ReCell Project

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 detailed data dictionary is given below.

Data Dictionary

- 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
- new_price: Price of a new device of the same model in euros
- used_price: Price of the used/refurbished device in euros

Importing necessary libraries and data

In [1]:
```
#import libraries needed for data manipulation

import numpy as np
import pandas as pd

pd.set_option('display.float_format', lambda x: '%.3f' % x)

#import libraries needed for data visualization

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# import libary if needed for probability distributions
import pylab
import scipy.stats as stats

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

# import functions needed to build and test linear regression model using sklearn

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# using statsmodels for linear regression model
import statsmodels.api as sm

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

Data Overview

- Observations
- Sanity checks

In [2]:
```
#import dataset named 'used_device_data.csv'

df = pd.read_csv('used_device_data.csv')
```

```
# read first five rows of the dataset

df.head()
```

Out[2]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	new_pric
0	Honor	Android	14.500	yes	no	13.000	5.000	64.000	3.000	3020.000	146.000	2020	127	111.62
1	Honor	Android	17.300	yes	yes	13.000	16.000	128.000	8.000	4300.000	213.000	2020	325	249.39
2	Honor	Android	16.690	yes	yes	13.000	8.000	128.000	8.000	4200.000	213.000	2020	162	359.47
3	Honor	Android	25.500	yes	yes	13.000	8.000	64.000	6.000	7250.000	480.000	2020	345	278.93
4	Honor	Android	15.320	yes	no	13.000	8.000	64.000	3.000	5000.000	185.000	2020	293	140.87

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   brand_name        3454 non-null   object 
 1   os                3454 non-null   object 
 2   screen_size       3454 non-null   float64
 3   4g                3454 non-null   object 
 4   5g                3454 non-null   object 
 5   main_camera_mp    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  new_price         3454 non-null   float64
 14  used_price        3454 non-null   float64
dtypes: float64(9), int64(2), object(4)
memory usage: 404.9+ KB
```

Observations

- There are 3454 rows and 15 columns.
- `brand_name`, `os`, `4g`, and `5g` are *object* type columns while the rest are numeric in nature.

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
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
new_price             0
used_price            0
dtype: int64
```

In [5]:

```
df.duplicated().sum()
```

Out[5]: 0

Observations

- There are 179 missing values in the `main_camera_mp` column, of float type.
- There are less than 10 missing values each in the `selfie_camera_mp`, `int_memory`, `ram`, `battery`, and `weight` columns.
- There are no duplicate values.

In [6]:

```
# let's view a sample of the data (random_state set to 1 to validate data every time)

df.sample(n=10, random_state=1)
```

Out[6]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	new_p
866	Others	Android	15.240	no	no	8.000	2.000	16.000	4.000	3000.000	206.000	2014	632	179.
957	Celkon	Android	10.160	no	no	3.150	0.300	512.000	0.250	1400.000	140.000	2013	637	48.
280	Infinix	Android	15.390	yes	no	NaN	8.000	32.000	2.000	5000.000	185.000	2020	329	88.
2150	Oppo	Android	12.830	yes	no	13.000	16.000	64.000	4.000	3200.000	148.000	2017	648	281.
93	LG	Android	15.290	yes	no	13.000	5.000	32.000	3.000	3500.000	179.000	2019	216	200.
1040	Gionee	Android	12.830	yes	no	13.000	8.000	32.000	4.000	3150.000	166.000	2016	970	279.
3170	ZTE	Others	10.160	no	no	3.150	5.000	16.000	4.000	1400.000	125.000	2014	1007	69
2742	Sony	Android	12.700	yes	no	20.700	2.000	16.000	4.000	3000.000	170.000	2013	1060	330.
102	Meizu	Android	15.290	yes	no	NaN	20.000	128.000	6.000	3600.000	165.000	2019	332	420
1195	HTC	Android	10.290	no	no	8.000	2.000	32.000	4.000	2000.000	146.000	2015	892	131.

Observations:

- The data cover a variety of brands like Oppo, Sony, LG, etc.

- A high percentage of devices seem to be running on Android.

In [7]:
```
# create a copy of the data so that the original dataset is not changed.

df2 = df.copy()
```

Observations

- The os column is mainly Android, indicating that is the most popular.
- The main_camera_mp column has a few missing values.
- The int_memory column has a wide range of values, from 16.0 to 512.0.
- The release_year column seems to be fairly evenly split between the years from 2014 to 2020.
- The days_used column also has a wide range, from 216 to 1060.

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 thorough analysis of the data, in addition to the questions completed below, will help to approach the analysis in the right manner and generate insights from the data.

Questions:

1. What does the distribution of 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. Budget devices nowadays offer great selfie cameras, allowing us to capture our favorite moments with loved ones. What is the distribution of budget devices offering greater than 8MP selfie cameras across brands?
7. Which attributes are highly correlated with the price of a used device?

In [8]:
```
# define a function to plot a boxplot and a histogram along the same scale

def histbox(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, (box, hist) = plt.subplots(
        nrows=2,                                           # Number of rows of the subplot grid = 2
                                                             # boxplot first then histogram created below
        sharex=True,                                       # x-axis same among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},       # boxplot 1/3 height of histogram
        figsize=figsize,                                   # figsize defined above as (12, 7)
    )
    # defining boxplot inside function, so when using it say histbox(df, 'cost'), df: data and cost: feature

    sns.boxplot(
        data=data, x=feature, ax=box, showmeans=True, color="chocolate"
    )  # showmeans makes mean val on boxplot have star, ax =
    sns.histplot(
        data=data, x=feature, kde=kde, ax=hist, bins=bins, color = "darkgreen"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=hist, color = "darkgreen"
    )  # For histogram if there are bins in potential graph

    # add vertical line in histogram for mean and median
    hist.axvline(
        data[feature].mean(), color="purple", linestyle="--"
    )  # Add mean to the histogram
    hist.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [9]:

```
# define a function to create labeled barplots

def bar(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
```

In [10]:

```
df2.describe(include="all").T
```

Out[10]:

	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	3454.000	NaN	NaN	NaN	13.713	3.805	5.080	12.700	12.830	15.340	30.710
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	3275.000	NaN	NaN	NaN	9.460	4.815	0.080	5.000	8.000	13.000	48.000
selfie_camera_mp	3452.000	NaN	NaN	NaN	6.554	6.970	0.000	2.000	5.000	8.000	32.000
int_memory	3450.000	NaN	NaN	NaN	54.573	84.972	0.010	16.000	32.000	64.000	1024.000
ram	3450.000	NaN	NaN	NaN	4.036	1.365	0.020	4.000	4.000	4.000	12.000
battery	3448.000	NaN	NaN	NaN	3133.403	1299.683	500.000	2100.000	3000.000	4000.000	9720.000
weight	3447.000	NaN	NaN	NaN	182.752	88.413	69.000	142.000	160.000	185.000	855.000
release_year	3454.000	NaN	NaN	NaN	2015.965	2.298	2013.000	2014.000	2015.500	2018.000	2020.000
days_used	3454.000	NaN	NaN	NaN	674.870	248.580	91.000	533.500	690.500	868.750	1094.000

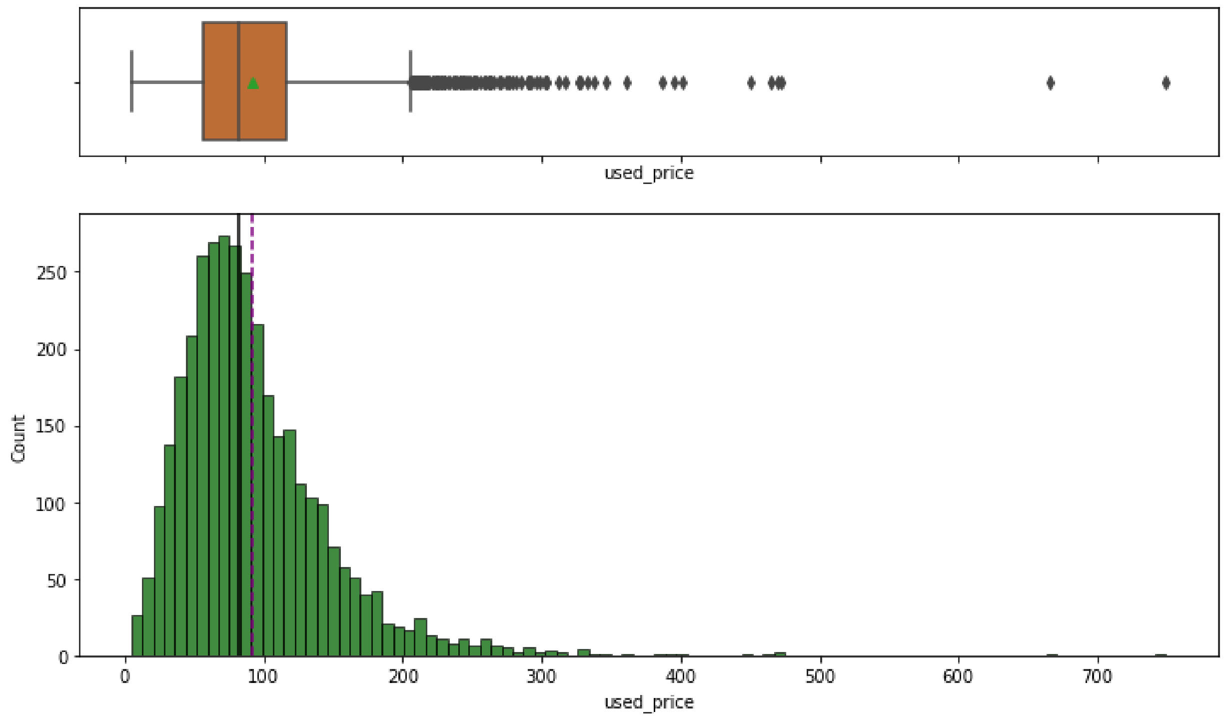
	count	unique	top	freq	mean	std	min	25%	50%	75%	max
new_price	3454.000	NaN	NaN	NaN	237.039	194.303	18.200	120.343	189.785	291.115	2560.200
used_price	3454.000	NaN	NaN	NaN	92.303	54.702	4.650	56.483	81.870	116.245	749.520

Observations

- There are 33 brands in the data and a category *Others* too.
- Android is the most common OS for the used devices.
- The weight ranges from 69g to 855g.
    - This does not seem incorrect as the data contains feature phones and tablets too.
- There are a few unusually low values for the internal memory and RAM of used devices, but those are likely due to the presence of feature phones in the data.
- The average value of the price of a used device is approx. 2/5 times the price of a new model of the same device.

Question 1: What does the distribution of used device prices look like?

In [11]:
```
histbox(df2, "used_price")
```

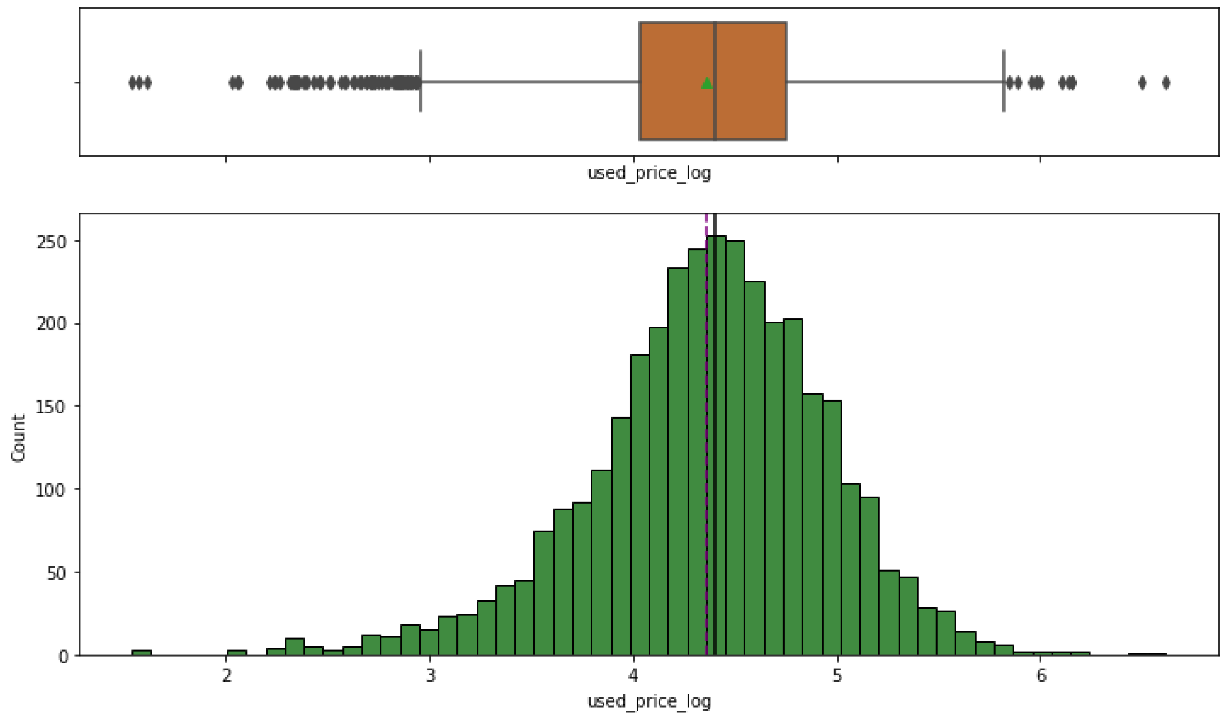


Observations:

- The distribution of used device prices is heavily right-skewed, with a mean value of ~100 euros.
- Let's apply the log transform to see if we can make the distribution closer to normal.

In [12]:
```
df2["used_price_log"] = np.log(df2["used_price"])
```

In [13]:
```
histbox(df2, "used_price_log")
```



- The used device prices are almost normally distributed now.

Question 2. What percentage of the used device market is dominated by Android devices?

In [14]:
```
# We know from above that there are no missing values in the used_price and new_price column, therefore the percentage
# of android devices is the same for both.

df2['os'].value_counts()
```

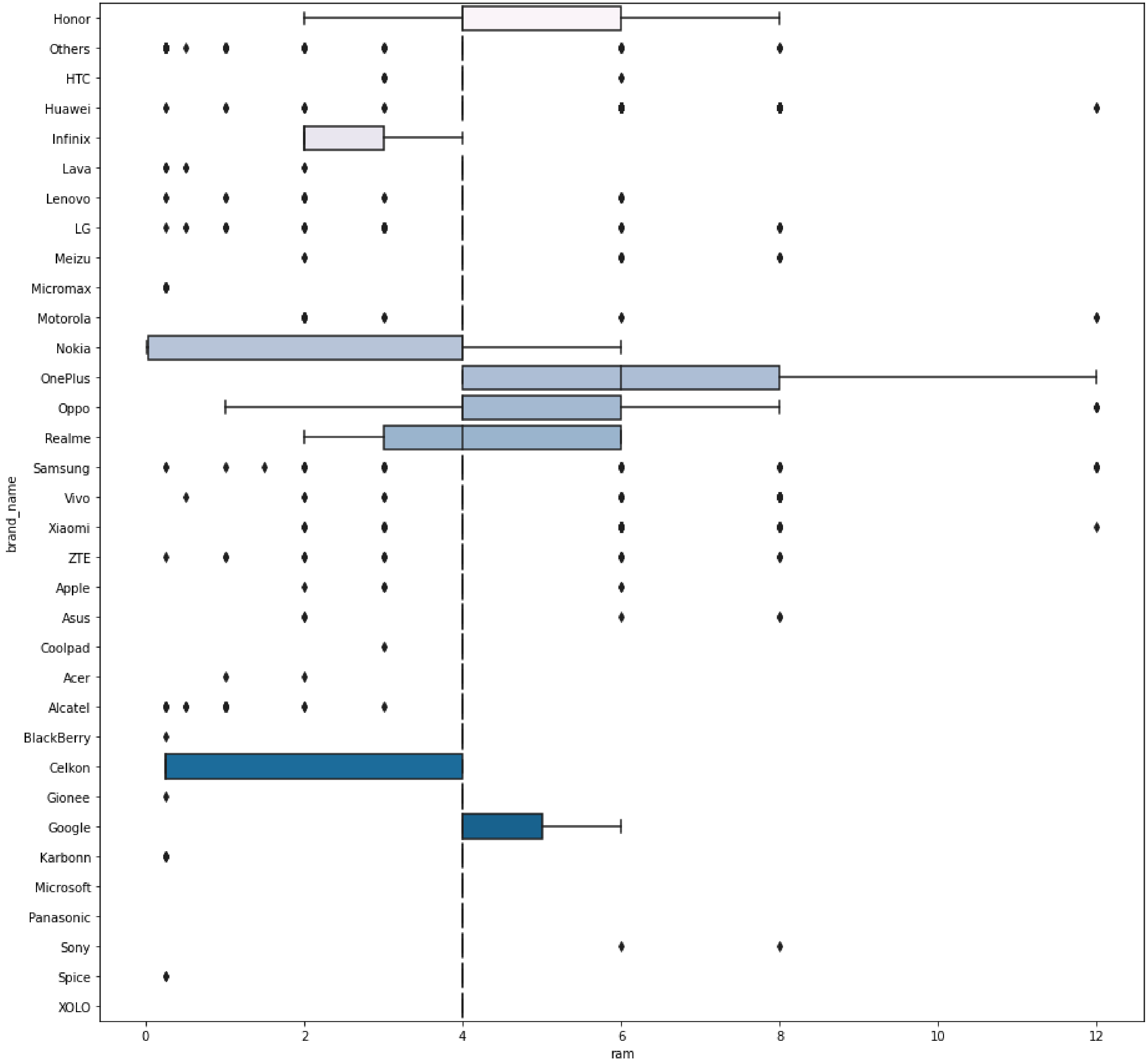
Out[14]:
```
Android    3214
Others      137
Windows      67
iOS          36
Name: os, dtype: int64
```

In [15]:
```
# divide Android device number by total number of rows in dataset:

print("The percent of Android devices in the used device market is ", (3214/df2.shape[0])*100, "%")
```

```
The percent of Android devices in the used device market is  93.05153445280834 %
```

Question 3. The amount of RAM is important for the smooth functioning of a device. How does the amount of RAM vary with the brand?

In [16]:
```
plt.figure(figsize=(15,15))
sns.boxplot(x = "ram", y = "brand_name", data = df2, palette = 'PuBu')
plt.show()
```


Honor
Others
HTC
Huawei
Infinix
Lava
Lenovo
LG
Meizu
Micromax
Motorola
Nokia
OnePlus
Oppo
Realme
Samsung
Vivo
Xiaomi
ZTE
Apple
Asus
Coolpad
Acer
Alcatel
BlackBerry
Celkon
Gionee
Google
Karbonn
Microsoft
Panasonic
Sony
Spice
XOLO
brand_name
0
2
4
6
8
10
12
ram

In [17]:
```
df2['ram'].value_counts()
```

Out[17]:
```
4.000     2815
6.000      154
8.000      130
2.000       90
0.250       83
3.000       81
1.000       34
12.000      18
0.020       18
0.030       17
0.500        9
1.500        1
Name: ram, dtype: int64
```

In [18]:
```
# display the 10 most common brand names

df2['brand_name'].value_counts()[:10]
```

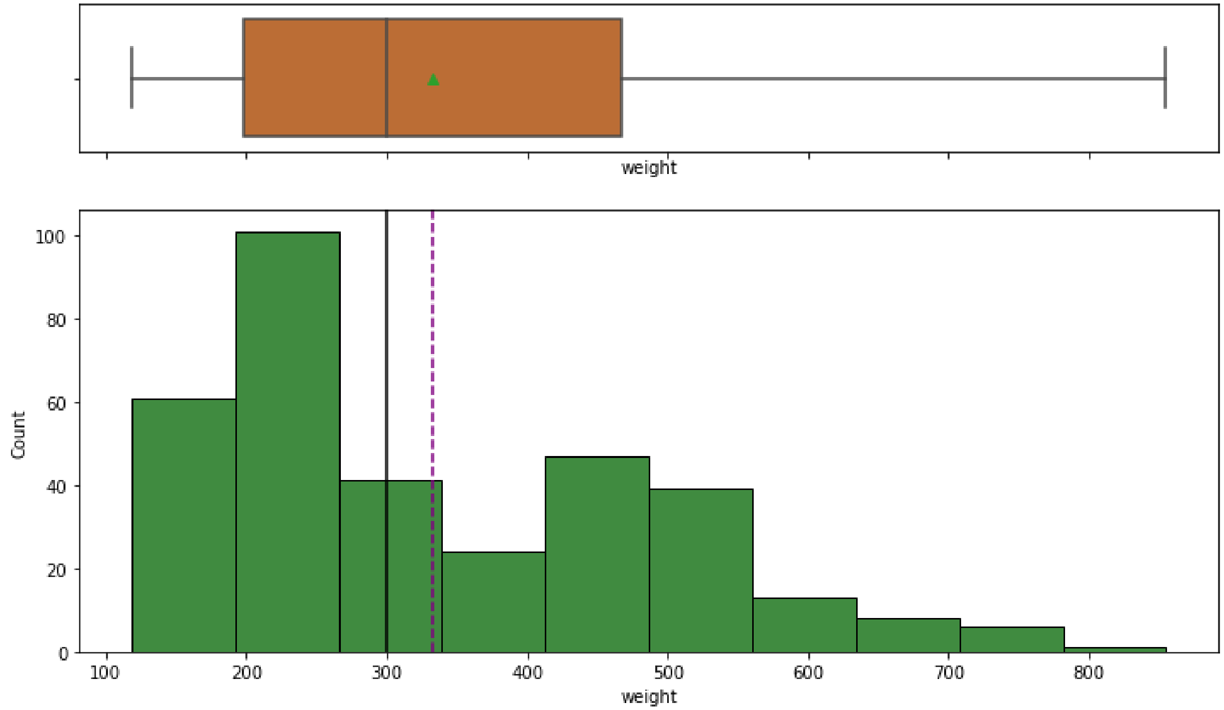
Out[18]:
```
Others     502
Samsung    341
Huawei     251
LG         201
Lenovo     171
ZTE        140
Xiaomi     132
Oppo       129
Asus       122
Alcatel    121
Name: brand_name, dtype: int64
```

Observations:

- The 10 most common brands (including "Others) make up 2110 entries in the dataset, about 65% of all, and the most common 'ram' median is 4.0. Looking at the boxplot above, most brands have a box that is very small at the 4 ram marker.
- Google, RealMe, Oppo, OnePlus, and Honor land in the exception for this, with the ranges weighing more heavily past the ram value of 4. (This is especially apparent with the OnePlus boxplot).
- Infinix, Nokia, and Celkon are also in the exception for this, but on the other end, with ranges weighing more heavily in the 0-4 ram value.
- There are a sizeable number of outliers across all brands, with more appearing in the 0-4 range of ram values.

Question 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)?

In [19]:
```
large_battery = df2[df2['battery'] > 4500.0]
histbox(large_battery, "weight")
```



Observations:

The weight distribution for phones above 4500 mAh is right skewed, with a median right around 300, and mean just above 300.

Question 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?

In [20]:

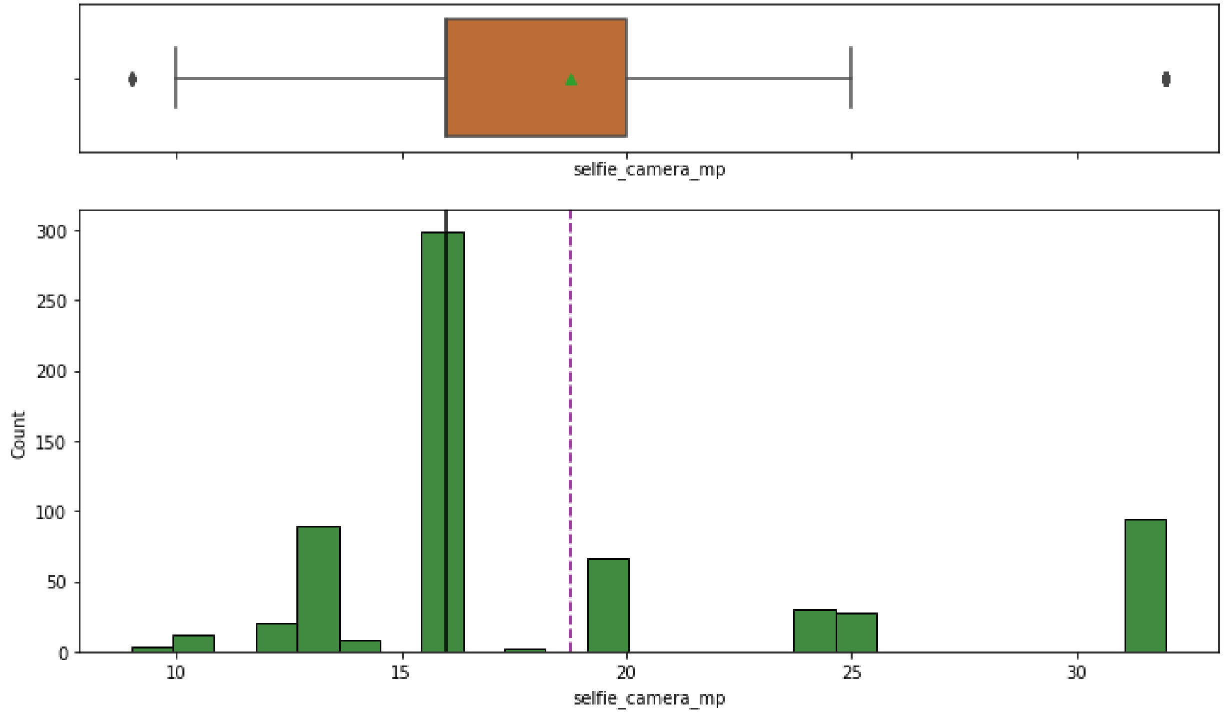
```
large_screen = df2[df2['screen_size'] > 6 * 2.54]

print("There are ", large_screen['os'].count(), "phones and tablets available across different brands with a screen size larger than
```

```
There are  1099 phones and tablets available across different brands with a screen size larger than 6 inches.
```

Question 6. Budget devices nowadays offer great selfie cameras, allowing us to capture our favorite moments with loved ones. What is the distribution of budget devices offering greater than 8MP selfie cameras across brands?

In [21]:

```
large_mp = df2[df2['selfie_camera_mp'] > 8.0]
histbox(large_mp, "selfie_camera_mp")
```



Observations:

This is a non-normal, multimodal distribution, with a slight right skew. The median falls close to 16 MP, and the mean just below 20 MP.

Question 7. Which attributes are highly correlated with the price of a used device?

In [22]:
```
df2.corr()
```

Out[22]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	new_price	used_price	use
screen_size	1.000	0.150	0.272	0.071	0.274	0.814	0.829	0.364	-0.292	0.341	0.529	
main_camera_mp	0.150	1.000	0.429	0.019	0.261	0.249	-0.088	0.354	-0.145	0.358	0.459	
selfie_camera_mp	0.272	0.429	1.000	0.296	0.477	0.370	-0.005	0.691	-0.553	0.416	0.615	
int_memory	0.071	0.019	0.296	1.000	0.122	0.118	0.015	0.235	-0.243	0.369	0.378	
ram	0.274	0.261	0.477	0.122	1.000	0.281	0.090	0.314	-0.280	0.494	0.529	
battery	0.814	0.249	0.370	0.118	0.281	1.000	0.703	0.489	-0.371	0.370	0.550	
weight	0.829	-0.088	-0.005	0.015	0.090	0.703	1.000	0.071	-0.067	0.219	0.358	
release_year	0.364	0.354	0.691	0.235	0.314	0.489	0.071	1.000	-0.750	0.304	0.495	
days_used	-0.292	-0.145	-0.553	-0.243	-0.280	-0.371	-0.067	-0.750	1.000	-0.246	-0.386	

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	new_price	used_price	use
new_price	0.341	0.358	0.416	0.369	0.494	0.370	0.219	0.304	-0.246	1.000	0.809	
used_price	0.529	0.459	0.615	0.378	0.529	0.550	0.358	0.495	-0.386	0.809	1.000	
used_price_log	0.615	0.587	0.608	0.191	0.520	0.614	0.382	0.510	-0.358	0.674	0.895	

Observations

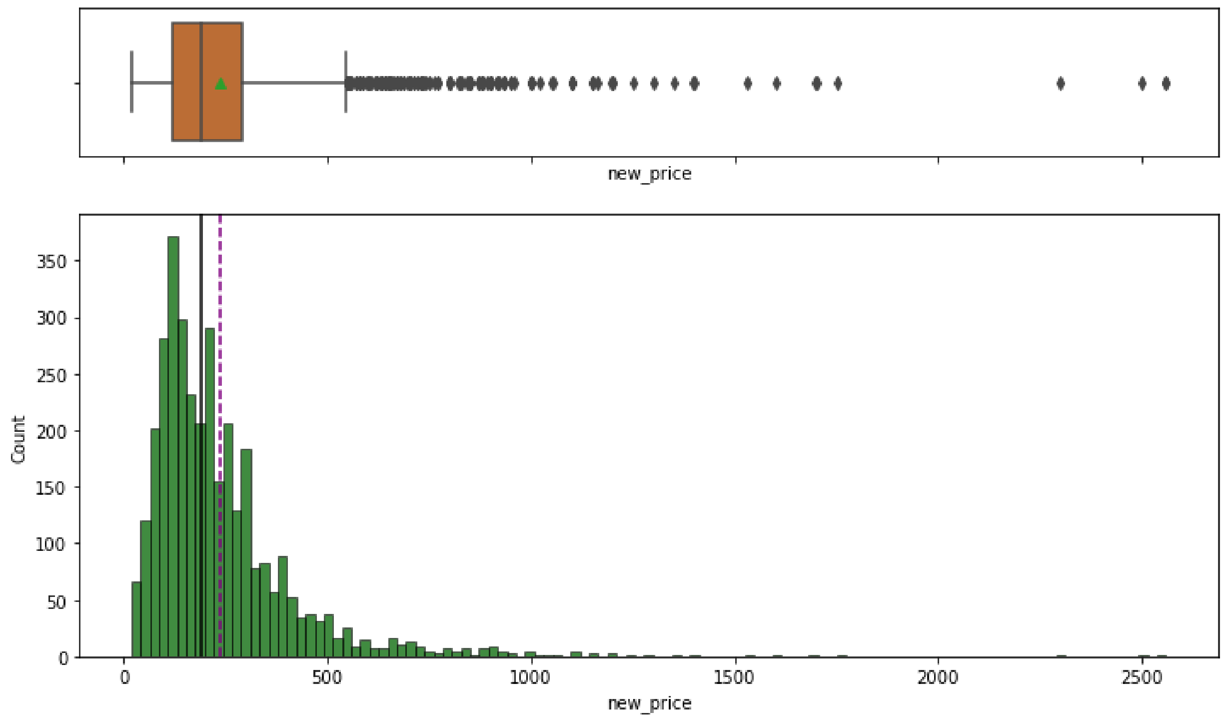
When we look at the column for used_price, the attributes with the highest correlation are selfie_camera_mp, and new_price. There are a few with medium high correlations, like screen_size, ram, and battery.

Exploratory Data Analysis (EDA) Visualizations

Univariate Analysis

In [23]:

```
# Used_price histogram/boxplot above for Question 1.

histbox(df2, "new_price")
```

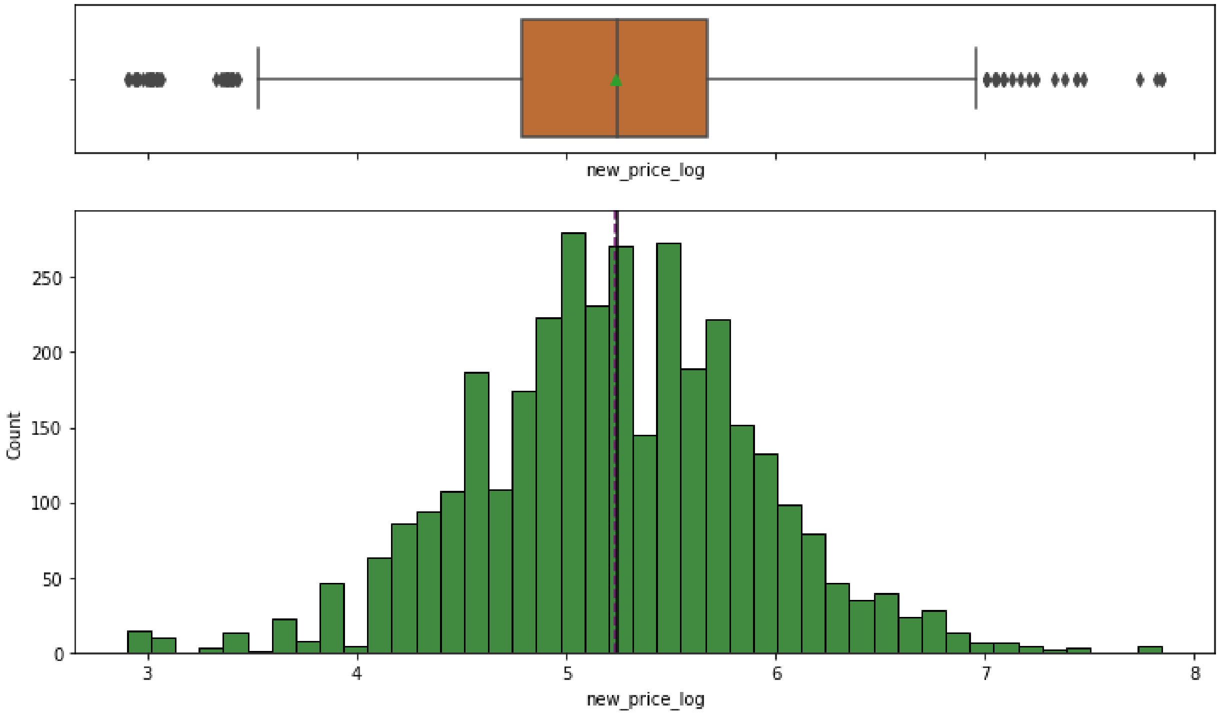


Observations

- The distribution is heavily right-skewed, with a mean value of ~240 euros.
- Let's apply the log transform to see if we can make the distribution closer to normal.

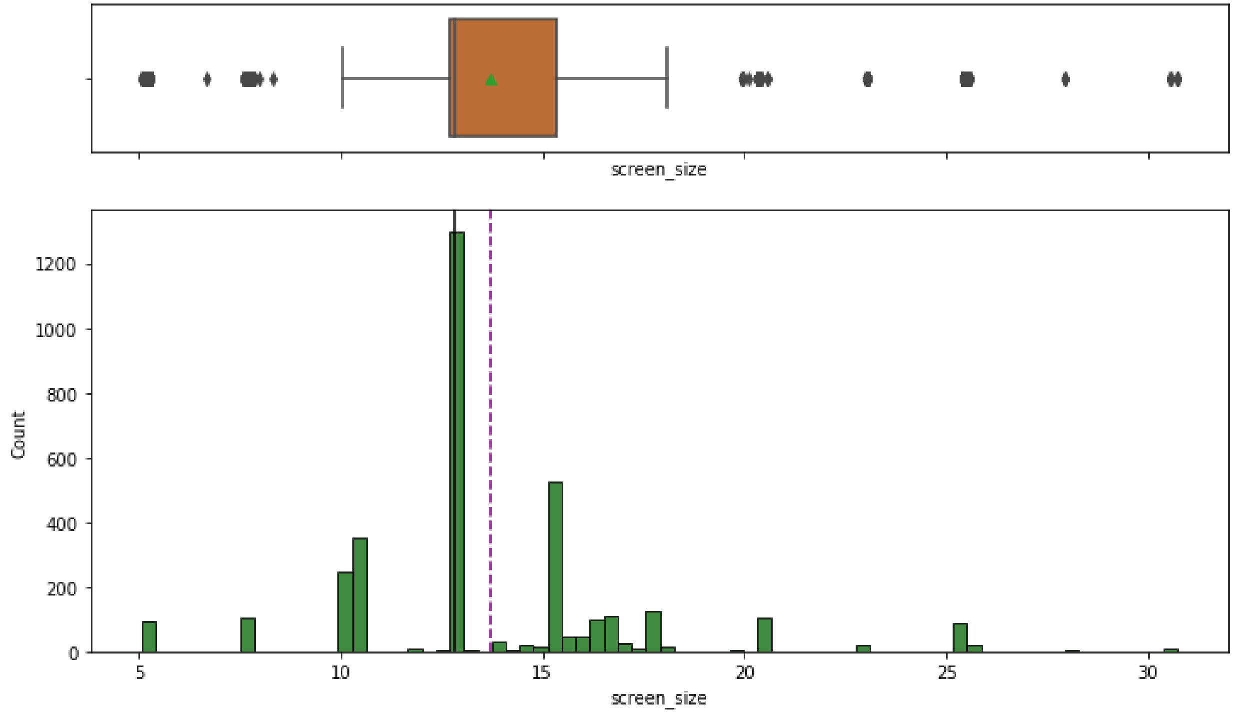
In [24]:
```
df2["new_price_log"] = np.log(df2["new_price"])
```

In [25]:
```
histbox(df2, "new_price_log")
```



- The prices of new device models are almost normally distributed now.

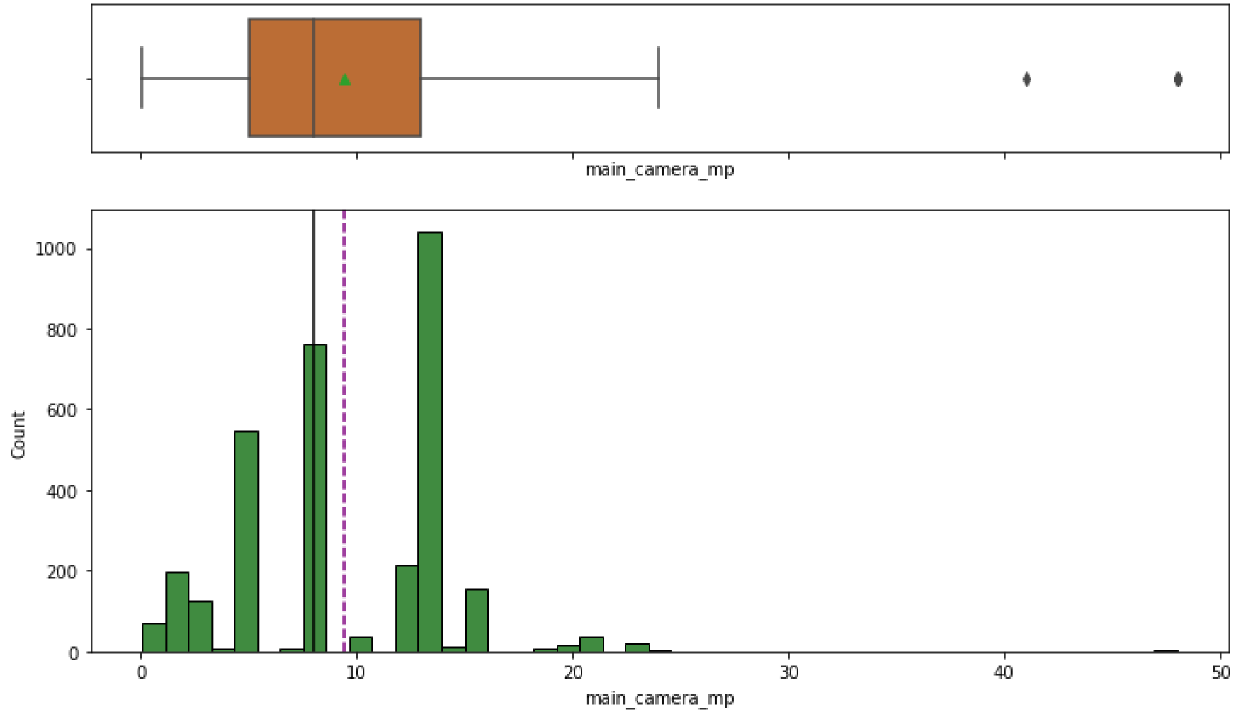
In [26]:
```
histbox(df2, "screen_size")
```



Observations:

- Around 50% of the devices have a screen larger than 13cm.

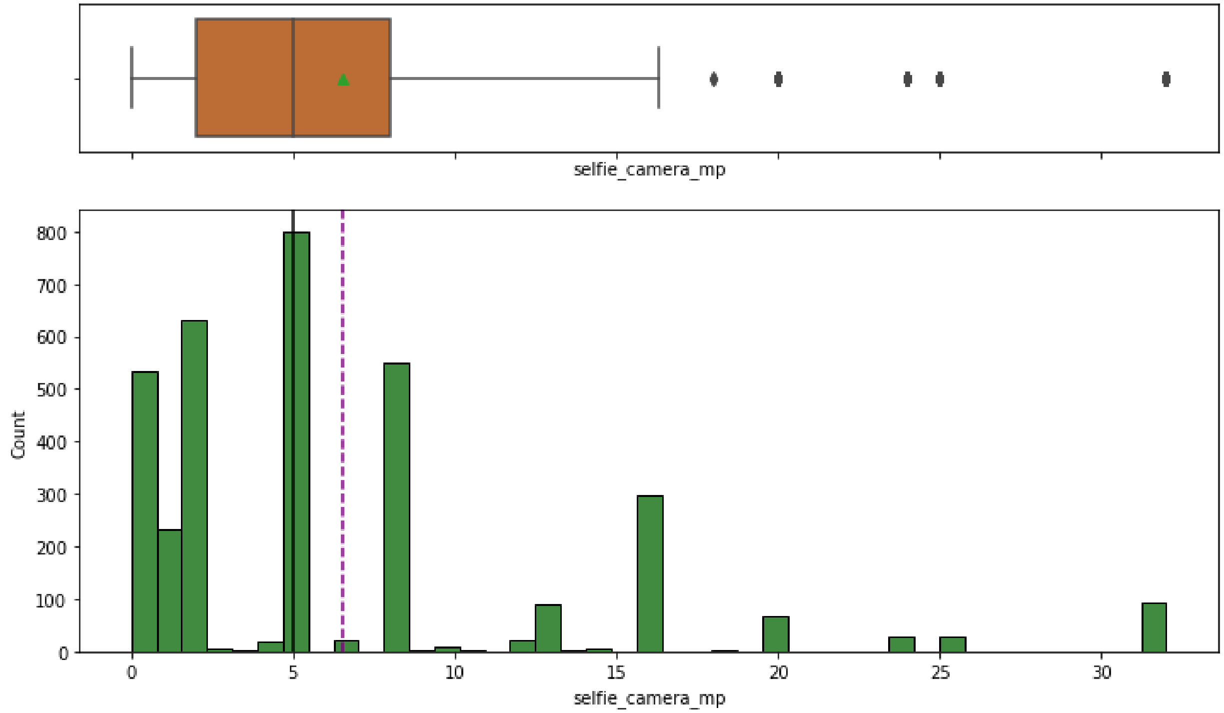
In [27]:
```
histbox(df2, "main_camera_mp")
```



Observations

- Few devices offer rear cameras with more than 20MP resolution.

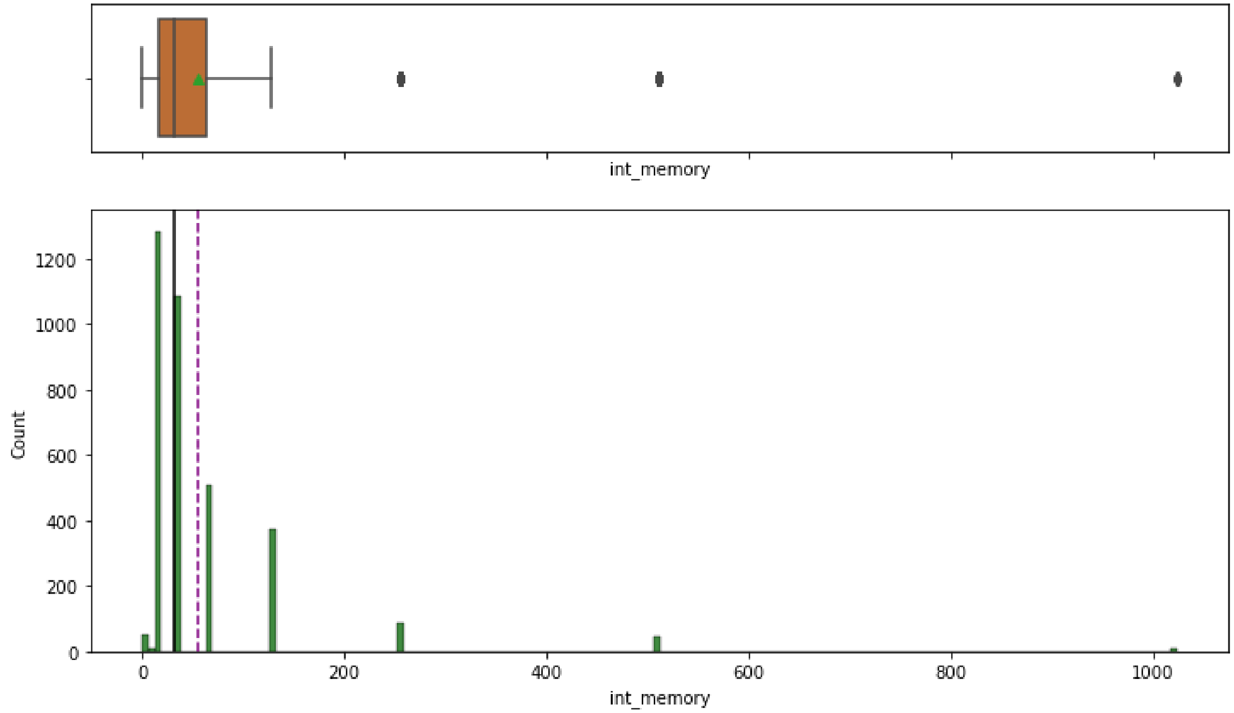
In [28]:
```
histbox(df2, "selfie_camera_mp")
```



Observations

- Some devices do not provide a front camera, while few devices offer ones with more than 16MP resolution.

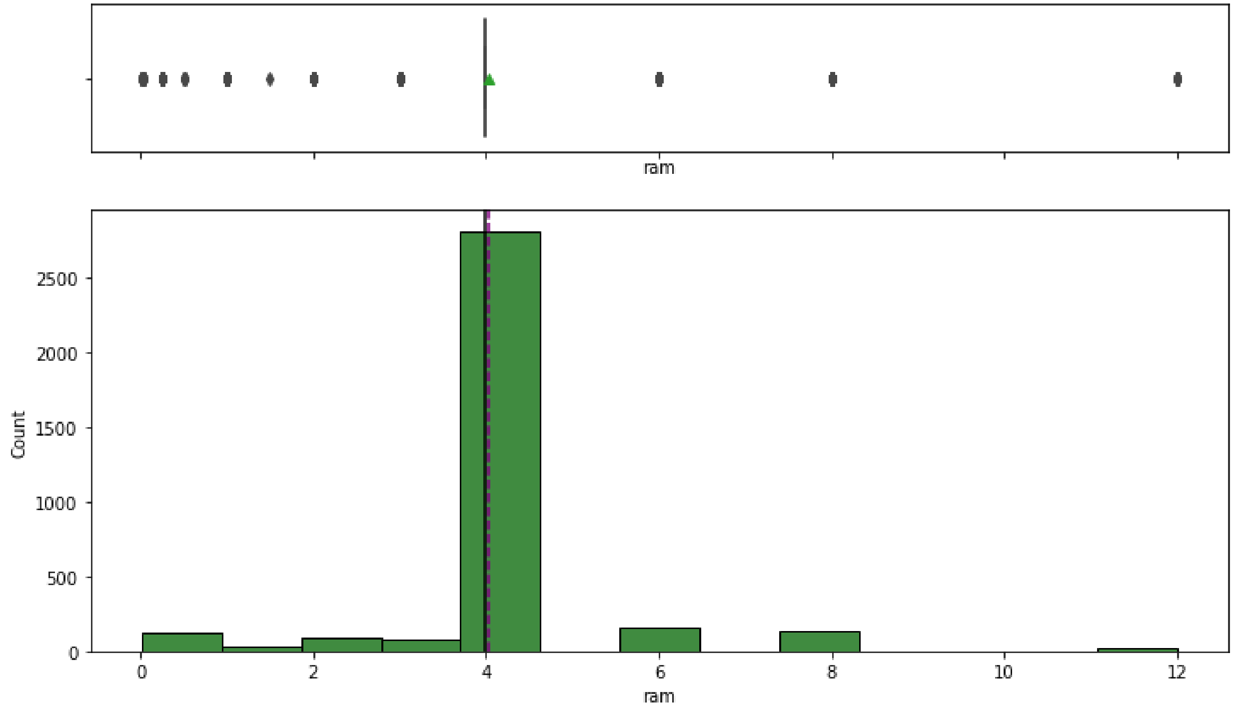
In [29]:
```
histbox(df2, "int_memory")
```



Observations

- Few devices offer more than 256GB internal memory.

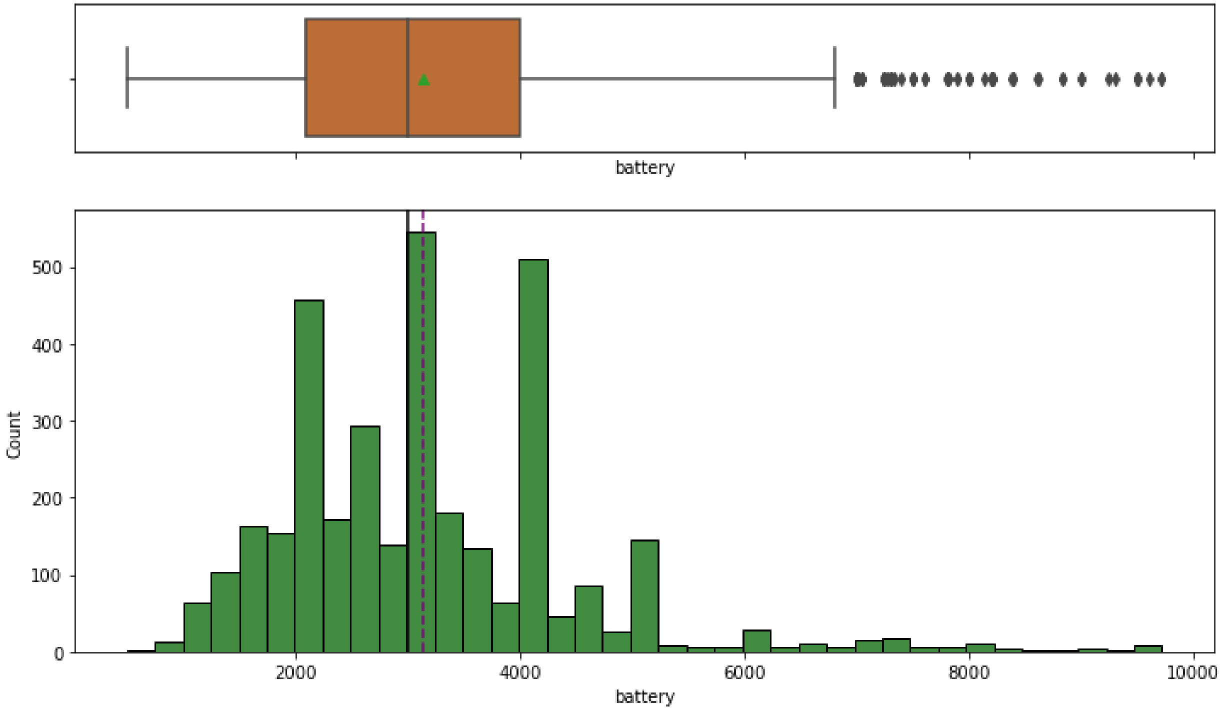
In [30]:
```
histbox(df2, "ram")
```



Observations

- Most of the devices offer 4GB RAM and very few offer greater than 8GB RAM.

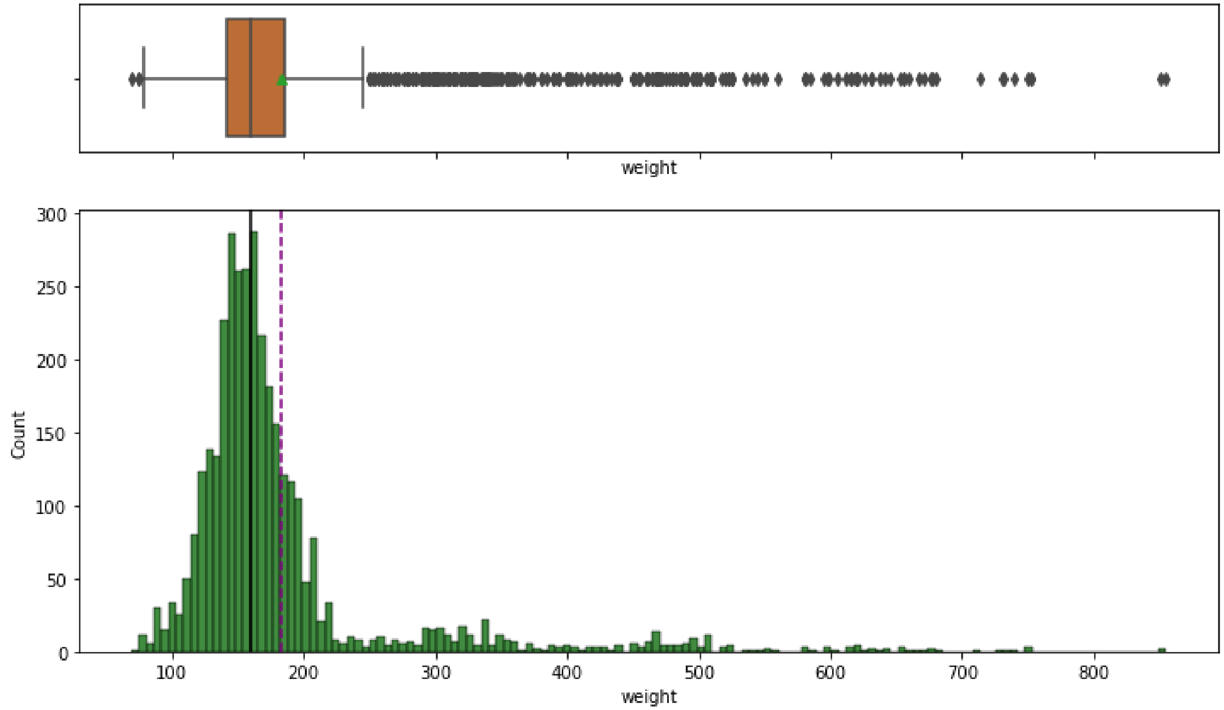
In [31]:
```
histbox(df2, "battery")
```



Observations

- The distribution of energy capacity of battery is close to normally distributed with a few upper outliers.

In [32]:
```
histbox(df2, "weight")
```

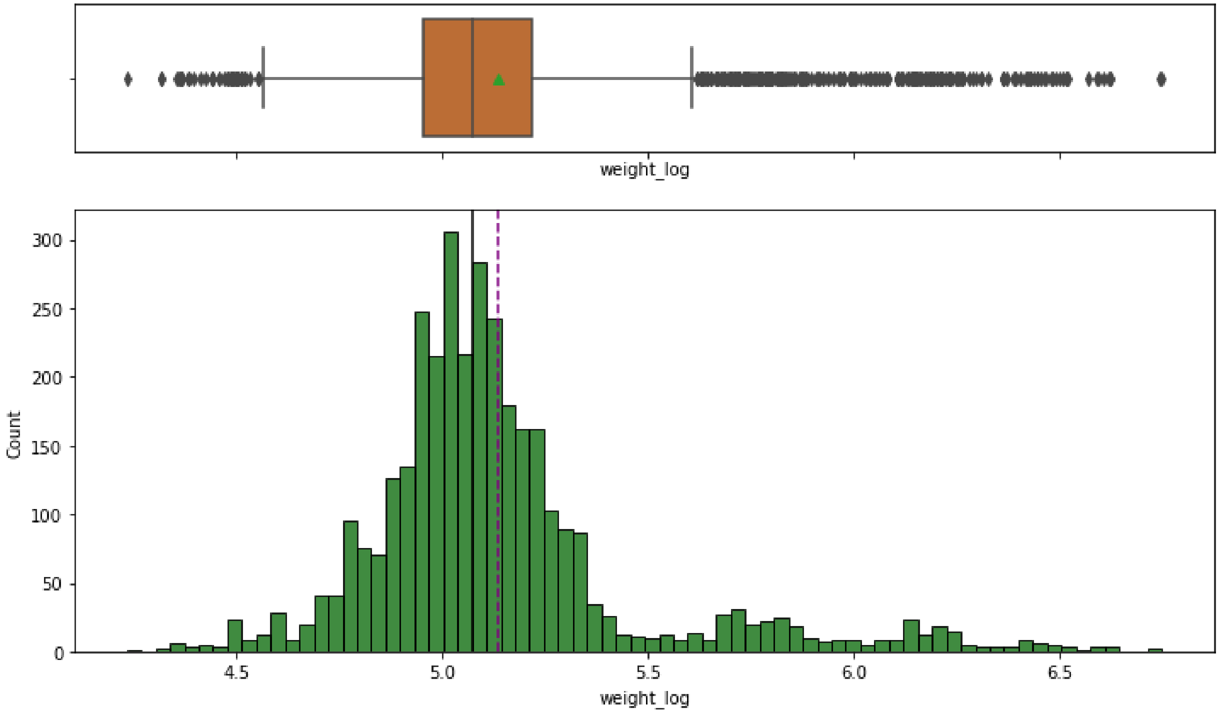


Observations

- The distribution of weight is right-skewed and has many upper outliers.
- Let's apply the log transform to see if we can make the distribution closer to normal.

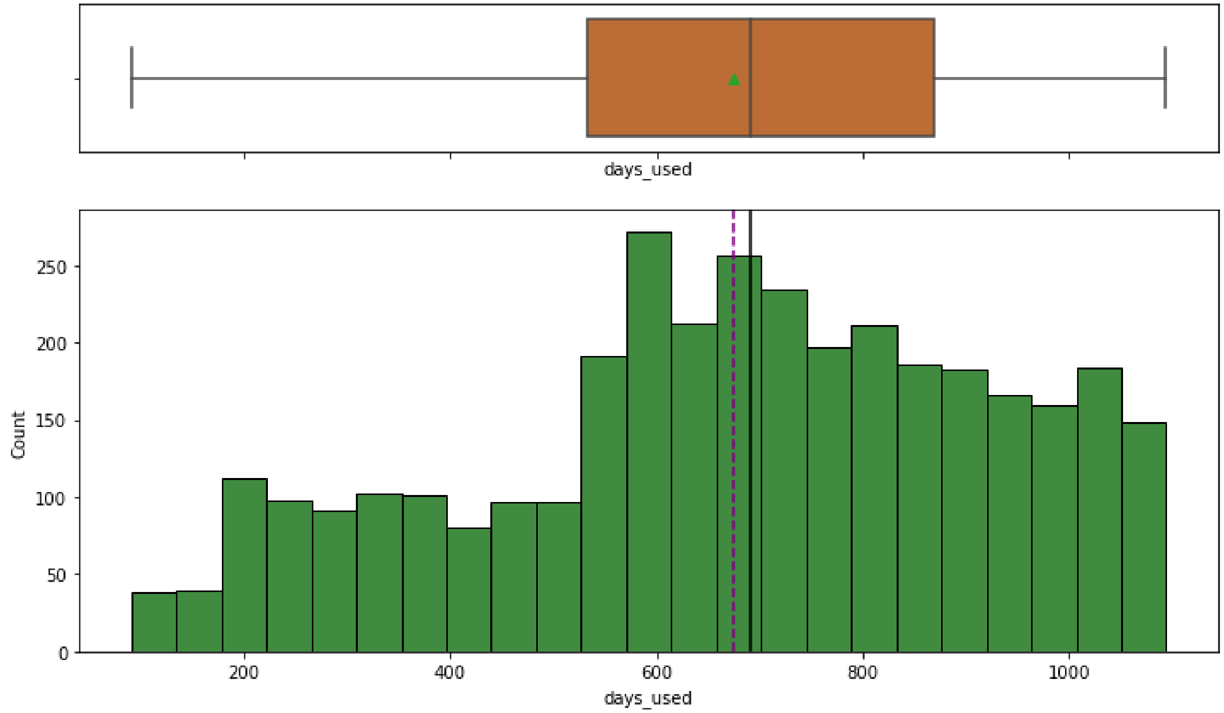
In [33]:
```
df2["weight_log"] = np.log(df2["weight"])
```

In [34]:
```
histbox(df2, "weight_log")
```



- The distribution is closer to normal now, but there are still a lot of upper outliers.

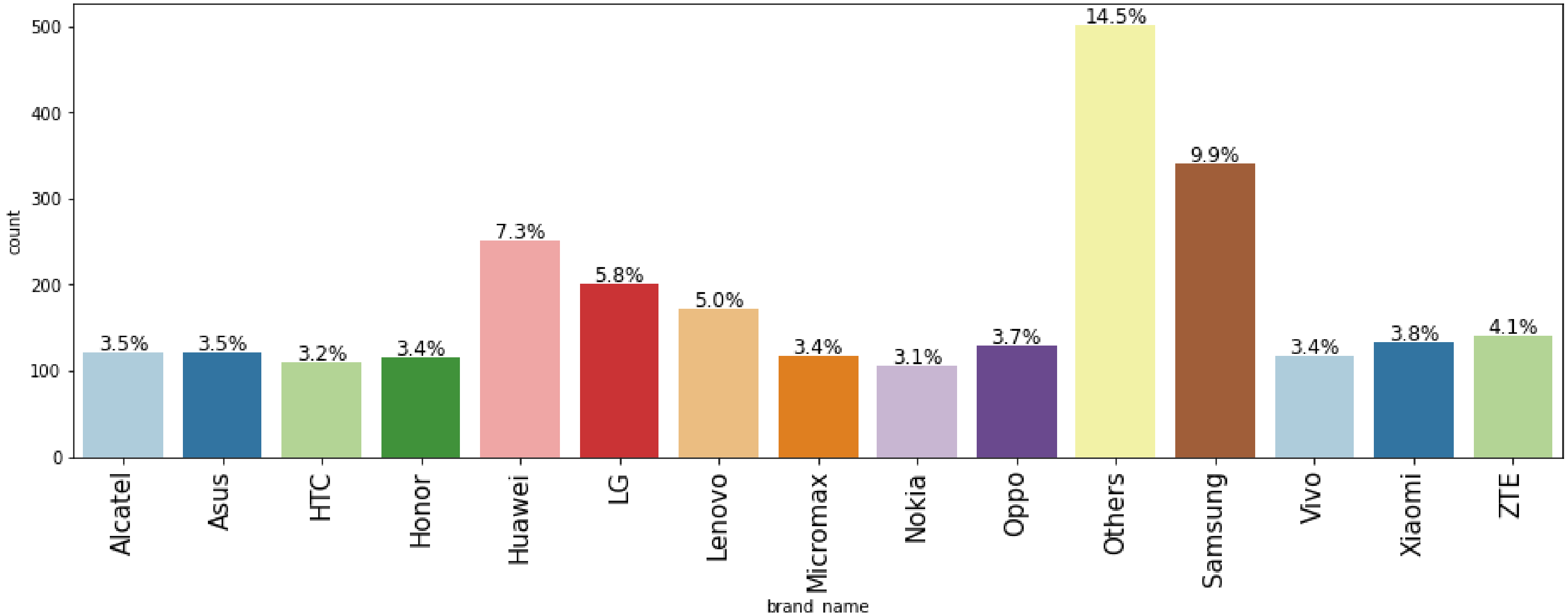
In [35]:
```
histbox(df2, "days_used")
```



Observations

- Around 50% of the devices in the data have been used for more than 700 days.

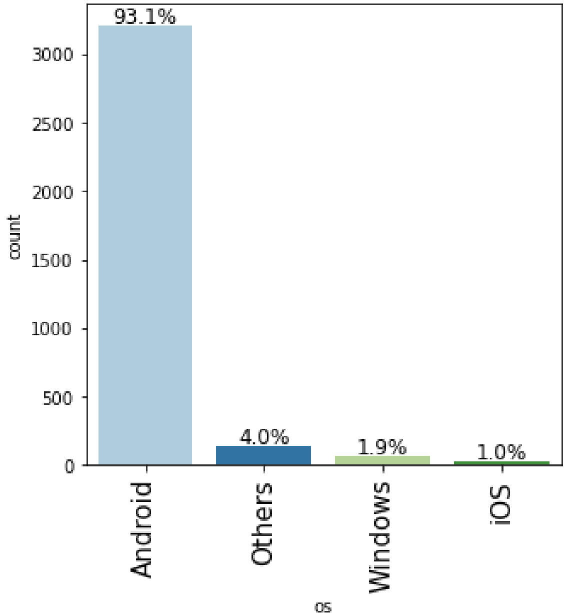
In [36]:
```
bar(df2, 'brand_name', perc = True, n=15)
```



Observations

- Samsung has the most number of devices in the data, followed by Huawei and LG.
- 14.5% of the devices in the data are from brands other than the listed ones.

In [37]:
```
bar(df2, "os", perc = True)
```

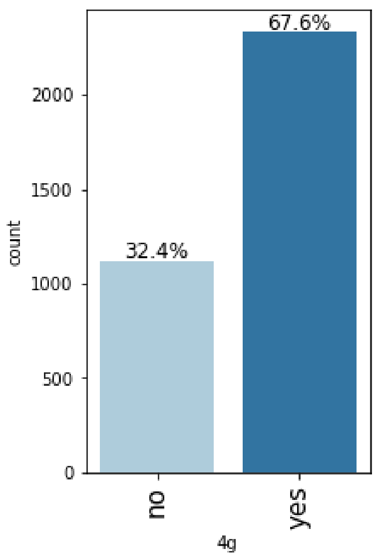


Observations

- Android devices dominate ~93% of the used device market.

In [38]:

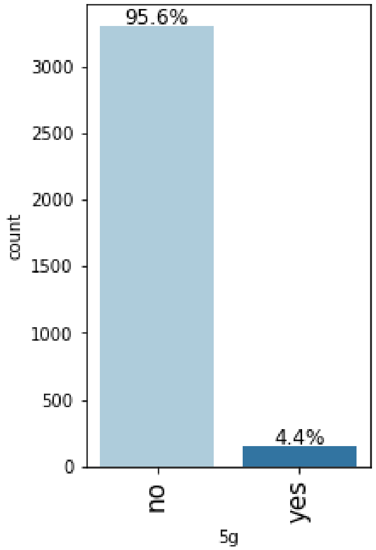
```
bar(df2, "4g", perc = True)
```



Observations

- Nearly two-thirds of the devices in this data have 4G available.

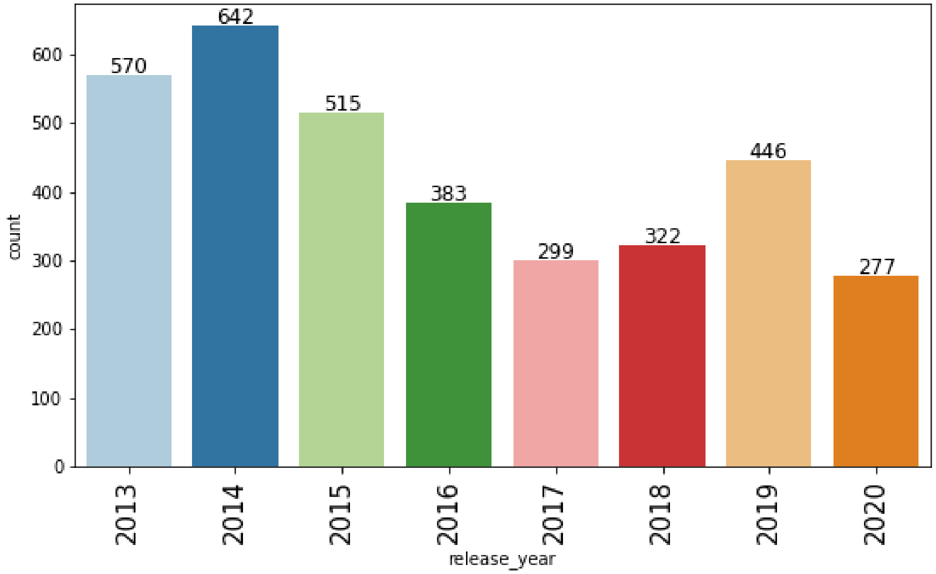
In [39]:
```
bar(df2, "5g", perc = True)
```



Observations

- Very few devices in this data provide 5G network.

In [40]:
```
bar(df2, "release_year")
```



Observations

- Around 50% of the devices in the data were originally released in 2015 or before.

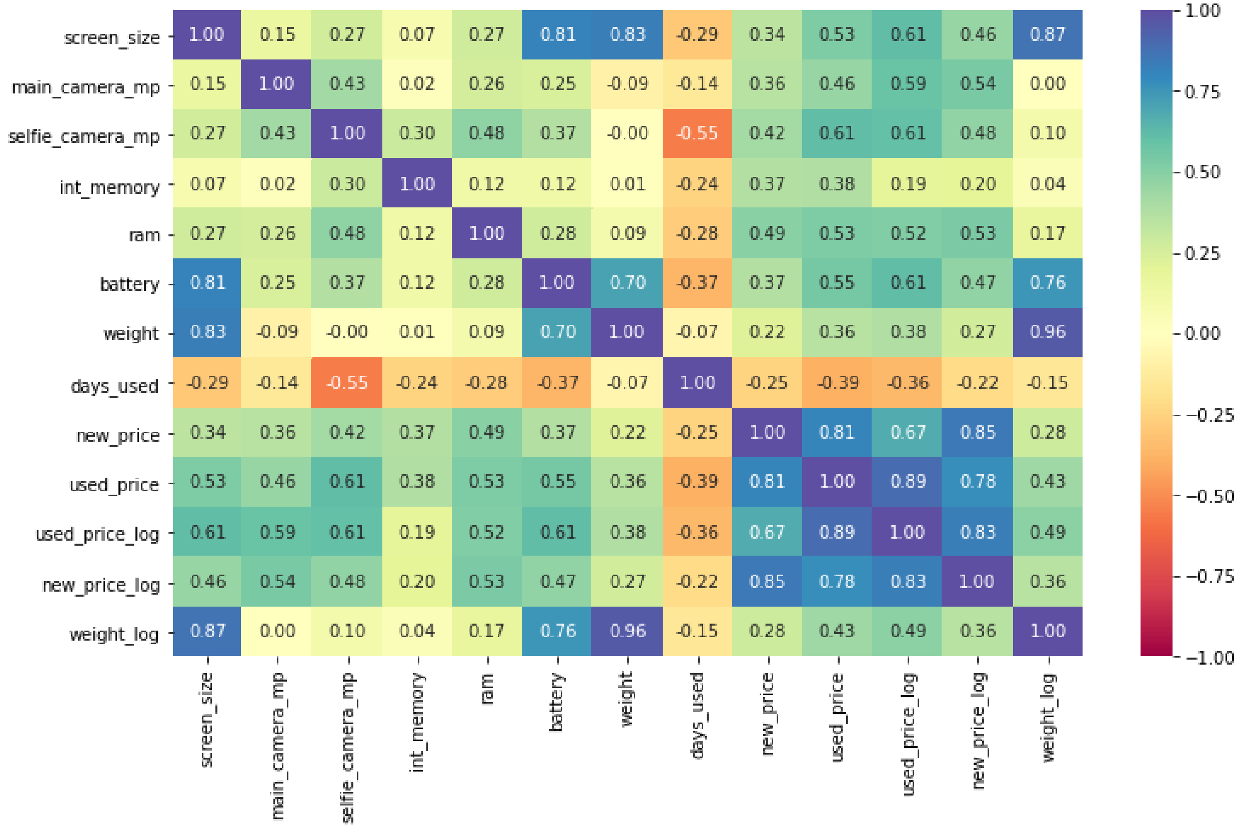
Bivariate Analysis

In [41]:

```
corr_cols = df2.select_dtypes(include=np.number).columns.tolist()

# dropping release_year as it is a temporal variable in preparation for heatmap/correlation table
corr_cols.remove("release_year")
```

In [42]:

```
plt.figure(figsize=(12, 7))
sns.heatmap(
    df2[corr_cols].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```

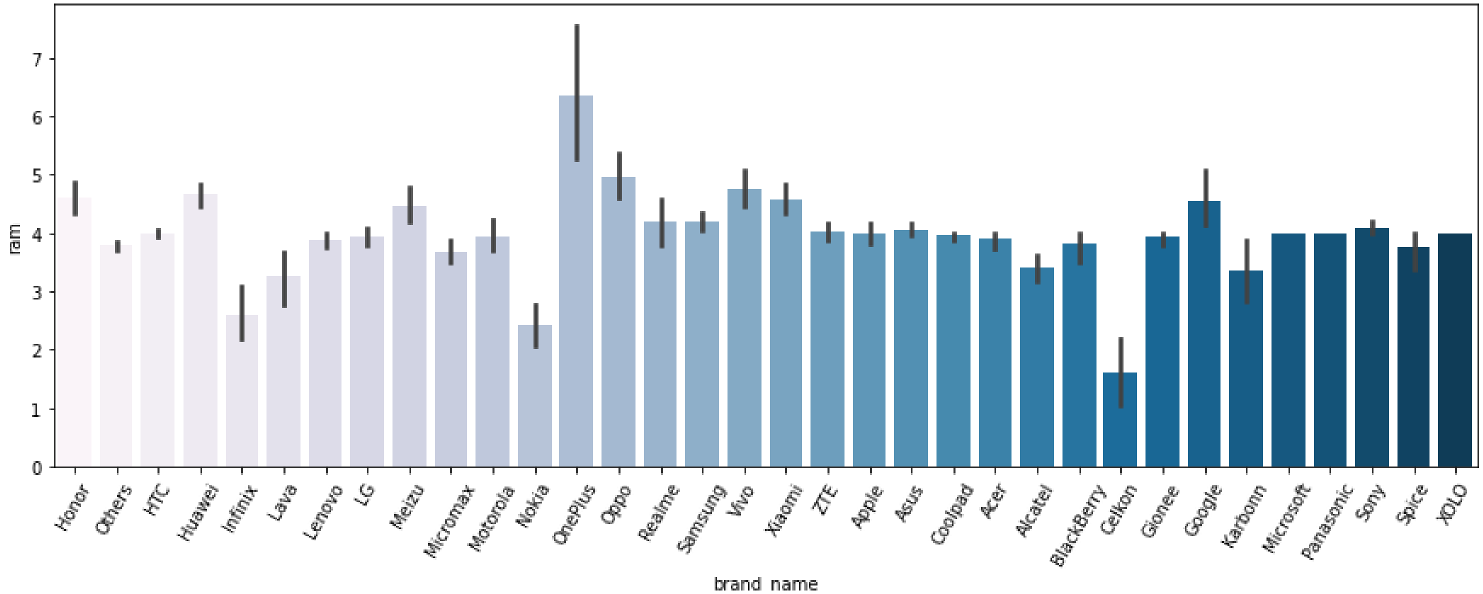


Observations

- The used device price is highly correlated with the price of a new device model.
  - This makes sense as the price of a new model is likely to affect the used device price.
- Weight, screen size, and battery capacity of a device show a good amount of correlation.
  - This makes sense as larger battery capacity requires bigger space, thereby increasing screen size and weight.
- The number of days a device is used is negatively correlated with the resolution of its front camera.
  - This makes sense as older devices did not offer as powerful front cameras as the recent ones.

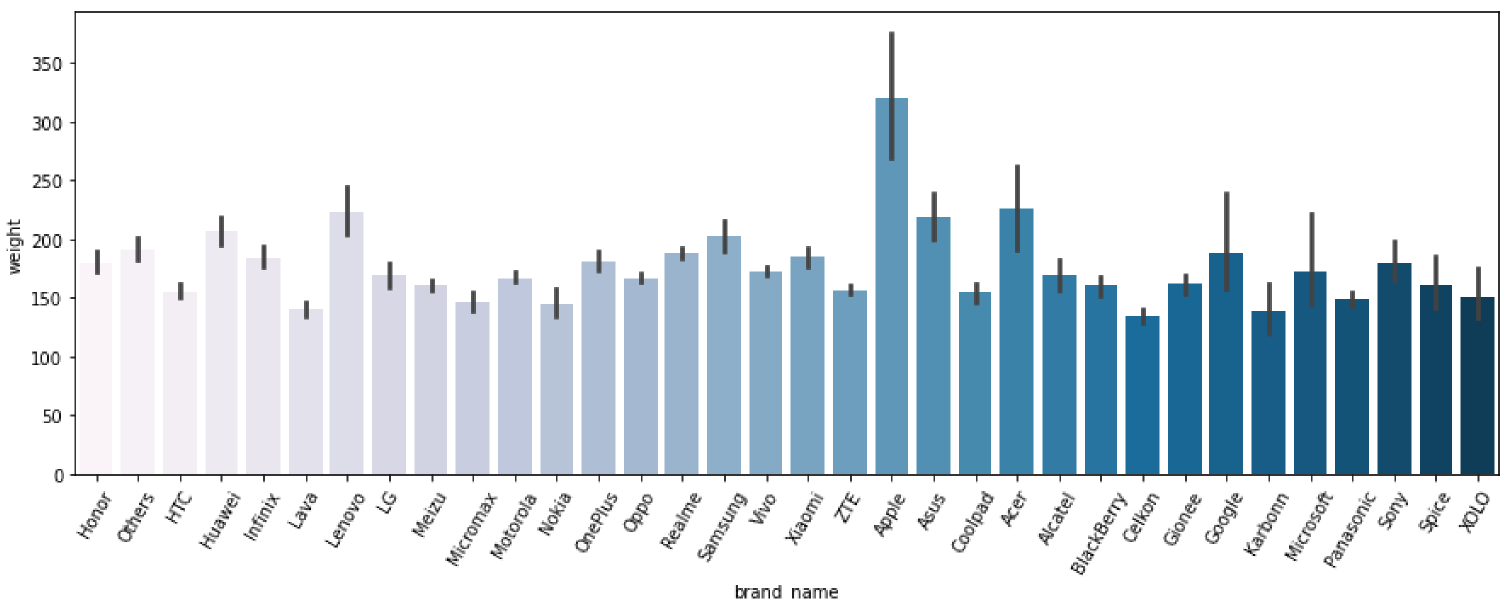
In [43]:

```
# check relationship between brand_name and ram (similar to Question 3, look there for boxplot version)

plt.figure(figsize=(15, 5))
sns.barplot(data=df2, x="brand_name", y="ram", palette ="PuBu")
plt.xticks(rotation =60)
plt.show()
```



In [44]:

```
# check relationship between brand name and weight

plt.figure(figsize=(15, 5))
sns.barplot(data=df2, x="brand_name", y="weight", palette ="PuBu")
plt.xticks(rotation =60)
plt.show()
```



Observations

- For brand_name/ram:
  - We saw from earlier (Question 3) that the majority of users have devices with a ram of 4. This is reflected here, with most brands' average ram around 4.
  - OnePlus offers the highest amount of RAM in general, while Celkon offers the least.
- For brand_name/weight
  - The average weight across brand names is about 150-200.
  - The higher end exceptions are noted in Apple, Asus, Acer, and Lenovo.
  - A few lower end exceptions are noted in HTC, Lava, Coolpad, Celkon.

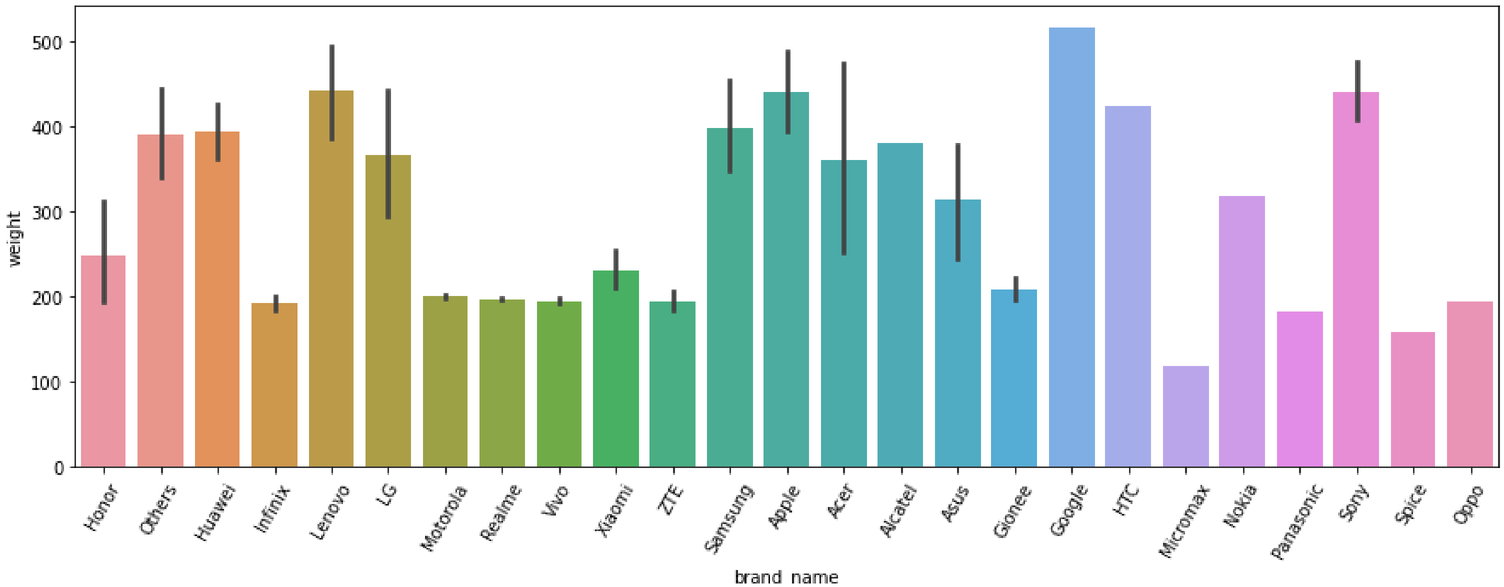
In [45]:
```
# Question 4 required us to create a subset of the dataframe, named large_battery. Use that to compare to other
# variables.

large_battery.groupby("brand_name")["weight"].mean().sort_values(ascending=True)
```

Out[45]:
```
brand_name
Micromax     118.000
Spice        158.000
Panasonic    182.000
Infinix      193.000
Oppo         195.000
ZTE          195.400
Vivo         195.631
Realme       196.833
Motorola     200.757
Gionee       209.430
Xiaomi       231.500
Honor        248.714
Asus         313.773
Nokia        318.000
Acer         360.000
LG           366.058
Alcatel      380.000
Others       390.546
Huawei       394.486
Samsung      398.352
HTC          425.000
Sony         439.500
Apple        439.559
Lenovo       442.721
Google       517.000
Name: weight, dtype: float64
```

In [46]:
```
large_battery.shape
```

Out[46]: (341, 16)

In [47]:
```
#brand_name vs weight for large battery subset of data

plt.figure(figsize=(15, 5))
sns.barplot(data=large_battery, x="brand_name", y="weight")
plt.xticks(rotation=60)
plt.show()
```



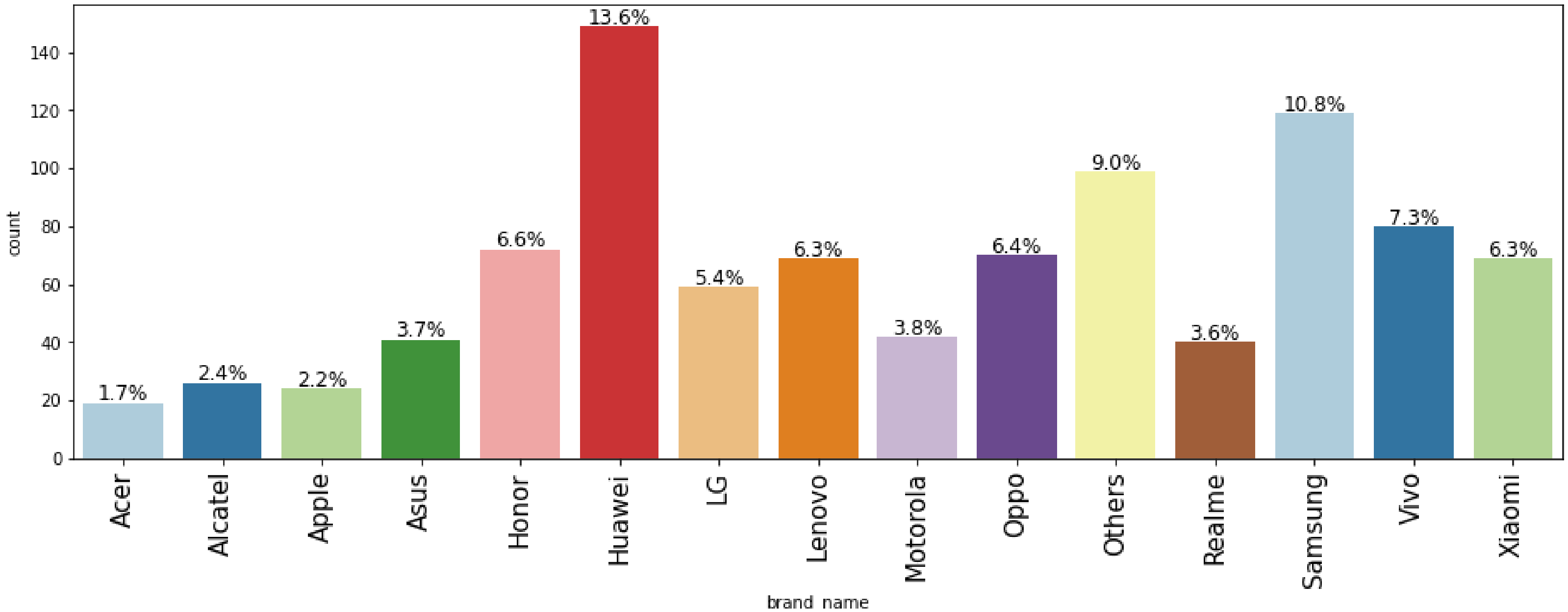
Observations

- For the subset of the data with only large batteries, the mean weights of the devices went up for most brands.
- For example, comparing it to the barplot above, the new Apple mean is 439.55 compared to 320.4. 8 brands still have a mean weight under 200.
- A lot of brands offer devices which are not very heavy but have a large battery capacity.
- Some devices offered by brands like Vivo, Realme, Motorola, etc. weigh just about 200g but offer great batteries.
- Some devices offered by brands like Huawei, Apple, Sony, etc. offer great batteries but are heavy.

In [48]:
```
# Question 5 required us to create a subset of the data, named large_screen. Use that to compare to other variables.

large_screen.brand_name.count()
```

Out[48]: 1099

In [49]:
```
bar(large_screen, "brand_name", perc=True, n=15)
```



Observations

- Huawei and Samsung offer a lot of devices suitable for customers buying phones and tablets for entertainment purposes.
- Brands like Alcatel, Acer, and Apple offer fewer devices for this customer segment.

Data Preprocessing

- Missing value treatment
- Feature engineering
- Outlier detection and treatment
- Preparing data for modeling
- Any other preprocessing steps (if needed)

Feature Engineering

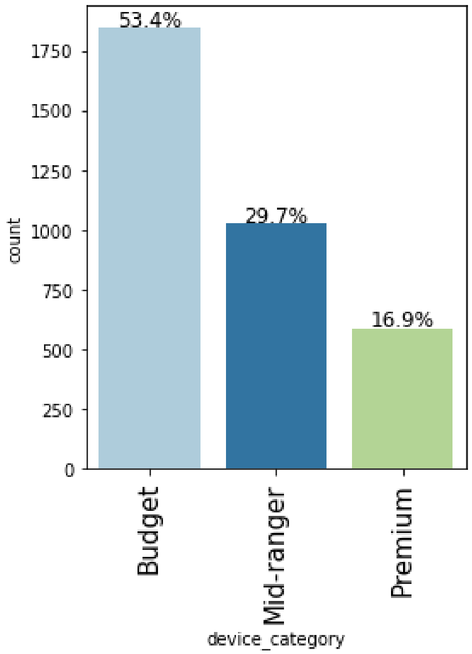
- Let's create a new column `device_category` from the `new_price` column to tag phones and tablets as budget, mid-ranger, or premium.

In [50]:
```
df2["device_category"] = pd.cut(
    x=df2.new_price,
    bins=[-np.infty, 200, 350, np.infty],
    labels=["Budget", "Mid-ranger", "Premium"],
)
```

```
df2["device_category"].value_counts()
```

Out[50]:
```
Budget        1844
Mid-ranger    1025
Premium        585
Name: device_category, dtype: int64
```

In [51]:
```
bar(df2, "device_category", perc=True)
```



- More than half the devices in the data are budget devices.

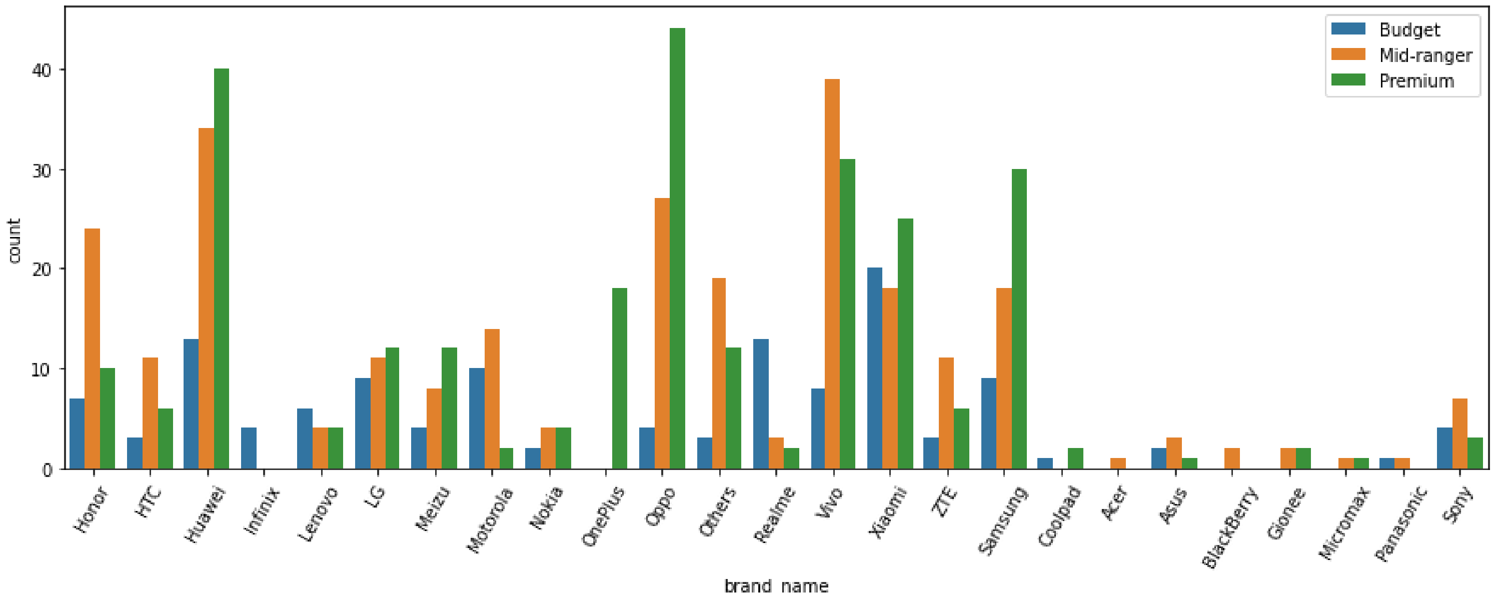
Everyone likes a good camera to capture their favorite moments with loved ones. Some customers specifically look for good front cameras to click cool selfies. Let's create a new dataframe of only those devices which are suitable for this customer segment and analyze.

In [52]:
```
selfie = df2[df2.selfie_camera_mp > 8]
selfie.shape
```

Out[52]:
```
(655, 19)
```

In [53]:
```
plt.figure(figsize=(15, 5))
sns.countplot(data=selfie, x="brand_name", hue="device_category")
plt.xticks(rotation=60)
```

```
plt.legend(loc=1)
plt.show()
```



Observations

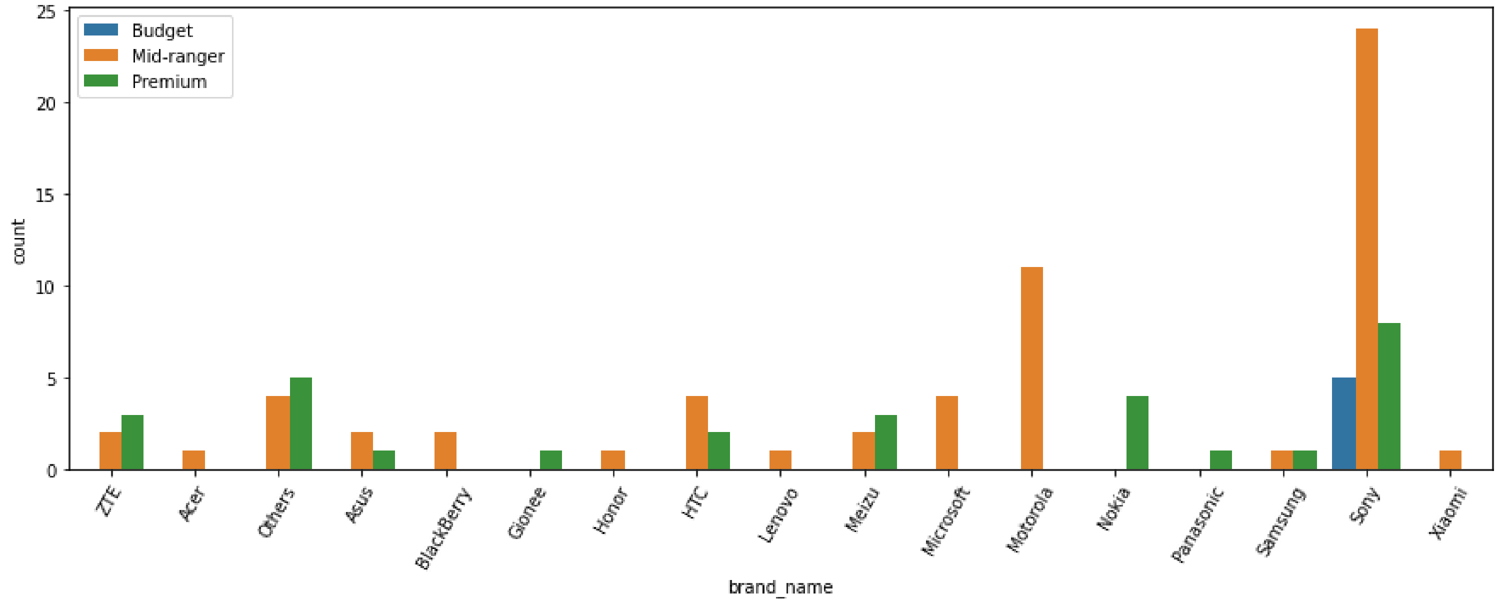
- Huawei is the go-to brand for this customer segment as they offer many devices across different price ranges with powerful front cameras.
- Xiaomi and Realme also offer a lot of budget devices capable of shooting crisp selfies.
- Oppo and Vivo offer many mid-rangers with great selfie cameras.
- Oppo, Vivo, and Samsung offer many premium devices for this customer segment.

Let's do a similar analysis for rear cameras.

In [54]:
```
main = df2[df2.main_camera_mp > 16]
main.shape
```

Out[54]: (94, 19)

In [55]:
```
plt.figure(figsize=(15, 5))
sns.countplot(data=main, x="brand_name", hue="device_category")
plt.xticks(rotation=60)
plt.legend(loc=2)
plt.show()
```



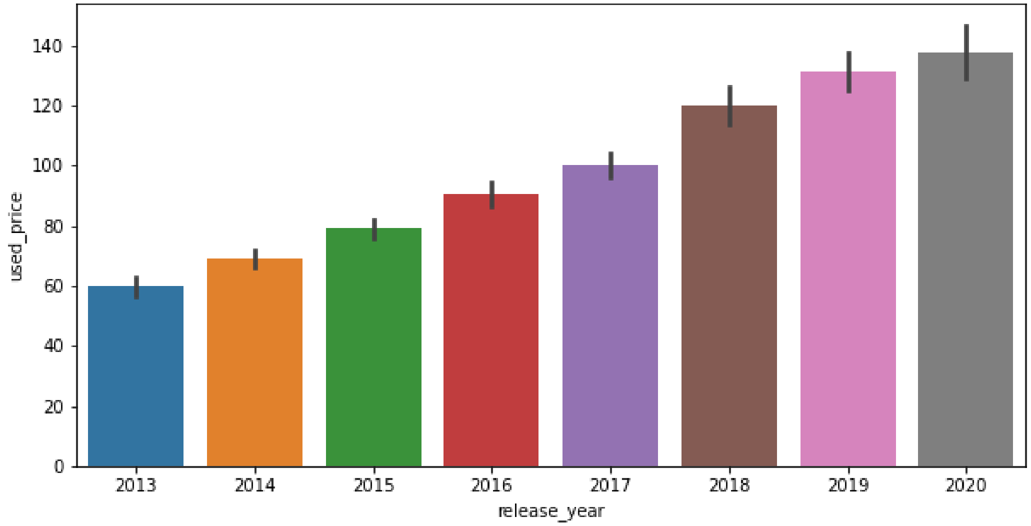
Observations

- Sony is the go-to brand for great rear cameras as they offer many devices across different price ranges.
- No brand other than Sony seems to be offering great rear cameras in budget devices.
- Brands like Motorola and HTC offer mid-rangers with great rear cameras.
- Nokia offers a few premium devices with great rear cameras.

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

In [56]:

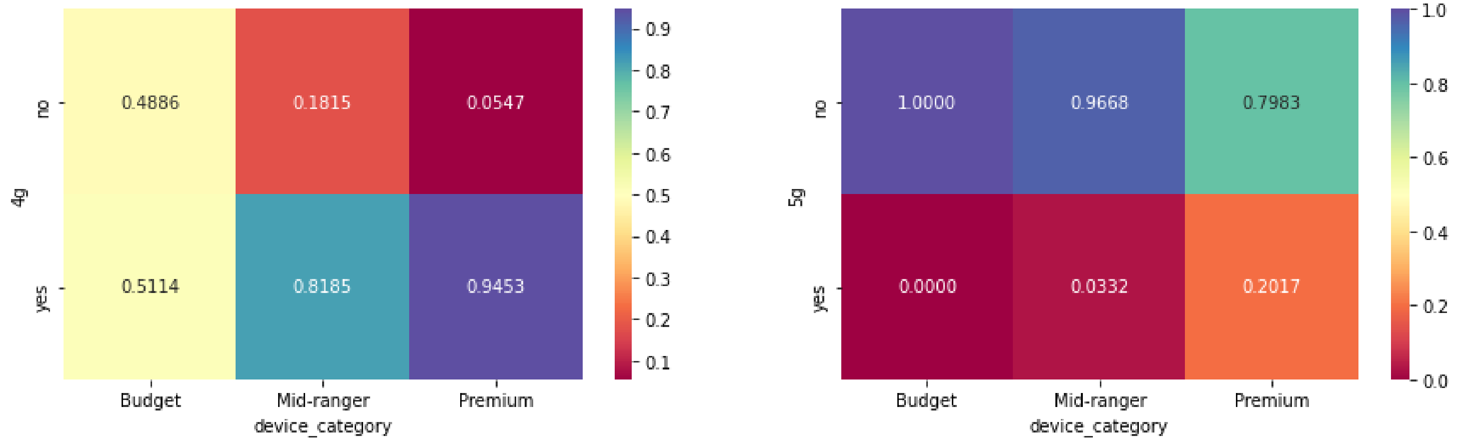
```
plt.figure(figsize=(10, 5))
sns.barplot(data=df2, x="release_year", y="used_price")
plt.show()
```



- The price of used devices has increased over the years.

Let's check the distribution of 4G and 5G phones and tablets wrt price segments.

In [57]:
```
plt.figure(figsize=(15, 4))

plt.subplot(121)
sns.heatmap(
    pd.crosstab(df2["4g"], df2["device_category"], normalize="columns"),
    annot=True,
    fmt=".4f",
    cmap="Spectral",
)

plt.subplot(122)
sns.heatmap(
    pd.crosstab(df2["5g"], df2["device_category"], normalize="columns"),
    annot=True,
    fmt=".4f",
    cmap="Spectral",
)

plt.show()
```



Observations

- There is an almost equal number of 4G and non-4G budget devices, but there are no budget devices offering 5G network.
- Most of the mid-rangers and premium devices offer 4G network.
- Very few mid-rangers (~3%) and around 20% of the premium devices offer 5G network.

Missing Value Imputation

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

In [58]:
```
df2.isnull().sum()
```

Out[58]:
```
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
new_price             0
used_price            0
used_price_log        0
new_price_log         0
weight_log            7
device_category       0
dtype: int64
```

In [59]:
```
# Impute the values of the missing entries with medians by grouping brand name and release year
```

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

for col in cols_impute:
    df2[col] = df2.groupby(["release_year", "brand_name"])[col].transform(
        lambda x: x.fillna(x.median())
    )

df2.isnull().sum()
```

Out[59]:

```
brand_name            0
os                    0
screen_size           0
4g                    0
5g                    0
main_camera_mp      179
selfie_camera_mp      2
int_memory            0
ram                   0
battery               6
weight                7
release_year          0
days_used             0
new_price             0
used_price            0
used_price_log        0
new_price_log         0
weight_log            7
device_category       0
dtype: int64
```

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

In [60]:

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

for col in cols_impute:
    df2[col] = df2.groupby(["brand_name"])[col].transform(
        lambda x: x.fillna(x.median())
    )

df2.isnull().sum()
```

Out[60]:
```
brand_name           0
os                   0
screen_size          0
4g                   0
5g                   0
main_camera_mp      10
selfie_camera_mp     0
int_memory           0
ram                  0
battery              0
weight               0
release_year         0
days_used            0
new_price            0
used_price           0
used_price_log       0
new_price_log        0
weight_log           7
device_category      0
dtype: int64
```

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

In [61]:
```
df2["main_camera_mp"] = df2["main_camera_mp"].fillna(df2["main_camera_mp"].median())
df2["weight_log"] = df2["weight_log"].fillna(df2["weight_log"].median())

df2.isnull().sum()
```

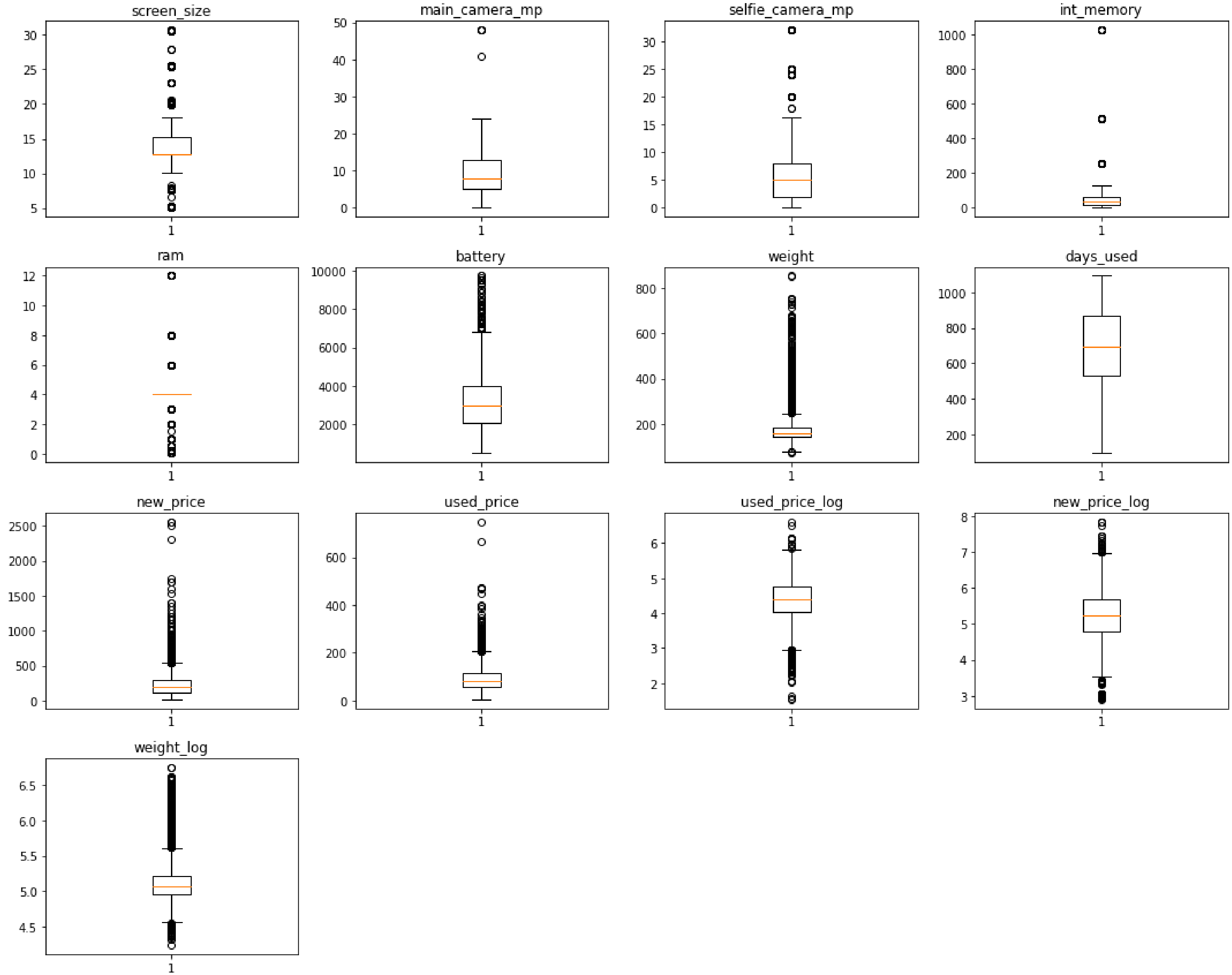
Out[61]:
```
brand_name          0
os                  0
screen_size         0
4g                  0
5g                  0
main_camera_mp      0
selfie_camera_mp    0
int_memory          0
ram                 0
battery             0
weight              0
release_year        0
days_used           0
new_price           0
used_price          0
used_price_log      0
new_price_log       0
weight_log          0
device_category     0
dtype: int64
```

- All missing values have been imputed.

Outlier Check

Check for outliers in the new data with a boxplot of all numeric variables

In [62]:
```
num_cols = df2.select_dtypes(include=np.number).columns.tolist()

# drop release_year, since it's just a year identifier (ram can stay as knowing the size is useful)

num_cols.remove("release_year")

plt.figure(figsize=(15, 12))
for i, variable in enumerate(num_cols):
    plt.subplot(4, 4, i + 1)
    plt.boxplot(df2[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



Observations

- There are quite a few outliers in the data.
- However, we will not treat them as they are proper values.

Data Preparation for Modeling

- We want to predict the used device price, so we will use the normalized version `used_price_log` for modeling.
- We will drop the `device_category` column for modeling.
- 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.

In [63]:

```
# defining the dependent and independent variables
X = df2.drop(["used_price", "used_price_log", "device_category"], axis=1)
y = df2["used_price_log"]

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

```
  brand_name       os  screen_size   4g   5g  main_camera_mp  \
0      Honor  Android       14.500  yes   no          13.000
1      Honor  Android       17.300  yes  yes          13.000
2      Honor  Android       16.690  yes  yes          13.000
3      Honor  Android       25.500  yes  yes          13.000
4      Honor  Android       15.320  yes   no          13.000

   selfie_camera_mp  int_memory   ram  battery  weight  release_year  \
0             5.000      64.000 3.000 3020.000 146.000          2020
1            16.000     128.000 8.000 4300.000 213.000          2020
2             8.000     128.000 8.000 4200.000 213.000          2020
3             8.000      64.000 6.000 7250.000 480.000          2020
4             8.000      64.000 3.000 5000.000 185.000          2020

   days_used  new_price  new_price_log  weight_log
0        127    111.620          4.715       4.984
1        325    249.390          5.519       5.361
2        162    359.470          5.885       5.361
3        345    278.930          5.631       6.174
4        293    140.870          4.948       5.220

0   4.308
1   5.162
2   5.111
3   5.135
4   4.390
Name: used_price_log, dtype: float64
```

In [64]:

```
# creating dummy variables
X = pd.get_dummies(
    X,
    columns=X.select_dtypes(include=["object", "category"]).columns.tolist(),
    drop_first=True,
```

```
)

X.head()
```

Out[64]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	new_price	...	brand_name_Spice	brand_
0	14.500	13.000	5.000	64.000	3.000	3020.000	146.000	2020	127	111.620	...	0	
1	17.300	13.000	16.000	128.000	8.000	4300.000	213.000	2020	325	249.390	...	0	
2	16.690	13.000	8.000	128.000	8.000	4200.000	213.000	2020	162	359.470	...	0	
3	25.500	13.000	8.000	64.000	6.000	7250.000	480.000	2020	345	278.930	...	0	
4	15.320	13.000	8.000	64.000	3.000	5000.000	185.000	2020	293	140.870	...	0	

5 rows × 50 columns

In [65]:

```
# Split the data in 70:30 ratio for train to test data (random_state set to 1 to validate data)

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [66]:

```
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
```

Building Our Linear Regression Model

In [67]:

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

olsmodel1 = sm.OLS(y_train, x_train1).fit()
print(olsmodel1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         used_price_log   R-squared:                       0.848
Model:                            OLS   Adj. R-squared:                  0.845
Method:                 Least Squares   F-statistic:                     263.7
Date:                Sun, 30 Jan 2022   Prob (F-statistic):               0.00
Time:                        00:45:53   Log-Likelihood:                 147.33
No. Observations:                2417   AIC:                            -192.7
Df Residuals:                    2366   BIC:                             102.6
Df Model:                          50
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
```

```
----------------------------------------------------------------------------------------
const                 -47.1230      9.266     -5.086      0.000     -65.293     -28.953
screen_size             0.0207      0.003      6.066      0.000       0.014       0.027
main_camera_mp          0.0207      0.001     13.831      0.000       0.018       0.024
selfie_camera_mp        0.0130      0.001     11.558      0.000       0.011       0.015
int_memory              0.0002   7.36e-05      2.698      0.007    5.42e-05       0.000
ram                     0.0231      0.005      4.505      0.000       0.013       0.033
battery             -2.138e-05   7.24e-06     -2.953      0.003   -3.56e-05   -7.18e-06
weight                 -0.0002      0.000     -0.652      0.514      -0.001       0.000
release_year            0.0231      0.005      5.029      0.000       0.014       0.032
days_used            4.833e-05   3.06e-05      1.580      0.114   -1.17e-05       0.000
new_price              -0.0002   5.41e-05     -2.915      0.004      -0.000   -5.16e-05
new_price_log           0.4664      0.019     24.441      0.000       0.429       0.504
weight_log              0.3587      0.059      6.085      0.000       0.243       0.474
brand_name_Alcatel      0.0192      0.047      0.406      0.685      -0.073       0.112
brand_name_Apple        0.0579      0.146      0.396      0.692      -0.229       0.344
brand_name_Asus         0.0098      0.047      0.206      0.837      -0.083       0.103
brand_name_BlackBerry  -0.0633      0.070     -0.907      0.364      -0.200       0.073
brand_name_Celkon      -0.0561      0.066     -0.854      0.393      -0.185       0.073
brand_name_Coolpad      0.0277      0.072      0.383      0.701      -0.114       0.169
brand_name_Gionee       0.0458      0.057      0.800      0.424      -0.066       0.158
brand_name_Google      -0.0192      0.084     -0.229      0.819      -0.184       0.145
brand_name_HTC         -0.0109      0.048     -0.228      0.820      -0.105       0.083
brand_name_Honor        0.0306      0.049      0.628      0.530      -0.065       0.126
brand_name_Huawei       0.0027      0.044      0.062      0.951      -0.084       0.089
brand_name_Infinix      0.1655      0.092      1.791      0.073      -0.016       0.347
brand_name_Karbonn      0.1160      0.067      1.743      0.082      -0.015       0.247
brand_name_LG          -0.0062      0.045     -0.138      0.890      -0.094       0.082
brand_name_Lava         0.0428      0.062      0.693      0.489      -0.078       0.164
brand_name_Lenovo       0.0447      0.045      0.997      0.319      -0.043       0.133
brand_name_Meizu       -0.0082      0.056     -0.148      0.882      -0.117       0.101
brand_name_Micromax    -0.0220      0.048     -0.462      0.644      -0.115       0.071
brand_name_Microsoft    0.1024      0.088      1.171      0.242      -0.069       0.274
brand_name_Motorola    -0.0123      0.049     -0.250      0.802      -0.109       0.084
brand_name_Nokia        0.0926      0.051      1.799      0.072      -0.008       0.193
brand_name_OnePlus      0.0760      0.077      0.990      0.322      -0.074       0.226
brand_name_Oppo         0.0158      0.047      0.334      0.739      -0.077       0.109
brand_name_Others      -0.0121      0.042     -0.289      0.773      -0.094       0.070
brand_name_Panasonic    0.0633      0.055      1.144      0.253      -0.045       0.172
brand_name_Realme       0.0214      0.061      0.350      0.726      -0.098       0.141
brand_name_Samsung     -0.0244      0.043     -0.570      0.569      -0.108       0.060
brand_name_Sony        -0.0616      0.050     -1.232      0.218      -0.160       0.036
brand_name_Spice       -0.0114      0.063     -0.181      0.856      -0.134       0.112
brand_name_Vivo        -0.0143      0.048     -0.298      0.765      -0.108       0.080
brand_name_XOLO         0.0205      0.054      0.377      0.707      -0.086       0.127
brand_name_Xiaomi       0.0814      0.048      1.708      0.088      -0.012       0.175
brand_name_ZTE         -0.0005      0.047     -0.011      0.991      -0.093       0.092
os_Others              -0.0028      0.034     -0.083      0.934      -0.069       0.063
os_Windows             -0.0210      0.045     -0.469      0.639      -0.109       0.067
os_iOS                 -0.0999      0.145     -0.687      0.492      -0.385       0.185
4g_yes                  0.0474      0.016      3.007      0.003       0.016       0.078
5g_yes                 -0.0618      0.031     -1.969      0.049      -0.123      -0.000
==============================================================================
Omnibus:                      189.798   Durbin-Watson:                   1.913
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              348.761
Skew:                          -0.549   Prob(JB):                     1.85e-76
Kurtosis:                       4.502   Cond. No.                     7.84e+06
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.84e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Observations

- Both the R-squared and Adjusted R squared of our model are ~0.85, indicating that it can explain ~85% of the variance in the price of used phones.
- This is a clear indication that we have been able to create a very good model which is not underfitting the data.
- To be able to make statistical inferences from our model, we will have to test that the linear regression assumptions are followed.

Model Performance Check

- We will be using metric functions defined in sklearn for RMSE and MAE.
- We will define functions to calculate MAPE.
  - The mean absolute percentage error (MAPE) measures the accuracy of predictions as a percentage, and can be calculated as the average absolute percent error for each predicted value minus actual values divided by actual values. It works best if there are no extreme values in the data and none of the actual values are 0.
- We will create a function that will print out all the above metrics in one go.

In [68]:
```
# 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)

    # computing the actual prices by using the exponential function
    target = np.exp(target)
    pred = np.exp(pred)

    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,
            "MAPE": mape,
        },
        index=[0],
    )

    return df_perf
```

In [69]:
```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel1_train_perf = model_performance_regression(olsmodel1, x_train1, y_train)
olsmodel1_train_perf
```

```
Training Performance
```

Out[69]:

	RMSE	MAE	MAPE
0	25.622	16.308	18.553

In [70]:
```
# checking model performance on test set (seen 30% data)

print("Test Performance\n")
olsmodel1_test_perf = model_performance_regression(olsmodel1, x_test1, y_test)
olsmodel1_test_perf
```

```
Test Performance
```

Out[70]:

	RMSE	MAE	MAPE
0	24.160	16.486	19.301

Observations

- RMSE and MAE of train and test data are very comparable, which indicates that our model is not overfitting the train data.
- MAE indicates that our current model is able to predict used phone prices within a mean error of ~16.5 euros on test data.
- The RMSE values are higher than the MAE values as the squares of residuals penalizes the model more for larger errors in prediction.
- Despite being able to capture 85% of the variation in the data, the MAE is around 16.5 euros as it makes larger predictions errors for the extreme values (very high or very low prices).
- MAPE of ~19.3 on the test data indicates that the model can predict within ~19.3% of the used phone 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**

TEST FOR MULTICOLINEARITY 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.

In [71]:

```
# 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 [72]:

```
checking_vif(x_train1)
```

Out[72]:

	feature	VIF
0	const	3919079.488
1	screen_size	7.880
2	main_camera_mp	2.306
3	selfie_camera_mp	2.879
4	int_memory	1.546
5	ram	2.308
6	battery	4.117

	feature	VIF
7	weight	20.239
8	release_year	5.077
9	days_used	2.663
10	new_price	5.333
11	new_price_log	7.691
12	weight_log	19.271
13	brand_name_Alcatel	3.409
14	brand_name_Apple	13.115
15	brand_name_Asus	3.334
16	brand_name_BlackBerry	1.641
17	brand_name_Celkon	1.777
18	brand_name_Coolpad	1.468
19	brand_name_Gionee	1.952
20	brand_name_Google	1.323
21	brand_name_HTC	3.412
22	brand_name_Honor	3.343
23	brand_name_Huawei	5.990
24	brand_name_Infinix	1.286
25	brand_name_Karbonn	1.578
26	brand_name_LG	4.853
27	brand_name_Lava	1.713
28	brand_name_Lenovo	4.560
29	brand_name_Meizu	2.180
30	brand_name_Micromax	3.378
31	brand_name_Microsoft	1.870
32	brand_name_Motorola	3.275
33	brand_name_Nokia	3.492
34	brand_name_OnePlus	1.437
35	brand_name_Oppo	3.972
36	brand_name_Others	9.715
37	brand_name_Panasonic	2.106

	feature	VIF
38	brand_name_Realme	1.948
39	brand_name_Samsung	7.544
40	brand_name_Sony	2.943
41	brand_name_Spice	1.695
42	brand_name_Vivo	3.652
43	brand_name_XOLO	2.139
44	brand_name_Xiaomi	3.721
45	brand_name_ZTE	3.799
46	os_Others	1.979
47	os_Windows	1.596
48	os_iOS	11.827
49	4g_yes	2.481
50	5g_yes	1.833

We will ignore the dummy variables (such as `brand_name_Apple` and `os_iOS`) that have VIFs above 5.

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 to help us do this.

In [73]:
```
def treating_multicollinearity(predictors, target, high_vif_columns):
    """
    Checking the effect of dropping the columns showing high multicollinearity
    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 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 [74]:

```
col_list = [
    "screen_size",
    "weight",
    "release_year",
    "new_price",
    "new_price_log",
    "weight_log",
]

res = treating_multicollinearity(x_train1, y_train, col_list)
res
```

Out[74]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	release_year	0.843	0.231
1	screen_size	0.842	0.232
2	weight_log	0.842	0.232
3	weight	0.838	0.235
4	new_price_log	0.806	0.257
5	new_price	0.765	0.283

Dropping "release_year" would have the would have the maximum impact on the predictive power of the model (amongst the variables being considered).

Drop `release_year` and check VIF again.

In [75]:

```
col_to_drop = "release_year"
x_train2 = x_train1.loc[:, ~x_train1.columns.str.startswith(col_to_drop)]
x_test2 = x_test1.loc[:, ~x_test1.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  release_year
```

Out[75]:

	feature	VIF
0	const	3301.635
1	screen_size	7.616
2	main_camera_mp	2.288
3	selfie_camera_mp	2.539
4	int_memory	1.542
5	ram	2.303
6	battery	3.998
7	weight	19.490
8	days_used	1.941
9	new_price	5.195
10	new_price_log	7.040
11	weight_log	19.080
12	brand_name_Alcatel	3.409
13	brand_name_Apple	13.086
14	brand_name_Asus	3.334
15	brand_name_BlackBerry	1.640
16	brand_name_Celkon	1.768
17	brand_name_Coolpad	1.468
18	brand_name_Gionee	1.952
19	brand_name_Google	1.316
20	brand_name_HTC	3.412
21	brand_name_Honor	3.341
22	brand_name_Huawei	5.989

	feature	VIF
23	brand_name_Infinix	1.285
24	brand_name_Karbonn	1.573
25	brand_name_LG	4.852
26	brand_name_Lava	1.713
27	brand_name_Lenovo	4.559
28	brand_name_Meizu	2.178
29	brand_name_Micromax	3.378
30	brand_name_Microsoft	1.865
31	brand_name_Motorola	3.274
32	brand_name_Nokia	3.462
33	brand_name_OnePlus	1.437
34	brand_name_Oppo	3.972
35	brand_name_Others	9.712
36	brand_name_Panasonic	2.106
37	brand_name_Realme	1.944
38	brand_name_Samsung	7.544
39	brand_name_Sony	2.943
40	brand_name_Spice	1.692
41	brand_name_Vivo	3.652
42	brand_name_XOLO	2.138
43	brand_name_Xiaomi	3.721
44	brand_name_ZTE	3.799
45	os_Others	1.979
46	os_Windows	1.589
47	os_iOS	11.818
48	4g_yes	2.134
49	5g_yes	1.817

In [76]:

```
col_list = ["screen_size", "weight", "new_price", "new_price_log", "weight_log"]

res = treating_multicollinearity(x_train2, y_train, col_list)
res
```

Out[76]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	weight_log	0.840	0.233
1	screen_size	0.840	0.234
2	weight	0.837	0.235
3	new_price_log	0.805	0.258
4	new_price	0.764	0.284

Drop `weight_log` next.

In [77]:

```
col_to_drop = "weight_log"
x_train3 = x_train2.loc[:, ~x_train2.columns.str.startswith(col_to_drop)]
x_test3 = x_test2.loc[:, ~x_test2.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  weight_log
```

Out[77]:

	feature	VIF
0	const	378.251
1	screen_size	7.450
2	main_camera_mp	2.287
3	selfie_camera_mp	2.537
4	int_memory	1.542
5	ram	2.299
6	battery	3.963
7	weight	6.184
8	days_used	1.924
9	new_price	5.194
10	new_price_log	7.018
11	brand_name_Alcatel	3.409
12	brand_name_Apple	13.037
13	brand_name_Asus	3.332
14	brand_name_BlackBerry	1.636
15	brand_name_Celkon	1.767

	feature	VIF
16	brand_name_Coolpad	1.467
17	brand_name_Gionee	1.952
18	brand_name_Google	1.316
19	brand_name_HTC	3.411
20	brand_name_Honor	3.341
21	brand_name_Huawei	5.989
22	brand_name_Infinix	1.285
23	brand_name_Karbonn	1.571
24	brand_name_LG	4.849
25	brand_name_Lava	1.713
26	brand_name_Lenovo	4.559
27	brand_name_Meizu	2.178
28	brand_name_Micromax	3.378
29	brand_name_Microsoft	1.865
30	brand_name_Motorola	3.273
31	brand_name_Nokia	3.453
32	brand_name_OnePlus	1.437
33	brand_name_Oppo	3.971
34	brand_name_Others	9.708
35	brand_name_Panasonic	2.105
36	brand_name_Realme	1.942
37	brand_name_Samsung	7.540
38	brand_name_Sony	2.943
39	brand_name_Spice	1.692
40	brand_name_Vivo	3.651
41	brand_name_XOLO	2.138
42	brand_name_Xiaomi	3.719
43	brand_name_ZTE	3.799
44	os_Others	1.943
45	os_Windows	1.589
46	os_iOS	11.789

	feature	VIF
47	4g_yes	2.126
48	5g_yes	1.817

In [78]:
```
col_list = ["screen_size", "weight", "new_price", "new_price_log"]

res = treating_multicollinearity(x_train3, y_train, col_list)
res
```

Out[78]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	weight	0.837	0.235
1	screen_size	0.836	0.236
2	new_price_log	0.801	0.260
3	new_price	0.758	0.287

Drop `weight` next.

In [79]:
```
col_to_drop = "weight"
x_train4 = x_train3.loc[:, ~x_train3.columns.str.startswith(col_to_drop)]
x_test4 = x_test3.loc[:, ~x_test3.columns.str.startswith(col_to_drop)]

# Check VIF now
vif = checking_vif(x_train4)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  weight
```

Out[79]:

	feature	VIF
0	const	366.716
1	screen_size	3.585
2	main_camera_mp	2.178
3	selfie_camera_mp	2.471
4	int_memory	1.539
5	ram	2.298
6	battery	3.627
7	days_used	1.845
8	new_price	5.184
9	new_price_log	7.015

	feature	VIF
10	brand_name_Alcatel	3.408
11	brand_name_Apple	13.026
12	brand_name_Asus	3.330
13	brand_name_BlackBerry	1.636
14	brand_name_Celkon	1.766
15	brand_name_Coolpad	1.467
16	brand_name_Gionee	1.952
17	brand_name_Google	1.316
18	brand_name_HTC	3.409
19	brand_name_Honor	3.339
20	brand_name_Huawei	5.989
21	brand_name_Infinix	1.283
22	brand_name_Karbonn	1.571
23	brand_name_LG	4.849
24	brand_name_Lava	1.712
25	brand_name_Lenovo	4.557
26	brand_name_Meizu	2.177
27	brand_name_Micromax	3.378
28	brand_name_Microsoft	1.864
29	brand_name_Motorola	3.269
30	brand_name_Nokia	3.452
31	brand_name_OnePlus	1.436
32	brand_name_Oppo	3.971
33	brand_name_Others	9.681
34	brand_name_Panasonic	2.105
35	brand_name_Realme	1.942
36	brand_name_Samsung	7.538
37	brand_name_Sony	2.938
38	brand_name_Spice	1.689
39	brand_name_Vivo	3.651
40	brand_name_XOLO	2.137

	feature	VIF
41	brand_name_Xiaomi	3.719
42	brand_name_ZTE	3.797
43	os_Others	1.834
44	os_Windows	1.589
45	os_iOS	11.751
46	4g_yes	2.061
47	5g_yes	1.817

In [80]:

```
col_list = ["new_price", "new_price_log"]

res = treating_multicollinearity(x_train4, y_train, col_list)
res
```

Out[80]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	new_price_log	0.798	0.263
1	new_price	0.752	0.291

Drop `new_price_log` next.

In [81]:

```
col_to_drop = "new_price_log"
x_train5 = x_train4.loc[:, ~x_train4.columns.str.startswith(col_to_drop)]
x_test5 = x_test4.loc[:, ~x_test4.columns.str.startswith(col_to_drop)]

# Check VIF now
vif = checking_vif(x_train5)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  new_price_log
```

Out[81]:

	feature	VIF
0	const	140.463
1	screen_size	3.488
2	main_camera_mp	1.984
3	selfie_camera_mp	2.416
4	int_memory	1.510
5	ram	2.278
6	battery	3.620

	feature	VIF
7	days_used	1.800
8	new_price	2.137
9	brand_name_Alcatel	3.403
10	brand_name_Apple	13.013
11	brand_name_Asus	3.329
12	brand_name_BlackBerry	1.625
13	brand_name_Celkon	1.761
14	brand_name_Coolpad	1.466
15	brand_name_Gionee	1.952
16	brand_name_Google	1.314
17	brand_name_HTC	3.402
18	brand_name_Honor	3.338
19	brand_name_Huawei	5.987
20	brand_name_Infinix	1.278
21	brand_name_Karbonn	1.567
22	brand_name_LG	4.846
23	brand_name_Lava	1.708
24	brand_name_Lenovo	4.554
25	brand_name_Meizu	2.177
26	brand_name_Micromax	3.340
27	brand_name_Microsoft	1.861
28	brand_name_Motorola	3.263
29	brand_name_Nokia	3.452
30	brand_name_OnePlus	1.436
31	brand_name_Oppo	3.969
32	brand_name_Others	9.680
33	brand_name_Panasonic	2.101
34	brand_name_Realme	1.936
35	brand_name_Samsung	7.527
36	brand_name_Sony	2.938
37	brand_name_Spice	1.678

	feature	VIF
38	brand_name_Vivo	3.650
39	brand_name_XOLO	2.135
40	brand_name_Xiaomi	3.716
41	brand_name_ZTE	3.793
42	os_Others	1.725
43	os_Windows	1.588
44	os_iOS	11.748
45	4g_yes	2.052
46	5g_yes	1.815

- **The above predictors have no multicollinearity and the assumption is satisfied.**
- **Let's check the model summary.**

In [82]:

```
olsmodel2 = sm.OLS(y_train, x_train5).fit()
print(olsmodel2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         used_price_log   R-squared:                       0.802
Model:                            OLS   Adj. R-squared:                  0.798
Method:                 Least Squares   F-statistic:                     208.3
Date:                Sun, 30 Jan 2022   Prob (F-statistic):               0.00
Time:                        00:46:00   Log-Likelihood:                -173.17
No. Observations:                2417   AIC:                             440.3
Df Residuals:                    2370   BIC:                             712.5
Df Model:                          46
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                     2.7514      0.063     43.478      0.000       2.627       2.876
screen_size               0.0528      0.003     20.327      0.000       0.048       0.058
main_camera_mp            0.0296      0.002     18.664      0.000       0.026       0.033
selfie_camera_mp          0.0178      0.001     15.136      0.000       0.016       0.020
int_memory            -6.008e-05   8.29e-05     -0.724      0.469      -0.000       0.000
ram                       0.0339      0.006      5.822      0.000       0.022       0.045
battery                1.033e-05   7.74e-06      1.335      0.182   -4.85e-06    2.55e-05
days_used              8.606e-05   2.87e-05      2.999      0.003    2.98e-05       0.000
new_price                 0.0009    3.9e-05     23.137      0.000       0.001       0.001
brand_name_Alcatel       -0.0285      0.054     -0.530      0.596      -0.134       0.077
brand_name_Apple          0.1136      0.166      0.684      0.494      -0.212       0.439
brand_name_Asus           0.0381      0.054      0.703      0.482      -0.068       0.144
brand_name_BlackBerry     0.0964      0.079      1.217      0.224      -0.059       0.252
brand_name_Celkon        -0.1419      0.075     -1.902      0.057      -0.288       0.004
brand_name_Coolpad       -0.0287      0.082     -0.349      0.727      -0.190       0.133
```

```
brand_name_Gionee          0.0521      0.065      0.798      0.425      -0.076       0.180
brand_name_Google          0.0822      0.095      0.862      0.389      -0.105       0.269
brand_name_HTC             0.0460      0.054      0.845      0.398      -0.061       0.153
brand_name_Honor           0.0090      0.056      0.162      0.871      -0.100       0.118
brand_name_Huawei         -0.0186      0.050     -0.370      0.711      -0.117       0.080
brand_name_Infinix         0.0178      0.105      0.170      0.865      -0.188       0.224
brand_name_Karbonn        -0.0004      0.076     -0.006      0.996      -0.149       0.148
brand_name_LG              0.0206      0.051      0.403      0.687      -0.080       0.121
brand_name_Lava           -0.0486      0.070     -0.691      0.490      -0.187       0.089
brand_name_Lenovo          0.0211      0.051      0.413      0.679      -0.079       0.121
brand_name_Meizu          -0.0238      0.063     -0.376      0.707      -0.148       0.100
brand_name_Micromax       -0.1494      0.054     -2.772      0.006      -0.255      -0.044
brand_name_Microsoft       0.0602      0.100      0.604      0.546      -0.135       0.255
brand_name_Motorola       -0.0466      0.056     -0.832      0.406      -0.156       0.063
brand_name_Nokia           0.1000      0.058      1.714      0.087      -0.014       0.214
brand_name_OnePlus         0.1045      0.087      1.194      0.232      -0.067       0.276
brand_name_Oppo            0.0392      0.054      0.726      0.468      -0.067       0.145
brand_name_Others          0.0026      0.047      0.055      0.956      -0.091       0.096
brand_name_Panasonic       0.0014      0.063      0.022      0.983      -0.122       0.125
brand_name_Realme         -0.0362      0.069     -0.521      0.603      -0.172       0.100
brand_name_Samsung         0.0151      0.049      0.309      0.758      -0.081       0.111
brand_name_Sony           -0.0364      0.057     -0.639      0.523      -0.148       0.075
brand_name_Spice          -0.1250      0.071     -1.755      0.079      -0.265       0.015
brand_name_Vivo           -0.0085      0.055     -0.155      0.877      -0.116       0.099
brand_name_XOLO           -0.0402      0.062     -0.650      0.516      -0.162       0.081
brand_name_Xiaomi          0.0527      0.054      0.970      0.332      -0.054       0.159
brand_name_ZTE            -0.0343      0.054     -0.641      0.521      -0.139       0.071
os_Others                 -0.1752      0.036     -4.903      0.000      -0.245      -0.105
os_Windows                -0.0060      0.051     -0.118      0.906      -0.106       0.094
os_iOS                    -0.0740      0.165     -0.448      0.655      -0.398       0.250
4g_yes                     0.0901      0.016      5.510      0.000       0.058       0.122
5g_yes                    -0.0657      0.036     -1.845      0.065      -0.136       0.004
==============================================================================
Omnibus:                      309.054   Durbin-Watson:                   1.956
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1122.494
Skew:                          -0.606   Prob(JB):                    1.79e-244
Kurtosis:                       6.111   Cond. No.                     1.78e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.78e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpreting the Regression Results:

1. **Adjusted. R-squared**: It reflects the fit of the model.
    - Adjusted R-squared values generally range from 0 to 1, where a higher value generally indicates a better fit, assuming certain conditions are met.
    - In our case, the value for adj. R-squared is 0.798, which is good!
2. ***const* coefficient**: It is the Y-intercept.
    - It means that if all the predictor variable coefficients are zero, then the expected output (i.e., Y) would be equal to the const coefficient.
    - In our case, the value for *const* coefficient is 2.7514
3. **Coefficient of a predictor variable**: It represents the change in the output Y due to a change in the predictor variable (everything else held constant).

- In our case, the coefficient of screen_size is 0.0528.

4. **std err**: It reflects the level of accuracy of the coefficients.
    - The lower it is, the higher is the level of accuracy.
5. **P>|t|**: It is the p-value.
    - For each independent feature, there is a null hypothesis and an alternate hypothesis. Here βi is the coefficient of the ith independent variable.
        - *Ho* : Independent feature is not significant (βi=0)
        - *Ha* : Independent feature is that it is significant ($\beta i \neq 0$)
    - (P>|t|) gives the p-value for each independent feature to check that null hypothesis. We are considering 0.05 (5%) as significance level.
        - A p-value of less than 0.05 is considered to be statistically significant.
6. **Confidence Interval**: It represents the range in which our coefficients are likely to fall (with a likelihood of 95%).

Observations:

- We can see that adj. R-squared has dropped from 0.845 to 0.798, which shows that the dropped columns did not have much effect on the model.
- As there is no multicollinearity, we can look at the p-values of predictor variables to check their significance.

Dropping high p-value variables

(Don't remove dummy variables unless all dummies of a column have a p-value > 0.05).

- 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 [83]:

```
# initial list of columns
cols = x_train5.columns.tolist()

# setting an initial max p-value
max_p_value = 1

while len(cols) > 0:
    # defining the train set
    x_train_aux = x_train5[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', 'main_camera_mp', 'selfie_camera_mp', 'ram', 'days_used', 'new_price', 'brand_name_Celkon', 'brand_name_Micr
omax', 'brand_name_Nokia', 'brand_name_Spice', 'os_Others', '4g_yes', '5g_yes']
```

In [84]:

```
x_train6 = x_train5[selected_features]
x_test6 = x_test5[selected_features]
```

In [85]:

```
olsmodel3 = sm.OLS(y_train, x_train6).fit()
print(olsmodel3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         used_price_log   R-squared:                       0.799
Model:                            OLS   Adj. R-squared:                  0.798
Method:                 Least Squares   F-statistic:                     734.5
Date:                Sun, 30 Jan 2022   Prob (F-statistic):               0.00
Time:                        00:46:01   Log-Likelihood:                -189.65
No. Observations:                2417   AIC:                             407.3
Df Residuals:                    2403   BIC:                             488.4
Df Model:                          13
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                   2.7217      0.041     66.921      0.000       2.642       2.801
screen_size             0.0560      0.002     34.595      0.000       0.053       0.059
main_camera_mp          0.0292      0.001     19.757      0.000       0.026       0.032
selfie_camera_mp        0.0179      0.001     16.397      0.000       0.016       0.020
ram                     0.0362      0.006      6.419      0.000       0.025       0.047
days_used            9.417e-05   2.75e-05      3.422      0.001    4.02e-05       0.000
new_price               0.0009   3.36e-05     27.206      0.000       0.001       0.001
brand_name_Celkon      -0.1580      0.058     -2.717      0.007      -0.272      -0.044
brand_name_Micromax    -0.1543      0.030     -5.081      0.000      -0.214      -0.095
brand_name_Nokia        0.0915      0.035      2.643      0.008       0.024       0.159
brand_name_Spice       -0.1270      0.056     -2.273      0.023      -0.237      -0.017
os_Others              -0.1488      0.033     -4.560      0.000      -0.213      -0.085
```

```
4g_yes                    0.0996      0.015      6.561      0.000       0.070       0.129
5g_yes                   -0.0735      0.035     -2.123      0.034      -0.141      -0.006
==============================================================================
Omnibus:                      316.251   Durbin-Watson:                   1.952
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1137.486
Skew:                          -0.624   Prob(JB):                    9.96e-248
Kurtosis:                       6.121   Cond. No.                     8.36e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.36e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [86]:

```
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel3_train_perf = model_performance_regression(olsmodel3, x_train6, y_train)
olsmodel3_train_perf
```

```
Training Performance
```

Out[86]:

	RMSE	MAE	MAPE
0	40.895	18.928	20.897

In [87]:

```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmodel3_test_perf = model_performance_regression(olsmodel3, x_test6, y_test)
olsmodel3_test_perf
```

```
Test Performance
```

Out[87]:

	RMSE	MAE	MAPE
0	26.916	17.573	20.959

Observations

- Dropping the high p-value predictor variables has not adversely affected the model performance.
- This shows that these variables do not significantly impact the target variables.

Now no feature (besides dummy variables) has a p-value greater than 0.05, so we'll consider the features in x_train6 as the final set of predictor variables and olsmodel3 as final model.

Now we'll check the rest of the assumptions on *olsmodel3*.

1. **Linearity of variables**

2. **Independence of error terms**

3. **Normality of error terms**

4. **No Heteroscedasticity**

TEST FOR LINEARITY AND INDEPENDENCE

Why the test?

- Linearity describes a straight-line relationship between two variables, predictor variables must have a linear relation with the dependent variable.
- The independence of the error terms (or residuals) is important. If the residuals are not independent, then the confidence intervals of the coefficient estimates will be narrower and make us incorrectly conclude a parameter to be statistically significant.

How to check linearity and independence?

- Make a plot of fitted values vs residuals.
- If they don't follow any 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.

How to fix if this assumption is not followed?

- We can try to transform the variables and make the relationships linear.

In [88]:

```
# 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"] = olsmodel3.fittedvalues  # predicted values
df_pred["Residuals"] = olsmodel3.resid  # residuals

df_pred.head()
```

Out[88]:

	Actual Values	Fitted Values	Residuals
3026	4.087	3.869	0.219
1525	4.448	4.558	-0.109
1128	4.315	4.270	0.045
3003	4.282	4.246	0.036
2907	4.456	4.543	-0.086

In [89]:

```
# 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()
```

- The scatter plot shows the distribution of residuals (errors) vs fitted values (predicted values).
- If there exist any pattern in this plot, we consider it as signs of non-linearity in the data and a pattern means that the model doesn't capture non-linear effects.
- **We see no pattern in the plot above. Hence, the assumptions of linearity and independence are satisfied.**

TEST FOR NORMALITY

Why the test?

- Error terms, or residuals, should be normally distributed. If the error terms are not normally distributed, confidence intervals of the coefficient estimates may become too wide or narrow. Once confidence interval becomes unstable, it leads to difficulty in estimating coefficients based on minimization of least squares. Non-normality suggests that there are a few unusual data points that must be studied closely to make a better model.

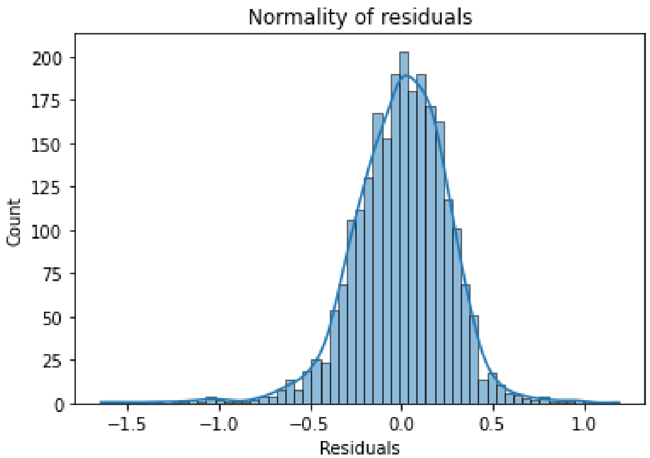
How to check normality?

- The shape of the histogram of residuals can give an initial idea about the normality.
- It can also be checked via a Q-Q plot of residuals. If the residuals follow a normal distribution, they will make a straight line plot, otherwise not.
- Other tests to check for normality includes the Shapiro-Wilk test.
  - Null hypothesis: Residuals are normally distributed
  - Alternate hypothesis: Residuals are not normally distributed

How to fix if this assumption is not followed?

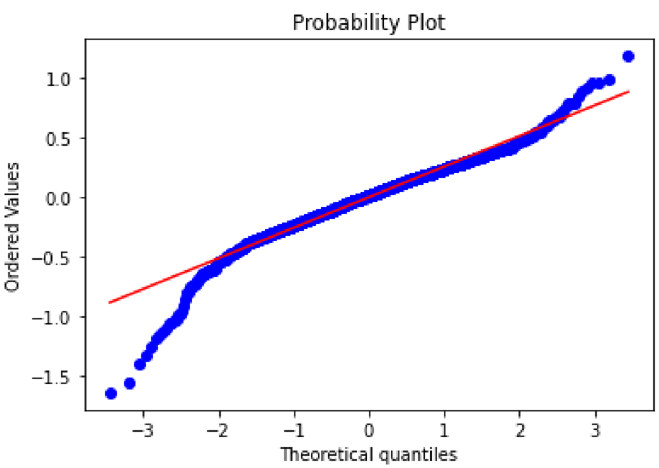
- We can apply transformations like log, exponential, arcsinh, etc. as per our data.

In [90]:
```
sns.histplot(data=df_pred, x="Residuals", kde=True)
plt.title("Normality of residuals")
plt.show()
```



- The histogram of residuals does have a bell shape.
- Let's check the Q-Q plot.

In [91]:
```
stats.probplot(df_pred["Residuals"], dist="norm", plot=pylab)
plt.show()
```



- The residuals more or less follow a straight line except for the tails.

- Let's check the results of the Shapiro-Wilk test.

In [92]:
```
stats.shapiro(df_pred["Residuals"])
```

Out[92]:
```
ShapiroResult(statistic=0.967004656791687, pvalue=4.084117499781e-23)
```

- Since p-value < 0.05, the residuals are not normal as per the Shapiro-Wilk test.
- Strictly speaking, the residuals are not normal.
- However, as an approximation, we can accept this distribution as close to being normal.
- **So, the assumption is satisfied.**

TEST FOR HOMOSCEDASTICITY

- **Homoscedascity**: If the variance of the residuals is symmetrically distributed across the regression line, then the data is said to be homoscedastic.

- **Heteroscedascity**: If the variance is unequal for the residuals across the regression line, then the data is said to be heteroscedastic.

Why the test?

- The presence of non-constant variance in the error terms results in heteroscedasticity. Generally, non-constant variance arises in presence of outliers.

How to check for homoscedasticity?

- The residual vs fitted values plot can be looked at to check for homoscedasticity. In the case of heteroscedasticity, the residuals can form an arrow shape or any other non-symmetrical shape.
- The goldfeldquandt test can also be used. If we get a p-value > 0.05 we can say that the residuals are homoscedastic. Otherwise, they are heteroscedastic.
  - Null hypothesis: Residuals are homoscedastic
  - Alternate hypothesis: Residuals have heteroscedasticity

How to fix if this assumption is not followed?

- Heteroscedasticity can be fixed by adding other important features or making transformations.

In [93]:
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(df_pred["Residuals"], x_train6)
lzip(name, test)
```

Out[93]:
```
[('F statistic', 1.0559085197297338), ('p-value', 0.17364530651957955)]
```

Since p-value > 0.05, we can say that the residuals are homoscedastic. So, this assumption is satisfied.

Now that we have checked all the assumptions of linear regression and they are satisfied, let's go ahead with prediction.

In [94]:
```
# predictions on the test set
pred = olsmodel3.predict(x_test6)

df_pred_test = pd.DataFrame({"Actual": y_test, "Predicted": pred})
df_pred_test.sample(10, random_state=1)
```

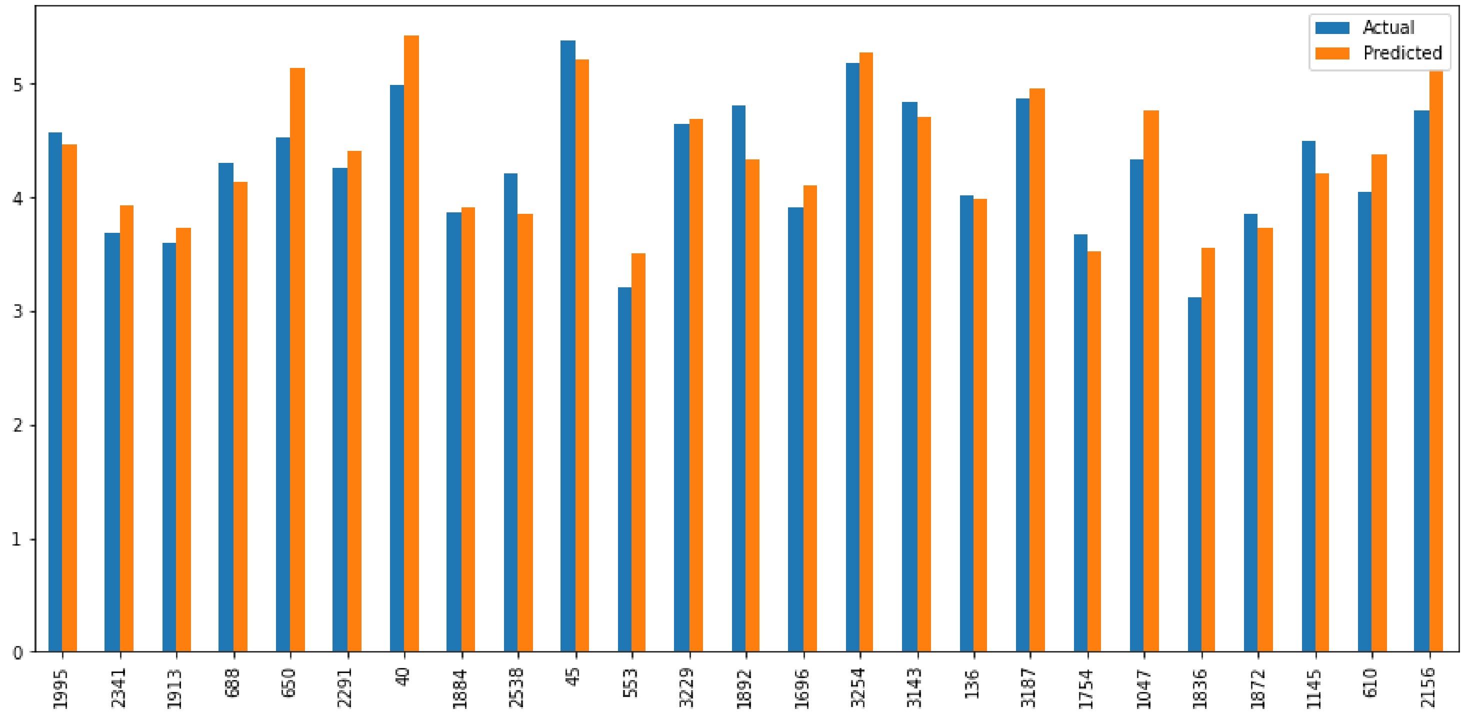
Out[94]:

	Actual	Predicted
1995	4.567	4.463
2341	3.696	3.935
1913	3.592	3.729
688	4.306	4.137
650	4.522	5.136
2291	4.259	4.406
40	4.998	5.428
1884	3.875	3.919
2538	4.207	3.854
45	5.380	5.210

- We can observe here that our model has returned good prediction results for most, and the actual and predicted values are comparable.
- We can also visualize comparison result as a bar graph.

Note: As the number of records is large, for representation purpose, we are taking a sample of 25 records only.

In [95]:
```
df3 = df_pred_test.sample(25, random_state=1)
df3.plot(kind="bar", figsize=(15, 7))
plt.show()
```



Final Model Summary

In [96]:

```
x_train_final = x_train6.copy()
x_test_final = x_test6.copy()

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

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:         used_price_log   R-squared:                       0.799
Model:                            OLS   Adj. R-squared:                  0.798
Method:                 Least Squares   F-statistic:                     734.5
Date:                Sun, 30 Jan 2022   Prob (F-statistic):               0.00
Time:                        00:46:08   Log-Likelihood:                -189.65
No. Observations:                2417   AIC:                             407.3
Df Residuals:                    2403   BIC:                             488.4
Df Model:                          13                                         
Covariance Type:            nonrobust                                         
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                     2.7217      0.041     66.921      0.000       2.642       2.801
screen_size               0.0560      0.002     34.595      0.000       0.053       0.059
main_camera_mp            0.0292      0.001     19.757      0.000       0.026       0.032
```

```
selfie_camera_mp        0.0179      0.001     16.397      0.000      0.016      0.020
ram                     0.0362      0.006      6.419      0.000      0.025      0.047
days_used            9.417e-05   2.75e-05      3.422      0.001   4.02e-05      0.000
new_price               0.0009   3.36e-05     27.206      0.000      0.001      0.001
brand_name_Celkon      -0.1580      0.058     -2.717      0.007     -0.272     -0.044
brand_name_Micromax    -0.1543      0.030     -5.081      0.000     -0.214     -0.095
brand_name_Nokia        0.0915      0.035      2.643      0.008      0.024      0.159
brand_name_Spice       -0.1270      0.056     -2.273      0.023     -0.237     -0.017
os_Others              -0.1488      0.033     -4.560      0.000     -0.213     -0.085
4g_yes                  0.0996      0.015      6.561      0.000      0.070      0.129
5g_yes                 -0.0735      0.035     -2.123      0.034     -0.141     -0.006
==============================================================================
Omnibus:                      316.251   Durbin-Watson:                   1.952
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1137.486
Skew:                          -0.624   Prob(JB):                    9.96e-248
Kurtosis:                       6.121   Cond. No.                     8.36e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.36e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [97]:

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

Training Performance

Out[97]:

	RMSE	MAE	MAPE
0	40.895	18.928	20.897

In [98]:

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

Test Performance

Out[98]:

	RMSE	MAE	MAPE
0	26.916	17.573	20.959

Actionable Insights

- The model explains ~80% of the variation in the data and can predict within 17.6 euros of the used device price.
- The most significant predictors of the used device price are the price of a new device of the same model, the size of the devices screen, the resolution of the rear and front cameras, the number of days it was used, the amount of RAM, and the availability of 4G and 5G network.
- A unit increase in new model price will result in a 0.09% increase in the used device price. *[100* {exp(0.0009) - 1} = 0.09]*
- A unit increase in size of the device's screen will result in a 5.76% increase in the used device price. *[100* {exp(0.0560) - 1} = 5.76]*
- A unit increase in the amount of RAM will result in a 3.69% increase in the used device price. *[100* {exp(0.0362) - 1} = 3.69]*

Business Recommendations

- The model can predict the used device price within ~21%, which is not bad, and can be used for predictive purposes.
- ReCell should look to attract people who want to sell used phones and tablets which have not been used for many days and have good front and rear camera resolutions.
- Devices with larger screens and more RAM are also good candidates for reselling to certain customer segments.
- They should also try to gather and put up phones having a high price for new models to try and increase revenue.
  - They can focus on volume for the budget phones and offer discounts during festive sales on premium phones.
- Additional data regarding customer demographics (age, gender, income, etc.) can be collected and analyzed to gain better insights into the preferences of customers across different segments.
- ReCell can also look to sell other used gadgets, like smart watches, which might attract certain segments of customers.