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 [71]:
```
col_list = [
    "screen_size",
    "weight",
    "brand_name_Huawei",
    "brand_name_Others",
    "brand_name_Samsung",
]
res = treating_multicollinearity(x_train, y_train, col_list)
res
```

Out[71]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	brand_name_Huawei	0.841855	0.232167
1	brand_name_Others	0.841854	0.232167
2	brand_name_Samsung	0.841819	0.232193
3	screen_size	0.838448	0.234655
4	weight	0.838179	0.234849

In [72]:
```
col_to_drop = "brand_name_Huawei"
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)]
# Check VIF now
vif = checking_vif(x_train2)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  brand_name_Huawei
```

Out[72]:

	feature	VIF
0	const	3.779913e+06
1	screen_size	7.677046e+00
2	main_camera_mp	2.135138e+00
3	selfie_camera_mp	2.805050e+00
4	int_memory	1.359744e+00
5	ram	2.258262e+00
6	battery	4.072488e+00
7	weight	6.380453e+00
8	release_year	4.883605e+00
9	days_used	2.668551e+00
10	normalized_new_price	3.121828e+00
11	brand_name_Alcatel	1.438997e+00
12	brand_name_Apple	1.217760e+01
13	brand_name_Asus	1.370902e+00
14	brand_name_BlackBerry	1.166939e+00
15	brand_name_Celkon	1.256707e+00
16	brand_name_Coolpad	1.079905e+00
17	brand_name_Gionee	1.170078e+00
18	brand_name_Google	1.061353e+00
19	brand_name_HTC	1.402002e+00
20	brand_name_Honor	1.361580e+00
21	brand_name_Infinix	1.070717e+00
22	brand_name_Karbonn	1.136048e+00
23	brand_name_LG	1.618771e+00
24	brand_name_Lava	1.144896e+00

	feature	VIF
25	brand_name_Lenovo	1.567687e+00
26	brand_name_Meizu	1.179936e+00
27	brand_name_Micromax	1.481369e+00
28	brand_name_Microsoft	1.541199e+00
29	brand_name_Motorola	1.368001e+00
30	brand_name_Nokia	1.722214e+00
31	brand_name_OnePlus	1.086426e+00
32	brand_name_Oppo	1.467503e+00
33	brand_name_Others	2.449072e+00
34	brand_name_Panasonic	1.188016e+00
35	brand_name_Realme	1.186674e+00
36	brand_name_Samsung	2.021488e+00
37	brand_name_Sony	1.334330e+00
38	brand_name_Spice	1.163222e+00
39	brand_name_Vivo	1.399729e+00
40	brand_name_XOLO	1.238754e+00
41	brand_name_Xiaomi	1.407521e+00
42	brand_name_ZTE	1.450721e+00
43	os_Others	1.855246e+00
44	os_Windows	1.594642e+00
45	os_iOS	1.178455e+01
46	4g_yes	2.459505e+00
47	5g_yes	1.844777e+00

- dropping brand name Huawei had very little impact on screen size or weight. The VIF for Samsung and Others did drop below VIF of 5

In [87]:
```
# drop screen_size
col_list = ["screen_size", "weight", "brand_name_Others", "brand_name_Samsung"]
res = treating_multicollinearity(x_train, y_train, col_list)
res
```

Out[87]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	weight	0.838223	0.234818
1	brand_name_Samsung	0.838223	0.234818
2	brand_name_Others	0.838104	0.234905
3	screen_size	0.815617	0.250687

In [91]:
```
# dropping screen_size column
col_to_drop = "screen_size"
x_train3 = x_train.loc[:, ~x_train.columns.str.startswith(col_to_drop)]
x_test3 = x_test.loc[:, ~x_test.columns.str.startswith(col_to_drop)]
# Check VIF now
vif = checking_vif(x_train3)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  screen_size
```

Out[91]:

	feature	VIF
0	const	3.629142e+06
1	main_camera_mp	2.000214e+00
2	selfie_camera_mp	2.775739e+00
3	int_memory	1.355343e+00
4	ram	2.253438e+00
5	battery	1.752100e+00
6	release_year	4.684848e+00
7	days_used	2.654363e+00
8	normalized_new_price	2.985261e+00
9	brand_name_Alcatel	1.318249e+00
10	brand_name_Apple	1.199618e+01
11	brand_name_Asus	1.260466e+00
12	brand_name_BlackBerry	1.127022e+00
13	brand_name_Celkon	1.231304e+00
14	brand_name_Coolpad	1.066020e+00
15	brand_name_Gionee	1.110913e+00
16	brand_name_Google	1.045337e+00
17	brand_name_HTC	1.282108e+00
18	brand_name_Honor	1.342304e+00
19	brand_name_Huawei	1.602878e+00
20	brand_name_Infinix	1.066050e+00
21	brand_name_Karbonn	1.114088e+00
22	brand_name_LG	1.431777e+00
23	brand_name_Lava	1.118744e+00
24	brand_name_Lenovo	1.431433e+00

	feature	VIF
25	brand_name_Meizu	1.155118e+00
26	brand_name_Micromax	1.386124e+00
27	brand_name_Microsoft	1.520908e+00
28	brand_name_Motorola	1.322643e+00
29	brand_name_Nokia	1.593822e+00
30	brand_name_OnePlus	1.077418e+00
31	brand_name_Oppo	1.422327e+00
32	brand_name_Others	2.032078e+00
33	brand_name_Panasonic	1.148504e+00
34	brand_name_Realme	1.180594e+00
35	brand_name_Sony	1.249905e+00
36	brand_name_Spice	1.132742e+00
37	brand_name_Vivo	1.366106e+00
38	brand_name_XOLO	1.187419e+00
39	brand_name_Xiaomi	1.374490e+00
40	brand_name_ZTE	1.355196e+00
41	os_Others	1.615736e+00
42	os_Windows	1.594287e+00
43	os_iOS	1.167152e+01
44	4g_yes	2.393566e+00
45	5g_yes	1.841128e+00

- all VIF values are now under 5

In [92]:
```
olsmod = sm.OLS(y_train, x_train3).fit()
print(olsmod.summary())
```

OLS Regression Results

Dep. Variable:	normalized_used_price	R-squared:	0.819
Model:	OLS	Adj. R-squared:	0.816
Method:	Least Squares	F-statistic:	238.5
Date:	Fri, 10 Feb 2023	Prob (F-statistic):	0.00
Time:	20:51:26	Log-Likelihood:	-62.317
No. Observations:	2417	AIC:	216.6
Df Residuals:	2371	BIC:	483.0
Df Model:	45		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-47.0275	9.714	-4.841	0.000	-66.076	-27.979
main_camera_mp	0.0139	0.002	9.057	0.000	0.011	0.017
selfie_camera_mp	0.0118	0.001	9.756	0.000	0.009	0.014
int_memory	1.817e-05	7.51e-05	0.242	0.809	-0.000	0.000
ram	0.0197	0.006	3.549	0.000	0.009	0.031
battery	9.29e-05	5.15e-06	18.055	0.000	8.28e-05	0.000
release_year	0.0239	0.005	4.977	0.000	0.015	0.033
days_used	3.893e-05	3.33e-05	1.170	0.242	-2.63e-05	0.000
normalized_new_price	0.4777	0.013	36.887	0.000	0.452	0.503
brand_name_Alcatel	0.0749	0.032	2.343	0.019	0.012	0.138
brand_name_Apple	0.2314	0.152	1.521	0.128	-0.067	0.530
brand_name_Asus	0.0585	0.032	1.838	0.066	-0.004	0.121
brand_name_BlackBerry	-0.0148	0.063	-0.234	0.815	-0.138	0.109
brand_name_Celkon	0.0218	0.060	0.367	0.714	-0.095	0.139
brand_name_Coolpad	0.0772	0.067	1.151	0.250	-0.054	0.209
brand_name_Gionee	-0.0169	0.047	-0.359	0.719	-0.109	0.075
brand_name_Google	-0.0490	0.081	-0.603	0.546	-0.208	0.110
brand_name_HTC	0.0042	0.032	0.131	0.895	-0.058	0.067
brand_name_Honor	0.0670	0.034	1.992	0.047	0.001	0.133
brand_name_Huawei	0.0312	0.025	1.256	0.209	-0.018	0.080
brand_name_Infinix	0.0811	0.092	0.884	0.377	-0.099	0.261
brand_name_Karbonn	0.1577	0.061	2.588	0.010	0.038	0.277
brand_name_LG	-0.0141	0.027	-0.531	0.596	-0.066	0.038
brand_name_Lava	0.0480	0.054	0.882	0.378	-0.059	0.155
brand_name_Lenovo	0.0813	0.027	2.973	0.003	0.028	0.135
brand_name_Meizu	0.0225	0.044	0.511	0.610	-0.064	0.109
brand_name_Micromax	-0.0208	0.033	-0.629	0.530	-0.086	0.044
brand_name_Microsoft	0.1583	0.086	1.841	0.066	-0.010	0.327
brand_name_Motorola	0.0266	0.034	0.782	0.434	-0.040	0.093
brand_name_Nokia	0.0872	0.038	2.302	0.021	0.013	0.161
brand_name_OnePlus	0.1248	0.072	1.724	0.085	-0.017	0.267

```
brand_name_Oppo           0.0290      0.031      0.940      0.347      -0.031       0.089
brand_name_Others         0.0241      0.021      1.158      0.247      -0.017       0.065
brand_name_Panasonic      0.0785      0.045      1.764      0.078      -0.009       0.166
brand_name_Realme         0.0901      0.052      1.740      0.082      -0.011       0.192
brand_name_Sony          -0.0069      0.036     -0.195      0.845      -0.077       0.063
brand_name_Spice         -0.0004      0.056     -0.006      0.995      -0.110       0.109
brand_name_Vivo           0.0085      0.032      0.267      0.789      -0.054       0.071
brand_name_XOLO           0.0181      0.044      0.410      0.682      -0.068       0.105
brand_name_Xiaomi         0.1016      0.032      3.219      0.001       0.040       0.164
brand_name_ZTE            0.0203      0.031      0.664      0.507      -0.040       0.080
os_Others                -0.1464      0.033     -4.435      0.000      -0.211      -0.082
os_Windows               -0.0096      0.049     -0.198      0.843      -0.105       0.086
os_iOS                   -0.2234      0.157     -1.420      0.156      -0.532       0.085
4g_yes                    0.0023      0.017      0.138      0.890      -0.031       0.035
5g_yes                   -0.0604      0.034     -1.764      0.078      -0.128       0.007
==============================================================================
Omnibus:                      156.577   Durbin-Watson:                   1.918
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              270.912
Skew:                          -0.487   Prob(JB):                     1.49e-59
Kurtosis:                       4.319   Cond. No.                     7.52e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.52e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

- adjusted R^2 is .816 which is a good value

In [95]:

```python
# creating a list of columns with p-values greater than 0.05
#initial list of columns
cols = x_train3.columns.tolist()
# setting an initial max p-value
max_p_value = 1
while len(cols) > 0:
 # defining the train set
    x_train_aux = x_train3[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', 'main_camera_mp', 'selfie_camera_mp', 'ram', 'battery', 'release_year', 'normalized_new_price', 'brand_name_K
arbonn', 'brand_name_Lenovo', 'brand_name_Xiaomi', 'os_Others']
```

In [96]:
```
x_train4 = x_train3[["const"] + selected_features]
x_test4 = x_test3[["const"] + selected_features]
```

In [97]:
```
olsmod2 = sm.OLS(y_train, x_train4).fit()
print(olsmod2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.816
Model:                               OLS   Adj. R-squared:                  0.815
Method:                    Least Squares   F-statistic:                     1068.
Date:                   Fri, 10 Feb 2023   Prob (F-statistic):               0.00
Time:                           20:57:35   Log-Likelihood:                -81.485
No. Observations:                   2417   AIC:                             185.0
Df Residuals:                       2406   BIC:                             248.7
Df Model:                             10
Covariance Type:               nonrobust
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                  -22.5463      3.360     -6.709      0.000     -29.136     -15.956
const                  -22.5463      3.360     -6.709      0.000     -29.136     -15.956
main_camera_mp           0.0147      0.001     11.101      0.000       0.012       0.017
selfie_camera_mp         0.0115      0.001     10.193      0.000       0.009       0.014
ram                      0.0146      0.005      3.085      0.002       0.005       0.024
battery               9.378e-05   4.95e-06     18.939      0.000    8.41e-05       0.000
release_year             0.0230      0.003      6.907      0.000       0.016       0.030
normalized_new_price     0.4709      0.011     42.268      0.000       0.449       0.493
brand_name_Karbonn       0.1324      0.058      2.270      0.023       0.018       0.247
brand_name_Lenovo        0.0580      0.023      2.499      0.013       0.012       0.103
brand_name_Xiaomi        0.0737      0.027      2.682      0.007       0.020       0.128
os_Others               -0.1394      0.029     -4.837      0.000      -0.196      -0.083
==============================================================================
Omnibus:                      164.060   Durbin-Watson:                   1.918
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              294.411
Skew:                          -0.494   Prob(JB):                     1.17e-64
Kurtosis:                       4.395   Cond. No.                     3.52e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.96e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

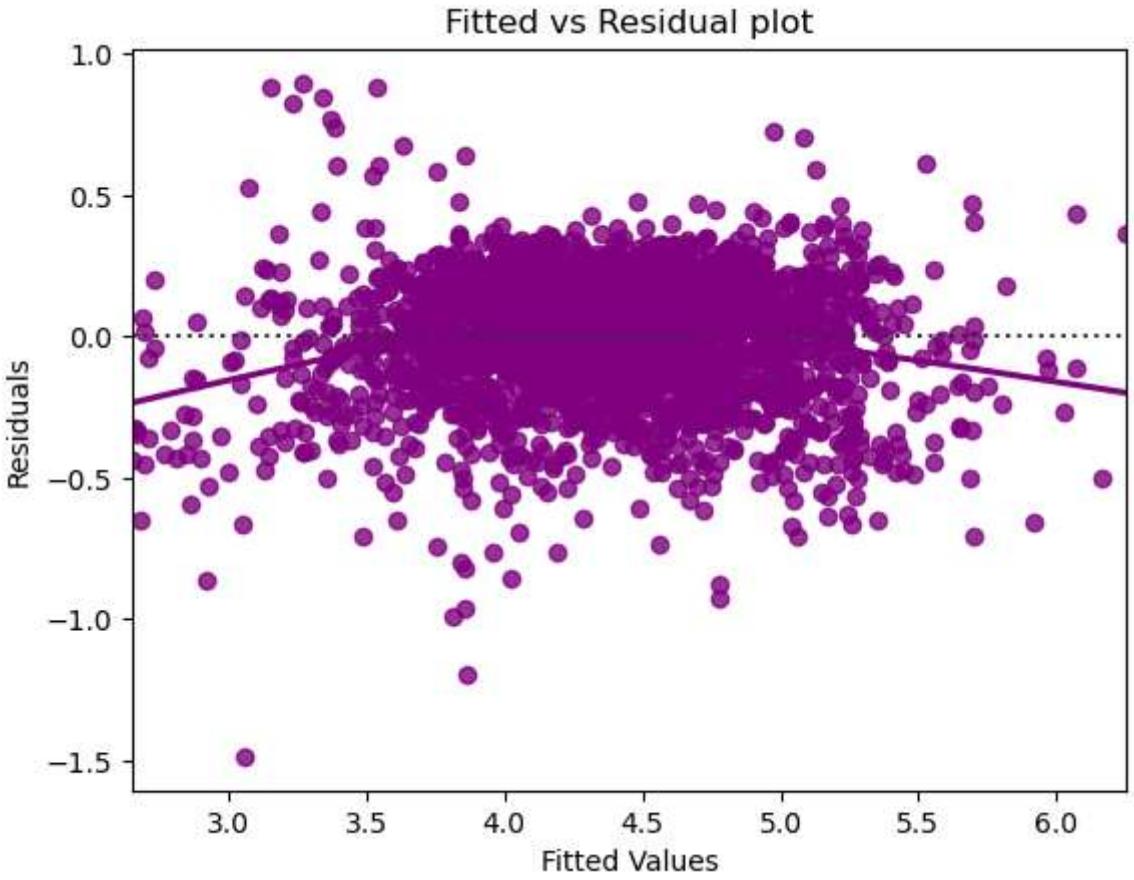
- the model has now been updated with no p-values greater than 0.05
- x_train4 will be considered the final set of predictor variables and olsmod2 as the final model
- R^2 is .815 which means our model explains about 81.5% of the variance.

In [98]:
```
# 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"] = olsmod2.fittedvalues  # predicted values
df_pred["Residuals"] = olsmod2.resid  # residuals
df_pred.head()
```

Out[98]:

	Actual Values	Fitted Values	Residuals
3026	4.087488	3.897407	0.190081
1525	4.448399	4.640640	-0.192241
1128	4.315353	4.279292	0.036061
3003	4.282068	4.290786	-0.008717
2907	4.456438	4.427247	0.029191

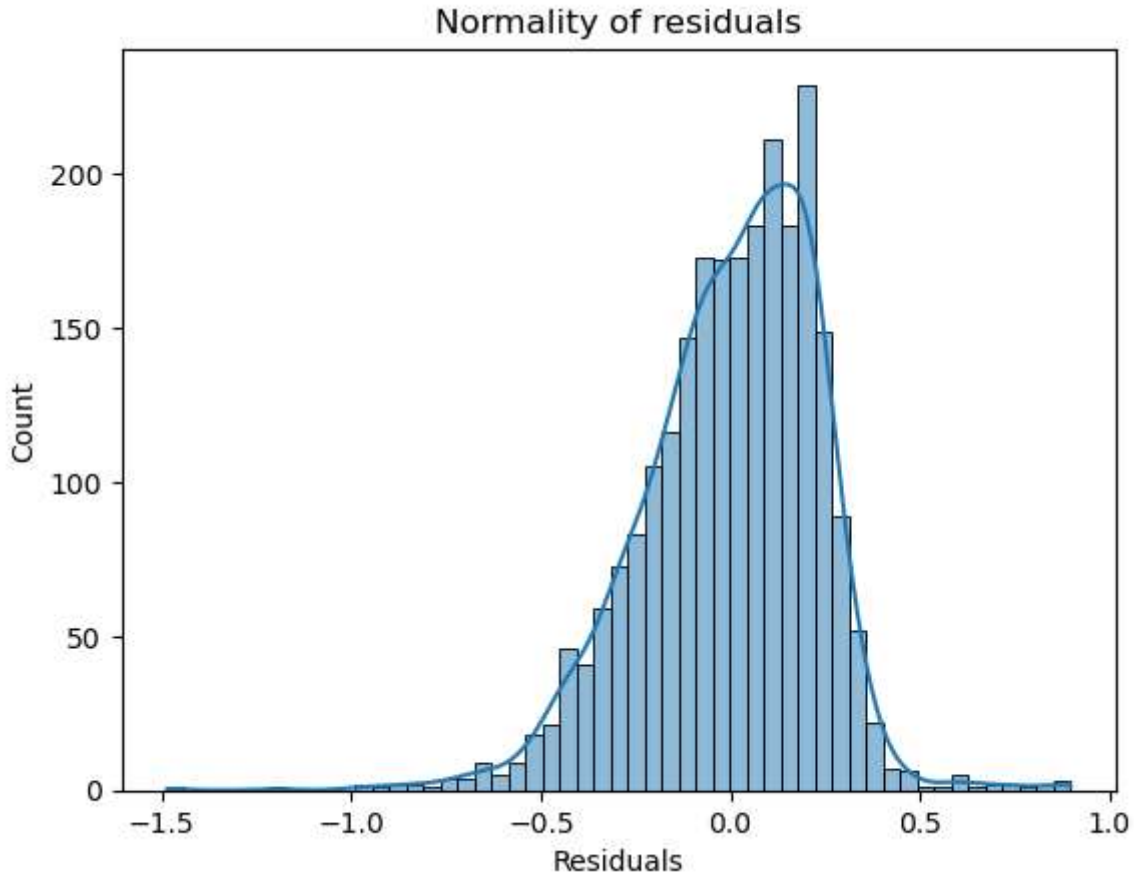
In [259…
```
# let's plot the fitted values vs residuals
sns.residplot(
    data=df_pred, x="Fitted Values", y="Residuals", color="purple", lowess=True
)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Fitted vs Residual plot")
plt.show()
```



- no pattern so the assumption of linearity and independence are satisified

In [260…

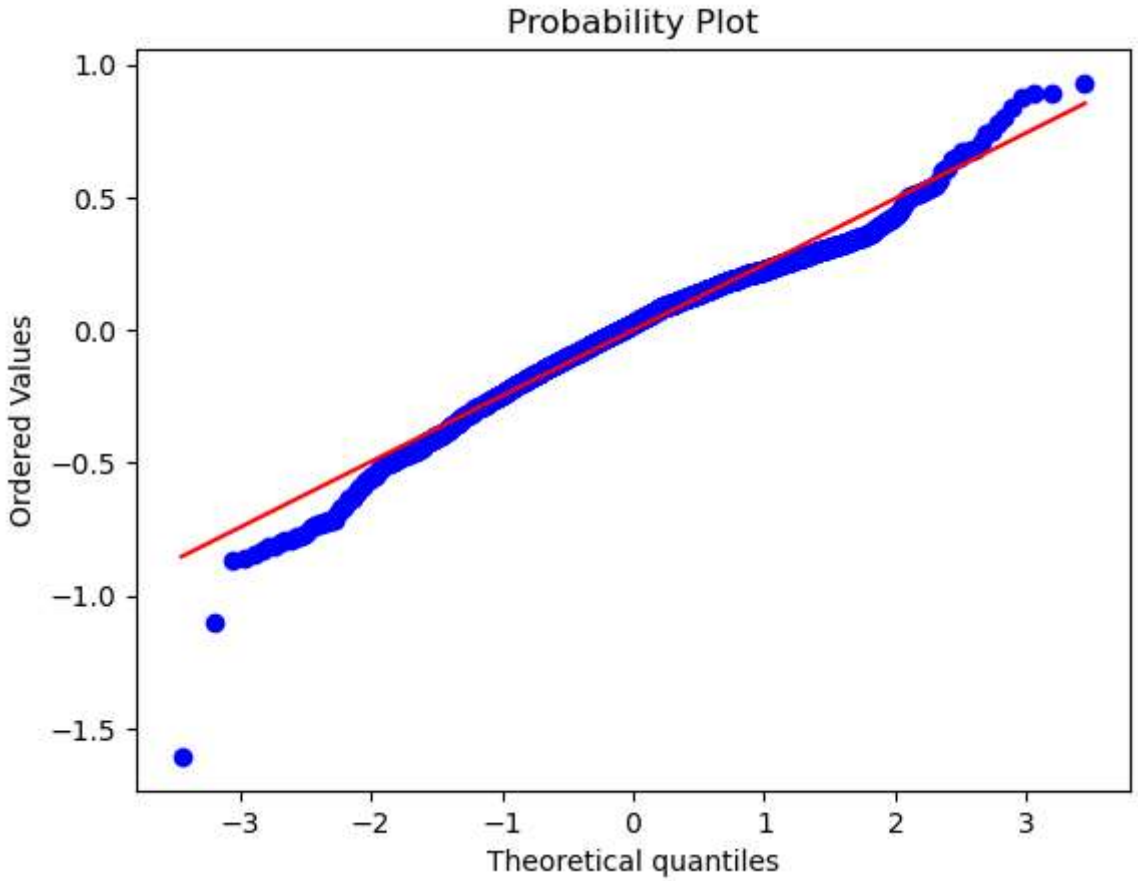
```
#test for normality
sns.histplot(data=df_pred, x="Residuals", kde=True)
plt.title("Normality of residuals")
plt.show()
```



- the histogram of residuals is approxmately bell shaped

In [100…

```
# Q-Q plot
import pylab
import scipy.stats as stats

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



- the residuals for the Q-Q plot follow approximately a straight line

In [101…
```
stats.shapiro(df_pred["Residuals"])
```

Out[101]:
```
ShapiroResult(statistic=0.978392481803894, pvalue=1.084487136206529e-18)
```

- the p-value is < 0.05 so the residuals are not normal per Shapiro-Wilk test.
- the assumption is satisfied as we will accept the distribution as being normal.

In [103…
```
# Heteroscedasticity check
import statsmodels.stats.api as sms
```

```
from statsmodels.compat import lzip
name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(df_pred["Residuals"], x_train4)
lzip(name, test)
```

Out[103]:
```
[('F statistic', 0.994833800284226), ('p-value', 0.5356945965994081)]
```

- the p-value is > 0.05 so the residuals are homoscedastic. Thus, the assumption is satisfied.

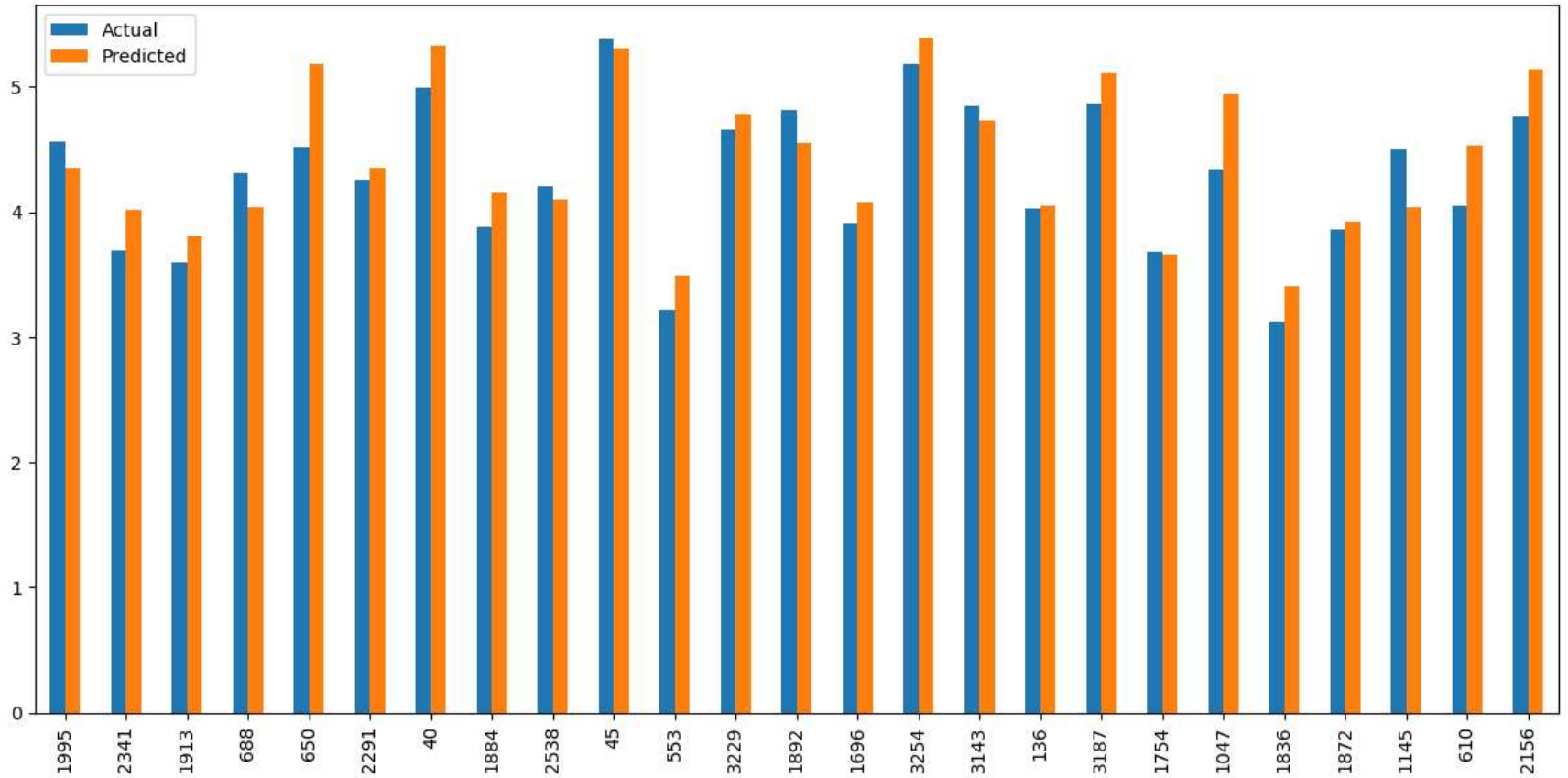
Final Model

In [104…
```
# predictions on the test set
pred = olsmod2.predict(x_test4)
df_pred_test = pd.DataFrame({"Actual": y_test, "Predicted": pred})
df_pred_test.sample(10, random_state=1)
```

Out[104]:

	Actual	Predicted
1995	4.566741	4.354627
2341	3.696103	4.012544
1913	3.592093	3.806089
688	4.306495	4.038043
650	4.522115	5.182079
2291	4.259294	4.354096
40	4.997685	5.332230
1884	3.875359	4.153752
2538	4.206631	4.105149
45	5.380450	5.312347

In [105…
```
df1 = df_pred_test.sample(25, random_state=1)
df1.plot(kind="bar", figsize=(15, 7))
plt.show()
```



In [106…

```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmod2_train_perf = model_performance_regression(olsmod2, x_train4, y_train)
olsmod2_train_perf
```

```
Training Performance
```

Out[106]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.250267	0.194949	0.816159	0.815241	4.694953

In [107…

```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
```

```
olsmod2_test_perf = model_performance_regression(olsmod2, x_test4, y_test)
olsmod2_test_perf
```

```
Test Performance
```

Out[107]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.252815	0.195519	0.822792	0.820715	4.788388

In [108…

```
# training performance comparison
models_train_comp_df = pd.concat(
    [olsmodel_train_perf.T, olsmod2_train_perf.T],
    axis=1,
)
models_train_comp_df.columns = [
    "Linear Regression (all variables)",
    "Linear Regression (selected variables)",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[108]:

	Linear Regression (all variables)	Linear Regression (selected variables)
RMSE	0.229849	0.250267
MAE	0.180336	0.194949
R-squared	0.844933	0.816159
Adj. R-squared	0.841723	0.815241
MAPE	4.326958	4.694953

In [109…

```
# test performance comparison
models_test_comp_df = pd.concat(
    [olsmodel_test_perf.T, olsmod2_test_perf.T],
    axis=1,
)
models_test_comp_df.columns = [
    "Linear Regression (all variables)",
    "Linear Regression (selected variables)",
]
```

```
print("Test performance comparison:")
models_test_comp_df
```

```
Test performance comparison:
```

Out[109]:

	Linear Regression (all variables)	Linear Regression (selected variables)
RMSE	0.238306	0.252815
MAE	0.184064	0.195519
R-squared	0.842547	0.822792
Adj. R-squared	0.834731	0.820715
MAPE	4.488006	4.788388

In [110…

```
olsmodel_final = sm.OLS(y_train, x_train4).fit()
print(olsmodel_final.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.816
Model:                               OLS   Adj. R-squared:                  0.815
Method:                    Least Squares   F-statistic:                     1068.
Date:                   Fri, 10 Feb 2023   Prob (F-statistic):               0.00
Time:                           21:11:01   Log-Likelihood:                -81.485
No. Observations:                   2417   AIC:                             185.0
Df Residuals:                       2406   BIC:                             248.7
Df Model:                             10
Covariance Type:               nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                    -22.5463      3.360     -6.709      0.000     -29.136     -15.956
const                    -22.5463      3.360     -6.709      0.000     -29.136     -15.956
main_camera_mp             0.0147      0.001     11.101      0.000       0.012       0.017
selfie_camera_mp           0.0115      0.001     10.193      0.000       0.009       0.014
ram                        0.0146      0.005      3.085      0.002       0.005       0.024
battery                 9.378e-05   4.95e-06     18.939      0.000    8.41e-05       0.000
release_year               0.0230      0.003      6.907      0.000       0.016       0.030
normalized_new_price       0.4709      0.011     42.268      0.000       0.449       0.493
brand_name_Karbonn         0.1324      0.058      2.270      0.023       0.018       0.247
brand_name_Lenovo          0.0580      0.023      2.499      0.013       0.012       0.103
brand_name_Xiaomi          0.0737      0.027      2.682      0.007       0.020       0.128
os_Others                 -0.1394      0.029     -4.837      0.000      -0.196      -0.083
==============================================================================
Omnibus:                      164.060   Durbin-Watson:                   1.918
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              294.411
Skew:                          -0.494   Prob(JB):                     1.17e-64
Kurtosis:                       4.395   Cond. No.                     3.52e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.96e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Actionable Insights and Recommendations

- All variables have positive coefficients (except 'os_Others), so used prices will increase as they increase.

- A one unit increase in 'os_Others' can create a drop in price of .1394 Euros
- New phone prices can have a significant impact on the used phone market prices. A one unit increase in new phone prices can have a .47 Euro increase in used prices.
