Supervised Learning - Foundations Project: ReCell

Problem Statement

Business Context

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

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

Objective

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

Data Description

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

- brand_name: Name of manufacturing brand
- os: OS on which the device runs

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

Importing necessary libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import statsmodels.api as sm

pd.set_option("display.max_columns", None)

pd.set_option("display.max_rows", 200)
```

Loading the dataset

```
1 df = pd.read_csv('/content/drive/MyDrive/Supervised Learning - Foundations/used_device_da
```

Data Overview

- Observations
- Sanity checks

```
1 df.head()
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memo
0	Honor	Android	14.50	yes	no	13.0	5.0	64
1	Honor	Android	17.30	yes	yes	13.0	16.0	128
2	Honor	Android	16.69	yes	yes	13.0	8.0	128
3	Honor	Android	25.50	yes	yes	13.0	8.0	64
4	Honor	Android	15.32	yes	no	13.0	8.0	64

```
1 df.tail()
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_me
3449	Asus	Android	15.34	yes	no	NaN	8.0	
3450	Asus	Android	15.24	yes	no	13.0	8.0	
3451	Alcatel	Android	15.80	yes	no	13.0	5.0	
3452	Alcatel	Android	15.80	yes	no	13.0	5.0	
3453	Alcatel	Android	12.83	yes	no	13.0	5.0	

```
1 df.shape
```

```
(3454, 15)
```

Observation:

- There are 3454 rows and 15 columns in the data

```
1 data = df.copy()
```

```
1 data.info()
```

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

Observation:

- The release_year and days_used columns are of int64 data type. There are four object type columns and the rest are all floats.
- From the non-null count, we can see that some columns have missing values as their total count does not total up to the total number of rows in the data frame

```
1 data.isnull().sum().sort_values(ascending=False)
```

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

```
normalized_used_price      0
normalized_new_price       0
dtype: int64
```

Observations:

- main_camera_mp column has the most missing values that total up to 179.
- int_memory and ram have the same missing number of missing values
- The target variable does not have missing values which is a good thing. Meaning we shall not lose any data by dropping rows.

```
missing_main_camera = data[data["main_camera_mp"].isnull()]
missing_main_camera.head()
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memo
59	Infinix	Android	17.32	yes	no	NaN	8.0	3
60	Infinix	Android	15.39	yes	no	NaN	8.0	6
61	Infinix	Android	15.39	yes	no	NaN	8.0	3
62	Infinix	Android	15.39	yes	no	NaN	16.0	3
63	Infinix	Android	15.29	yes	no	NaN	16.0	3

```
missing_main_camera["os"].value_counts()
```

```
Android    179
Name: os, dtype: int64
```

```
missing_main_camera["brand_name"].value_counts()
```

```
Realme        36
Xiaomi        23
Oppo          20
Motorola      18
OnePlus       17
Meizu         15
Vivo          14
Infinix       10
Sony           7
ZTE            4
Asus           4
BlackBerry     4
Coolpad        3
Lava           2
```

```
Panasonic      2
Name: brand_name, dtype: int64
```

Observations

- All devices that are missing main camera pixels values are Android devices but are of different brands.

```
1 missing_selfie_camera = data[data["selfie_camera_mp"].isnull()]
2 missing_selfie_camera.head(10)
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_me
1080	Google	Android	15.32	yes	no	12.2	NaN	
1081	Google	Android	12.83	yes	no	12.2	NaN	

Observations

- There are only two missing values in the selfie_camera_mp column
- All the devices have the same brand name Google and the same operating system Android

```
1 missing_ram = data[data["ram"].isnull()]
2 missing_ram.head()
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memo
114	Nokia	Others	5.18	no	no	0.3	0.0	0
335	Nokia	Others	5.18	no	no	0.3	0.0	0
2059	Nokia	Others	5.18	no	no	0.3	0.0	0
2090	Nokia	Others	7.62	no	no	5.0	0.0	0

Observations:

- There are four missing values of RAM
- All the four missing values the Nokia Brand

```
1 missing_weight= data[data["weight"].isnull()]
2 missing_weight.head(10)
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_m
3002	XOLO	Android	12.70	yes	no	13.0	5.0	
3003	XOLO	Android	12.83	yes	no	8.0	5.0	
3004	XOLO	Android	12.70	no	no	8.0	2.0	
3005	XOLO	Android	10.29	no	no	5.0	0.3	
3006	XOLO	Android	12.70	no	no	5.0	0.3	
3007	XOLO	Windows	12.70	no	no	8.0	2.0	
3008	XOLO	Android	12.70	no	no	8.0	5.0	

Observations:

- All weight missing values are for brandname XOLO and Windows and Android OS devices

```
1 missing_battery = data[data["battery"].isnull()]
2 missing_battery.head()
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_m
1829	Meizu	Android	12.83	yes	no	13.0	5.0	
1831	Meizu	Android	12.83	yes	no	20.7	5.0	
1832	Meizu	Android	13.61	yes	no	20.7	2.0	
1962	Microsoft	Windows	25.55	no	no	5.0	3.5	
2058	Nokia	Others	5.18	no	no	0.3	0.0	

Observations:

- Most of the missing values are from Android Devices
- The brand names are different for most of the missing values.
- From my view, since companies tend to improve devices for every release year, to impute these missing values, I will group the devices according to brand name and year of release with the medians of the columns during data preprocessing

```
1 data.nunique()
```

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

Observation:

- There are 34 different brand names
- There are 4 different operating systems
- Release_ year columns has 8 different values.

```
1 data["brand_name"].value_counts()
```

```
Others        502
Samsung       341
Huawei        251
LG            201
Lenovo        171
ZTE           140
Xiaomi        132
Oppo          129
Asus          122
Alcatel       121
```

```
Micromax      117
Vivo          117
Honor         116
HTC           110
Nokia         106
Motorola      106
Sony           86
Meizu          62
Gionee         56
Acer           51
XOLO           49
Panasonic      47
Realme         41
Apple          39
Lava           36
Celkon         33
Spice          30
Karbonn        29
Coolpad        22
BlackBerry     22
Microsoft      22
OnePlus        22
Google         15
Infinix        10
Name: brand_name, dtype: int64
```

Observation:

- The brand name column has very many categories.
- In my opinion, during data preprocessing, I will have to drop this column since this information is summarized by another column called "os".
- The os column has fewer categories as seen below

```
1 data["os"].value_counts()
```

```
Android    3214
Others      137
Windows      67
iOS          36
Name: os, dtype: int64
```

Observation:

- Android operating system has the most count in the data frame. This implies that more Android phones are refurbished

```
1 df.describe(include = ['category']).T
```

	count	unique	top	freq
brand_name	3454	34	Others	502
os	3454	4	Android	3214
4g	3454	2	yes	2335
5g	3454	2	no	3302

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Questions:

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

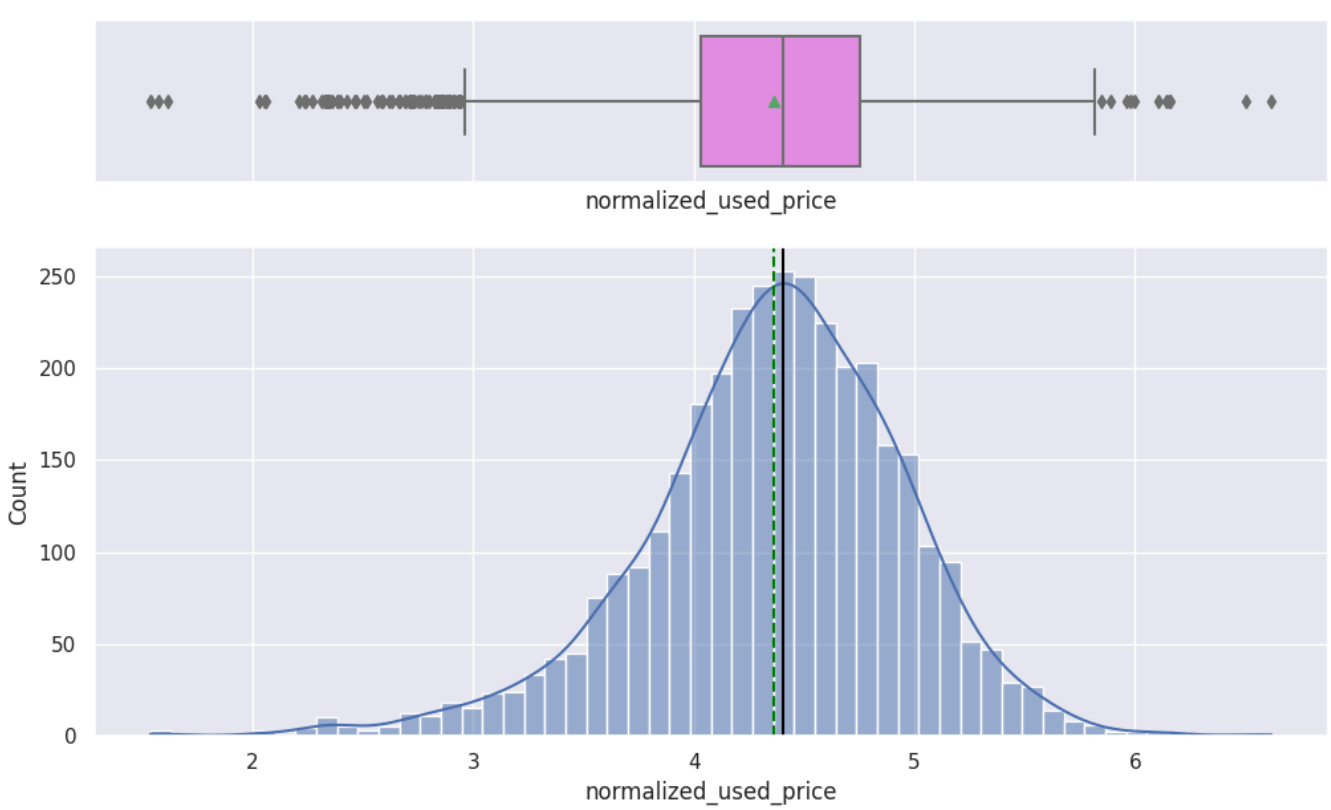
```
1 data.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	
brand_name	3454	34	Others	502	NaN	NaN	NaN	
os	3454	4	Android	3214	NaN	NaN	NaN	
screen_size	3454.0	NaN	NaN	NaN	13.713115	3.80528	5.08	
4g	3454	2	yes	2335	NaN	NaN	NaN	
5g	3454	2	no	3302	NaN	NaN	NaN	
main_camera_mp	3275.0	NaN	NaN	NaN	9.460208	4.815461	0.08	
selfie_camera_mp	3452.0	NaN	NaN	NaN	6.554229	6.970372	0.0	
int_memory	3450.0	NaN	NaN	NaN	54.573099	84.972371	0.01	
ram	3450.0	NaN	NaN	NaN	4.036122	1.365105	0.02	
battery	3448.0	NaN	NaN	NaN	3133.402697	1299.682844	500.0	
weight	3447.0	NaN	NaN	NaN	182.751871	88.413228	69.0	
release_year	3454.0	NaN	NaN	NaN	2015.965258	2.298455	2013.0	
days_used	3454.0	NaN	NaN	NaN	674.869716	248.580166	91.0	
normalized_used_price	3454.0	NaN	NaN	NaN	4.364712	0.588914	1.536867	4
normalized_new_price	3454.0	NaN	NaN	NaN	5.233107	0.683637	2.901422	4

Univariate Analysis

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

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """

    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x='normalized_used_price', ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x='normalized_used_price', kde=True, ax=ax_hist2, bins=bins, palette='
    ) if bins else sns.histplot(
        data=data, x='normalized_used_price', kde=True, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data['normalized_used_price'].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data['normalized_used_price'].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

```
histogram_boxplot(df, "normalized_used_price")
```

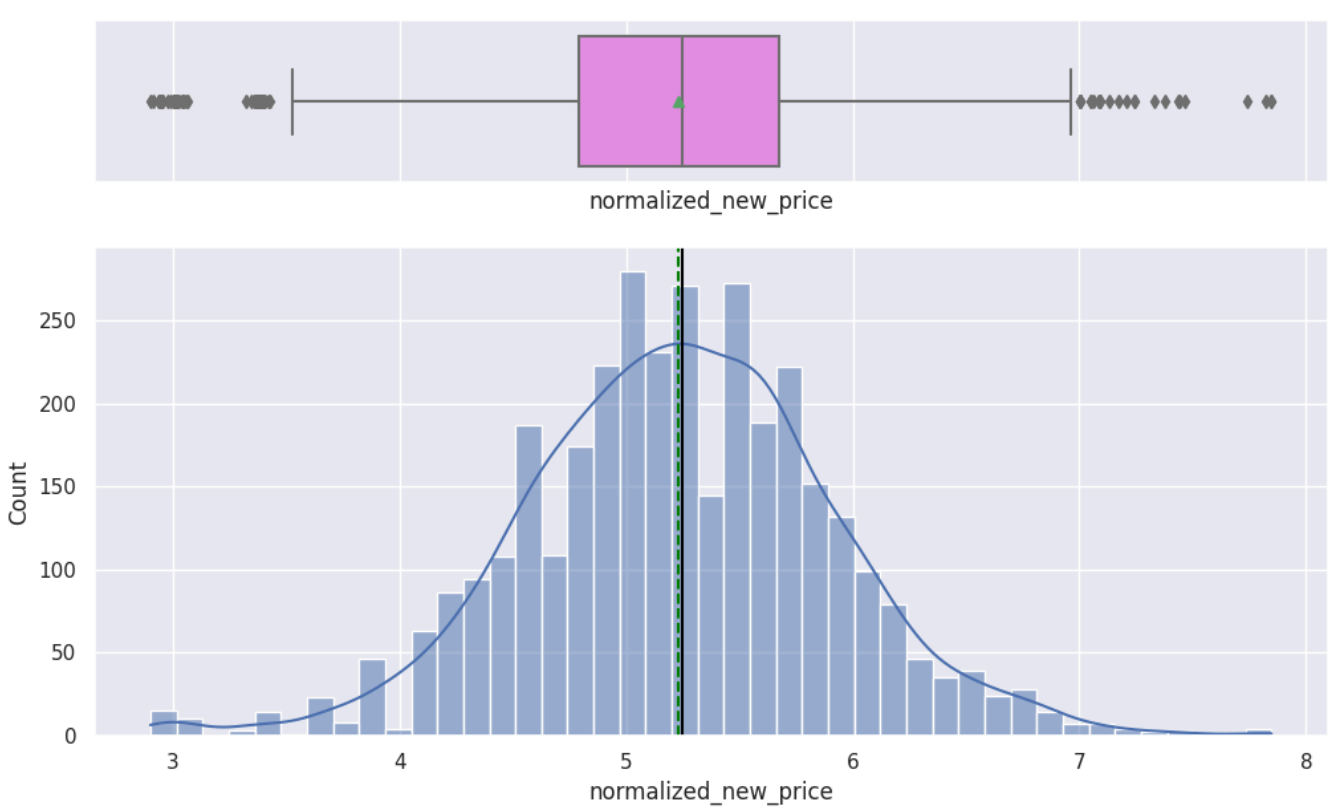


Observations

- The distribution of used phone price is a normal distribution

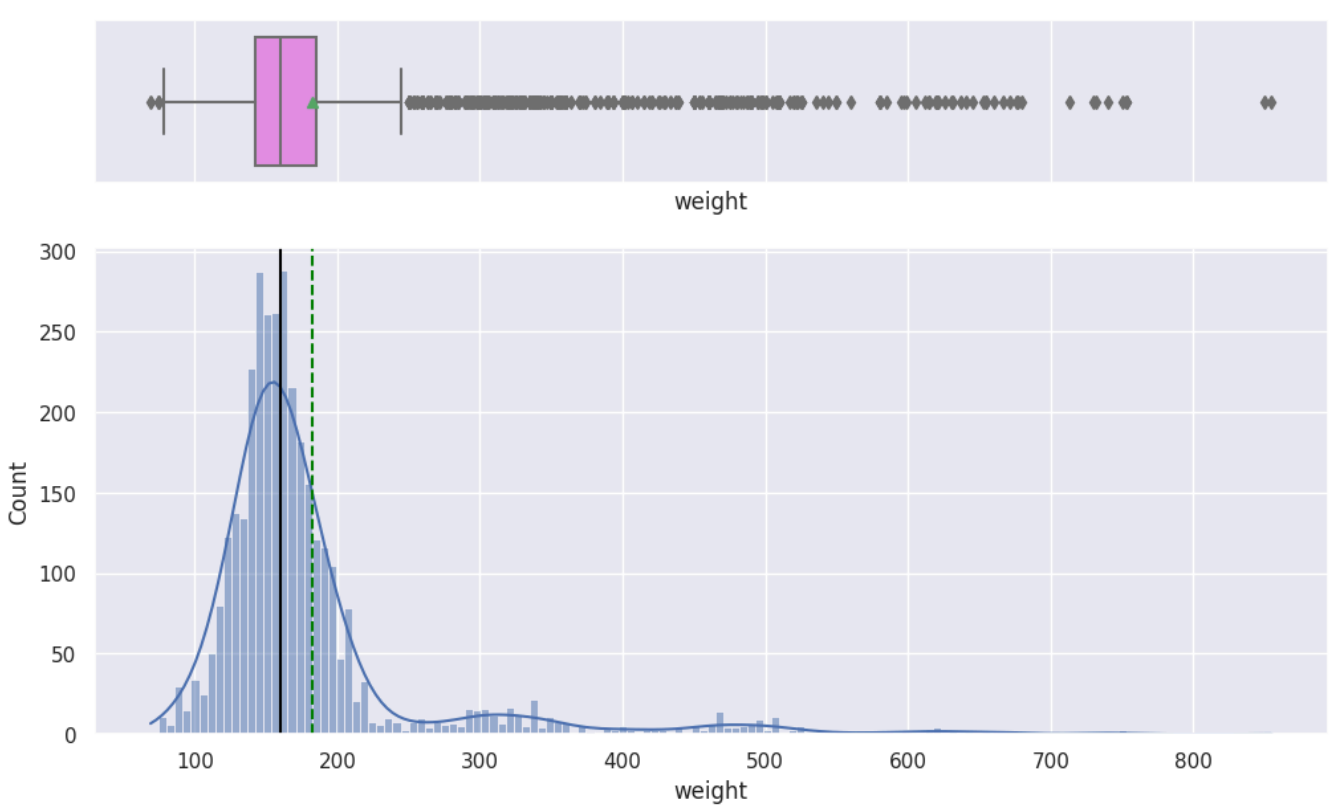
```
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """

    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x='normalized_new_price', ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x='normalized_new_price', kde=True, ax=ax_hist2, bins=bins, palette="s
    ) if bins else sns.histplot(
        data=data, x='normalized_new_price', kde=True, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data['normalized_new_price'].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data['normalized_new_price'].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

```
histogram_boxplot(df,'normalized_new_price')
```


normalized_new_price
Count
250
200
150
100
50
0
3
4
5
6
7
8
normalized_new_price

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

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """

    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x='weight', ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x='weight', kde=True, ax=ax_hist2, bins=bins, palette="summer"
    ) if bins else sns.histplot(
        data=data, x='weight', kde=True, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data['weight'].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data['weight'].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

```
histogram_boxplot(df, 'weight')
```

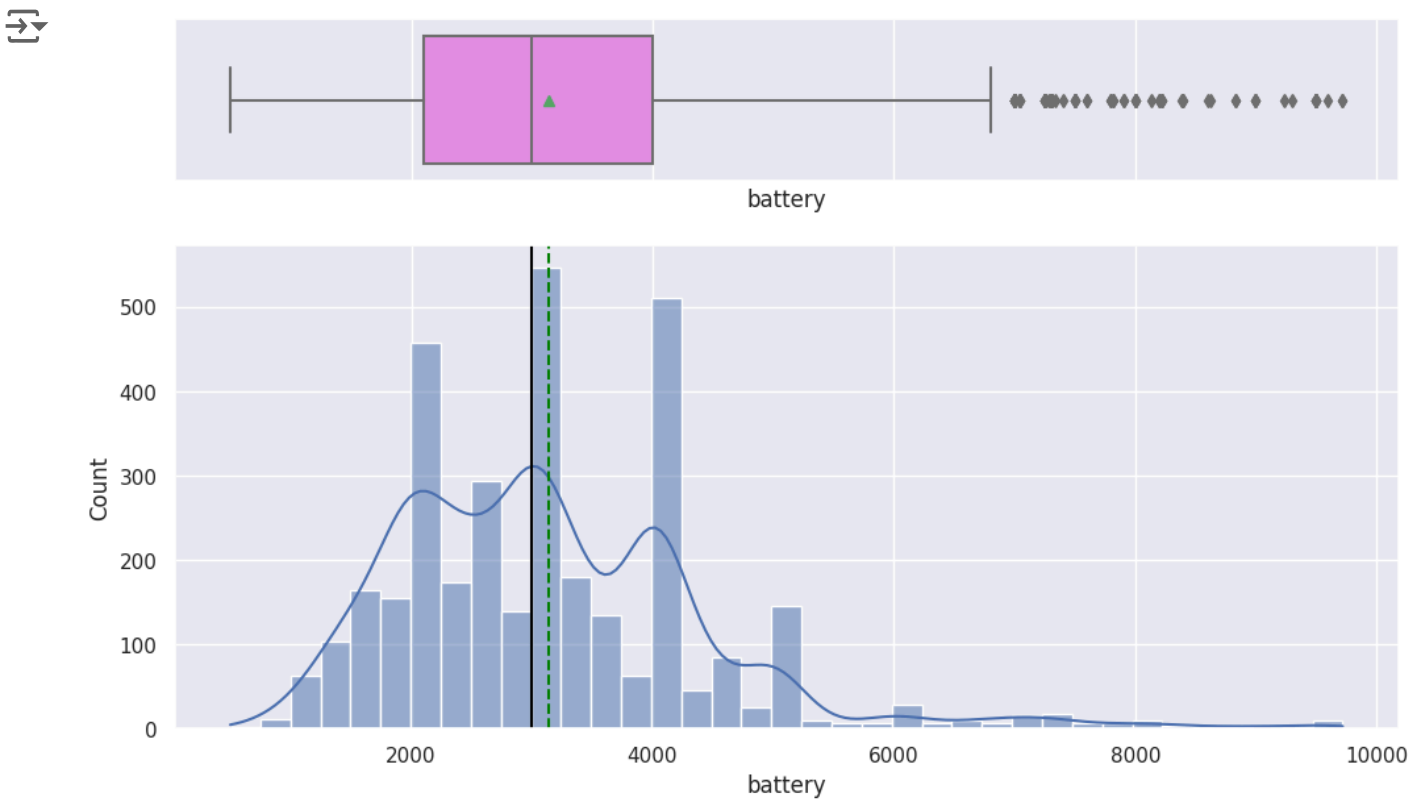


Observation:

- The distribution is right skewed. Distribution has many outliers on the right side

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

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """

    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x='battery', ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x='battery', kde=True, ax=ax_hist2, bins=bins, palette="summer"
    ) if bins else sns.histplot(
        data=data, x='battery', kde=True, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data['battery'].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data['battery'].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

```
histogram_boxplot(df, 'battery')
```

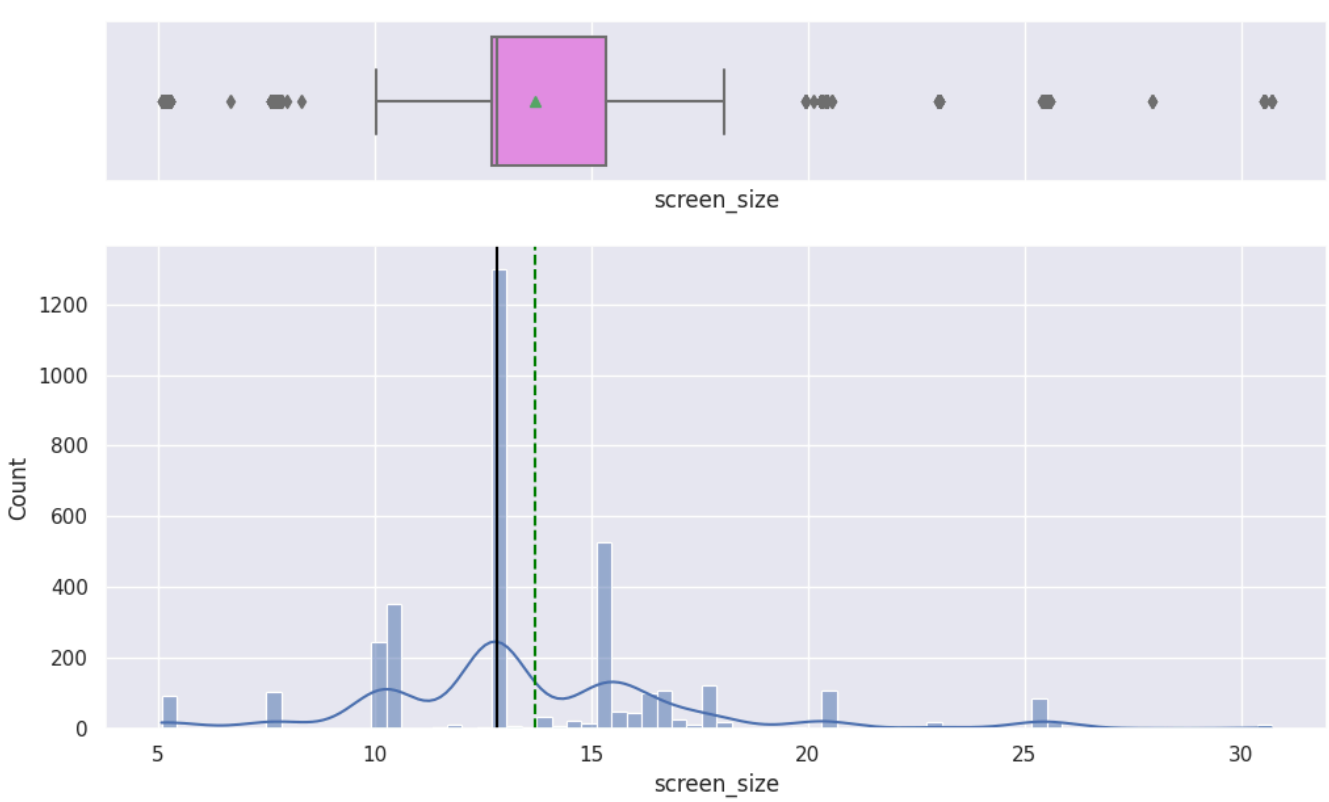


Observation:

Battery is almost a normal distribution It has some outliers It has a median and mean around 3000 mAh

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

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """

    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x='screen_size', ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x='screen_size', kde=True, ax=ax_hist2, bins=bins, palette="summer"
    ) if bins else sns.histplot(
        data=data, x='screen_size', kde=True, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data['screen_size'].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data['screen_size'].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

```
histogram_boxplot(df, 'screen_size')
```

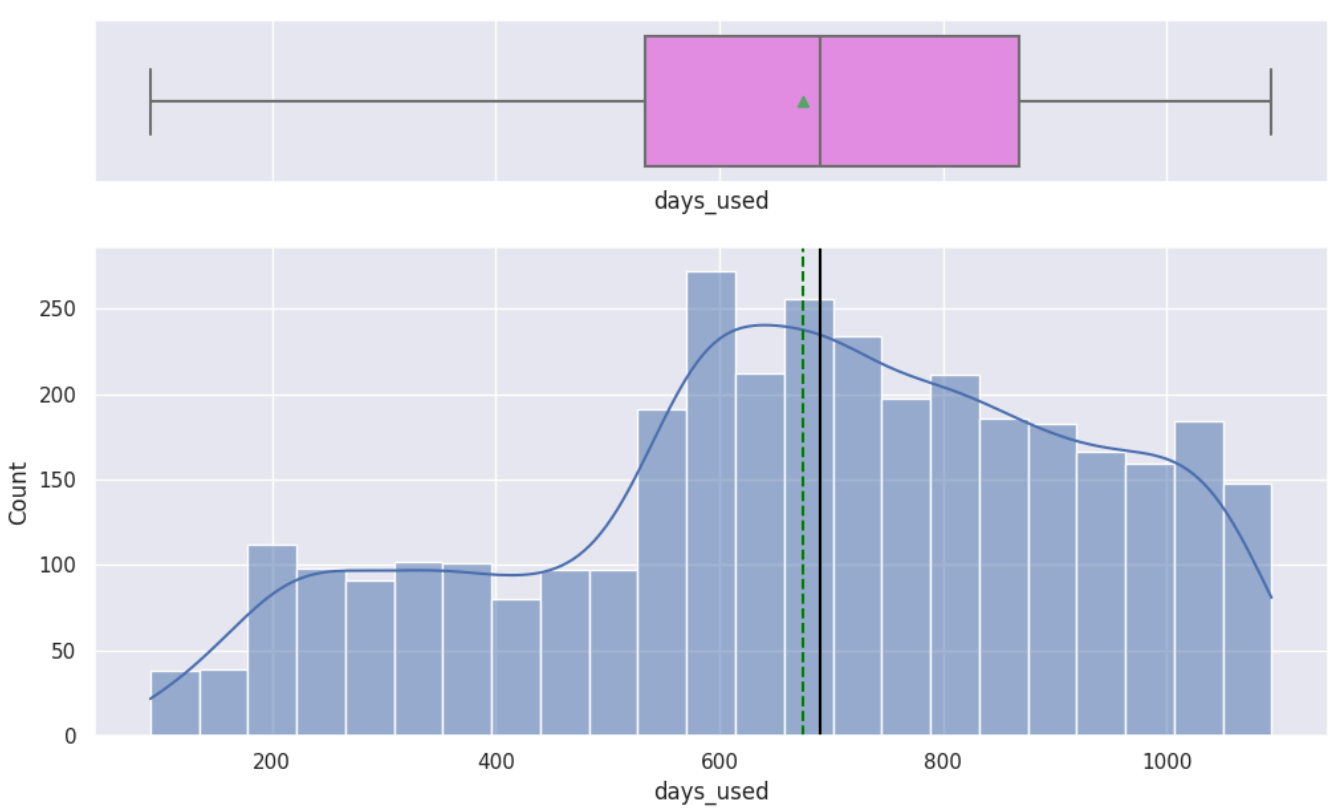


Observation

- The distribution is een to be right skewed with outliers on both sides and a median about 15cm.

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

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """

    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x='days_used', ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x='days_used', kde=True, ax=ax_hist2, bins=bins, palette="summer"
    ) if bins else sns.histplot(
        data=data, x='days_used', kde=True, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data['days_used'].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data['days_used'].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

```
histogram_boxplot(df, 'days_used')
```

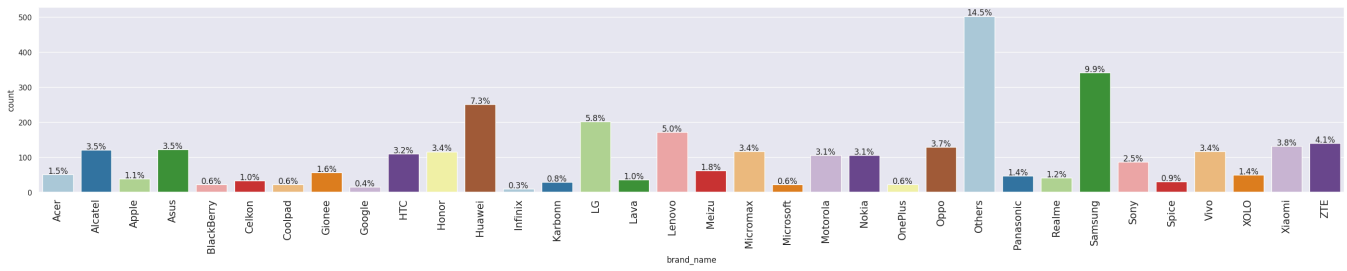


Observations:

- This distribution is skewed to the left and has a median of about 695 and a mean around 680 days.

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

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

    total = len(data[feature])
    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_cou

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(100 * p.get_height() / total)
        else:
            label = p.get_height()  #

        x = p.get_x() + p.get_width() / 2
        y = p.get_height()

        ax.annotate(label,(x, y),ha="center",va="center",size=12,xytext=(0, 5),textcoords
```

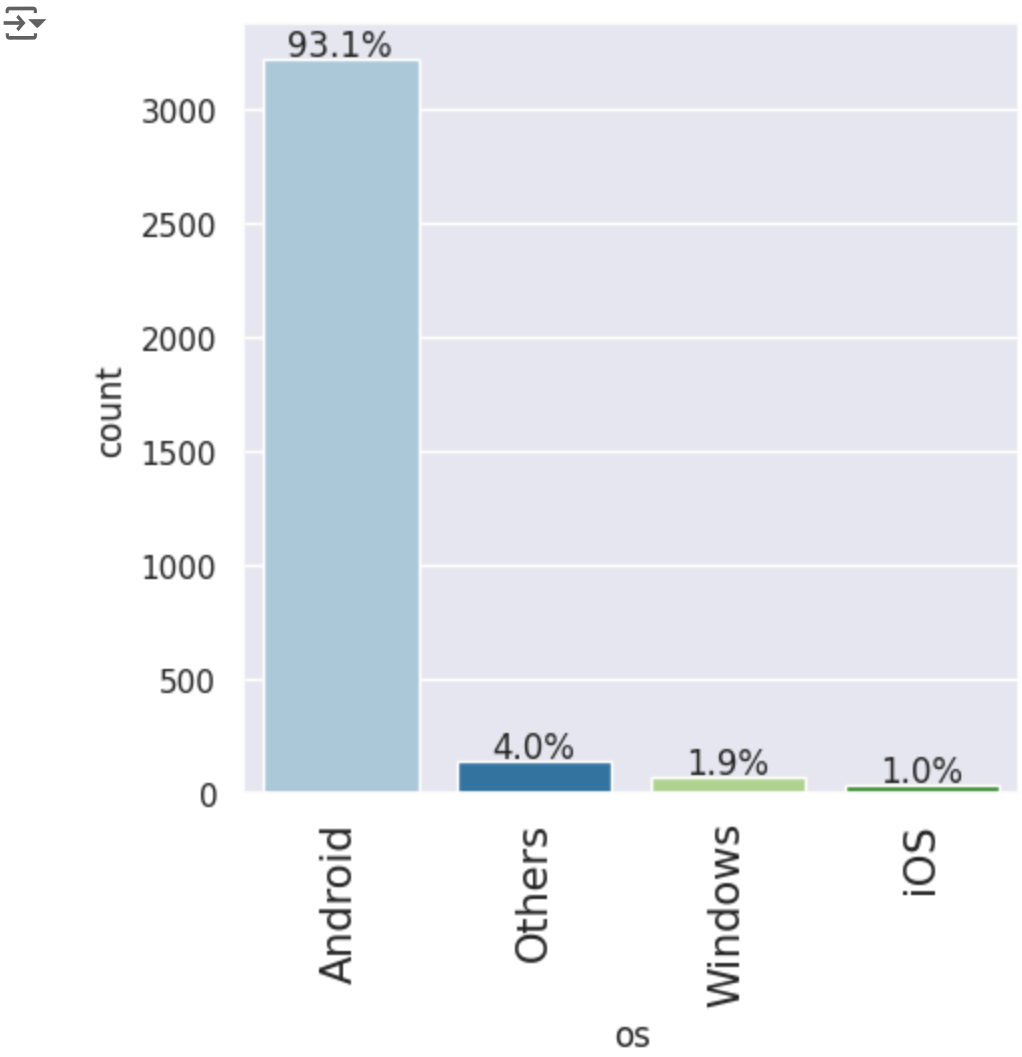
```
labeled_barplot(df, "brand_name", perc=True)
```



Observations:

- Most of the phones in the dataset are those with no known brand name Samsung is the highest represented known brand followed by Huawei.

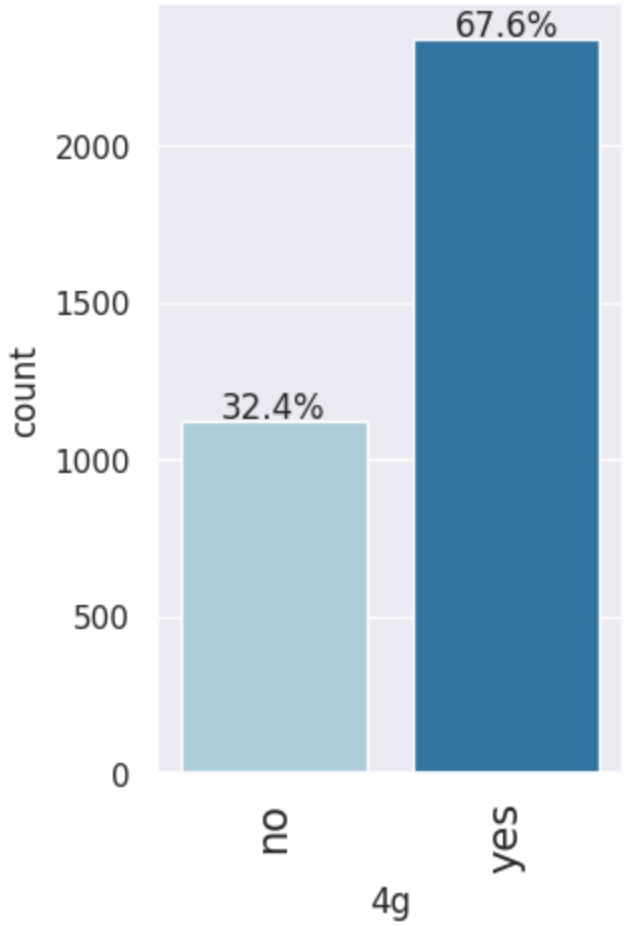
```
1 labeled_barplot(df, "os", perc=True)
```



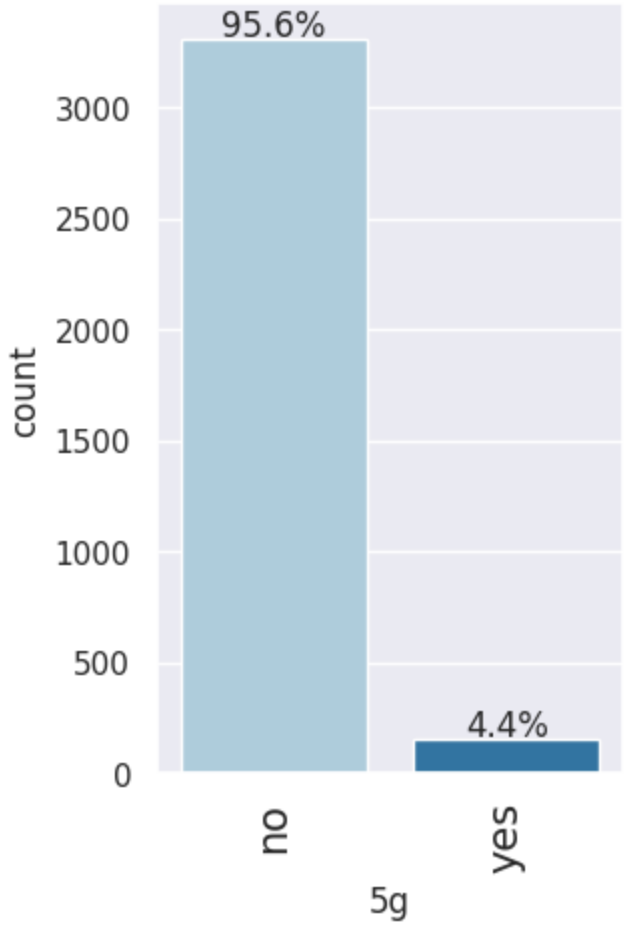
Observations:

- Android has the highest percentage at 93.1% while IOS has the least percentage at 1.0%

```
1 labeled_barplot(df, "4g", perc=True)
```



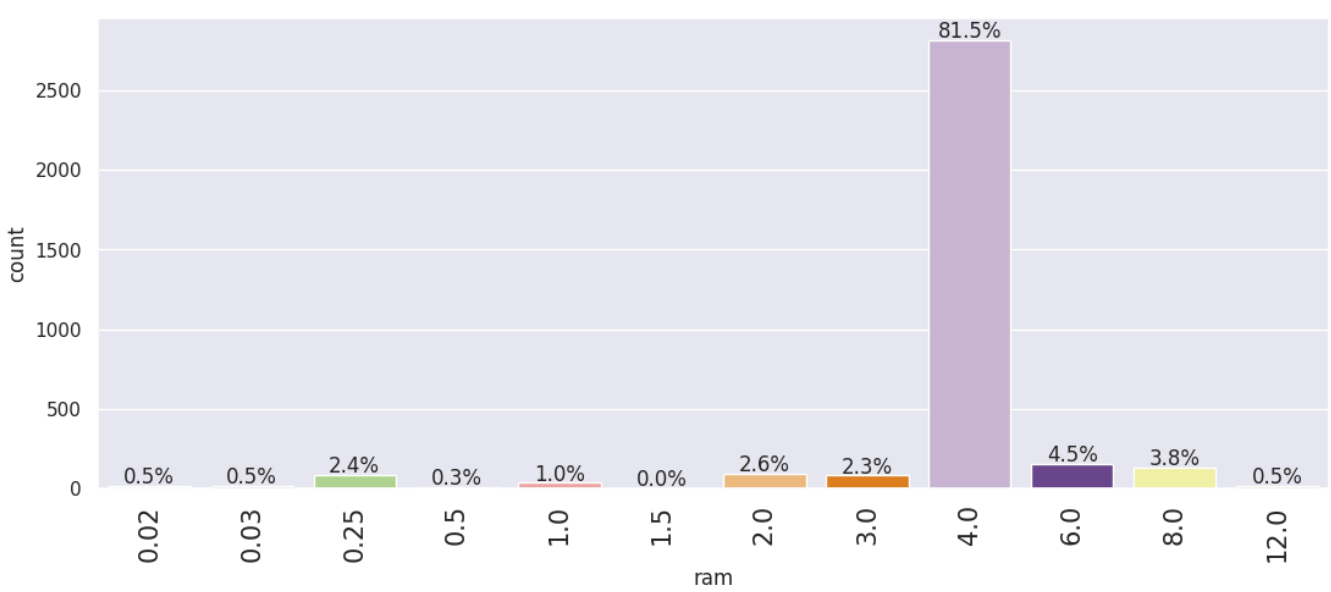

```
1 labeled_barplot(df, "5g", perc=True)
```



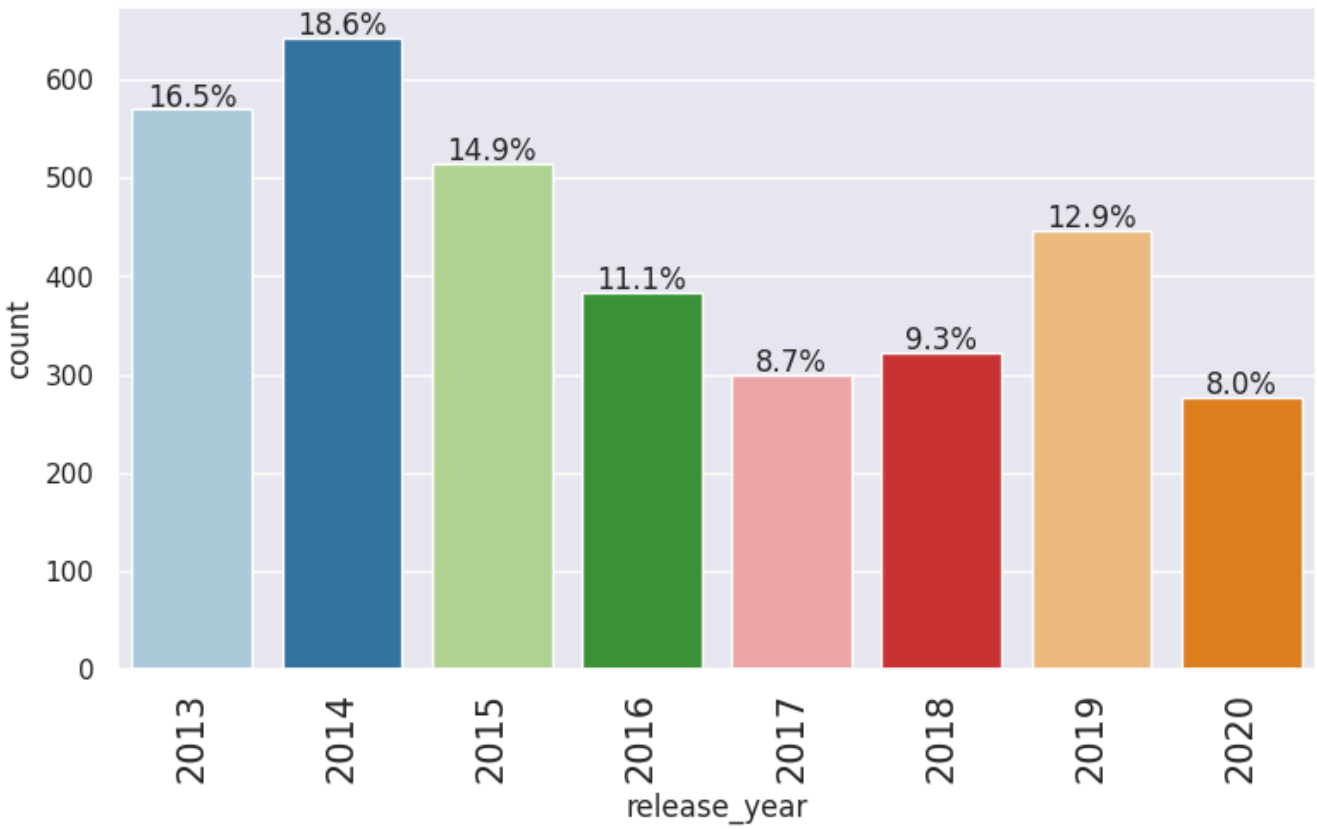

```
1 df["ram"].nunique()
```

```
12
```

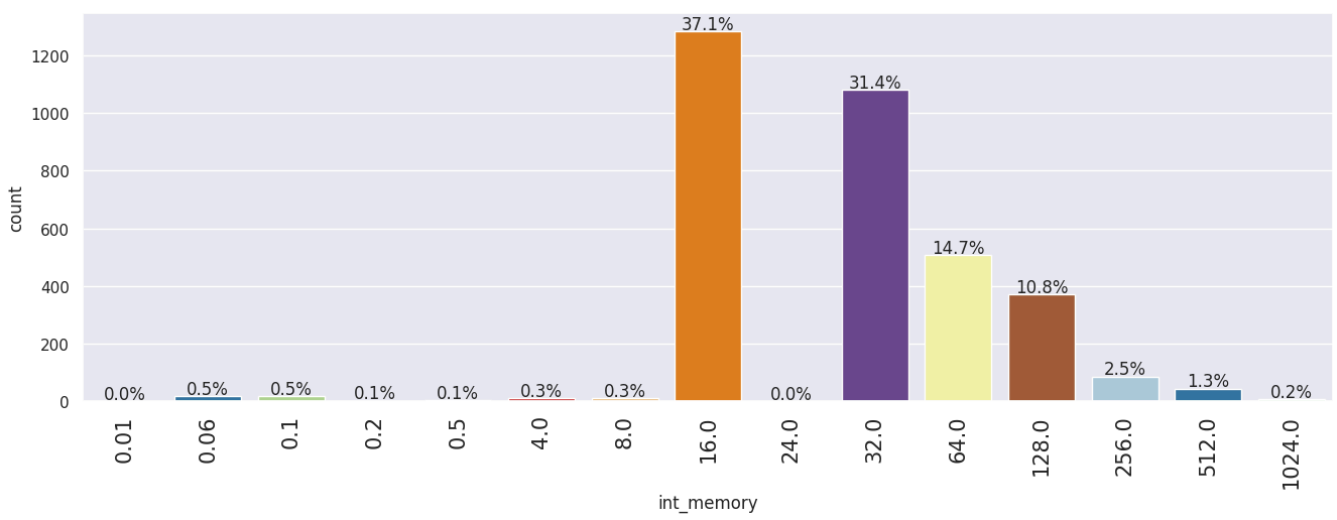
```
1 labeled_barplot(df, "ram", perc=True)
```



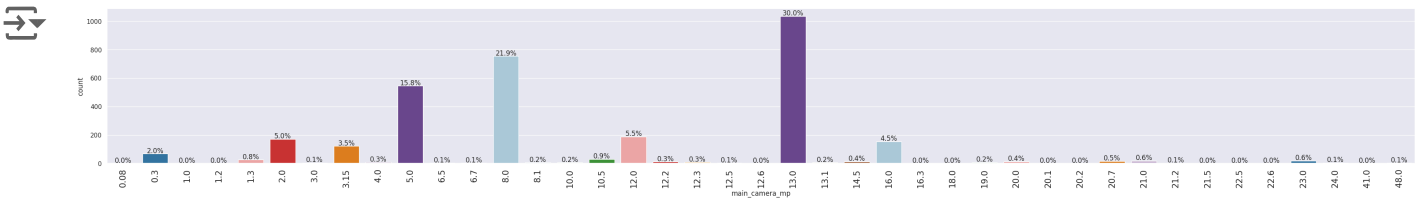

```
1 labeled_barplot(df, "release_year", perc=True)
```



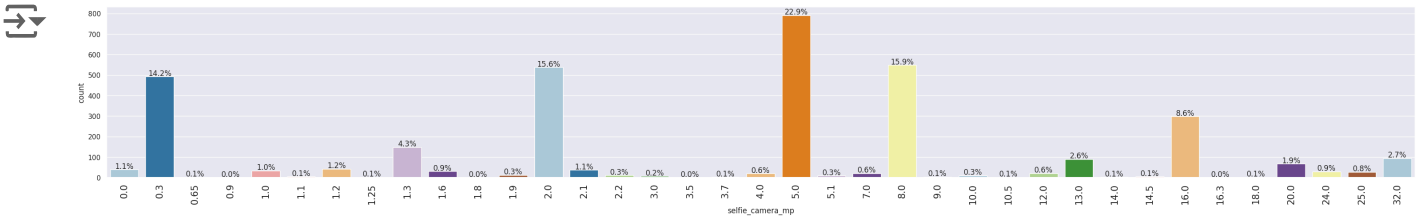

```
1 labeled_barplot(df, "int_memory", perc=True)
```



```
1 labeled_barplot(df, "main_camera_mp", perc=True)
```




```
1 labeled_barplot(df, "selfie_camera_mp", perc=True)
```

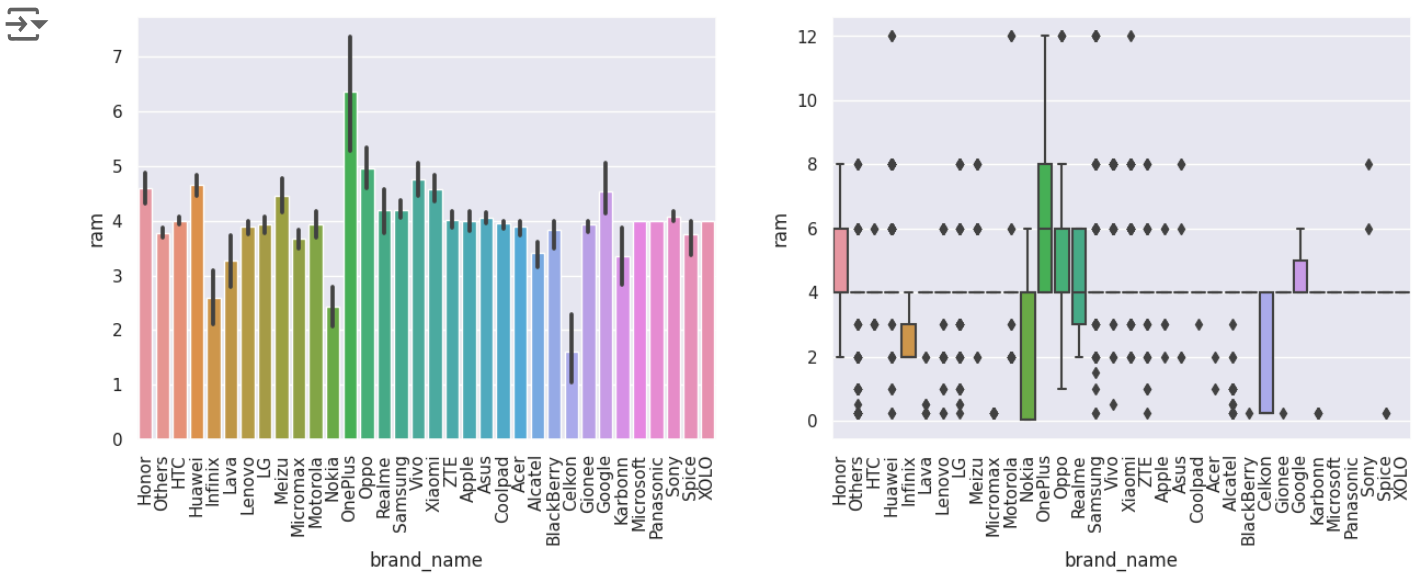



```
1 df.groupby("brand_name")["ram"].mean()
```

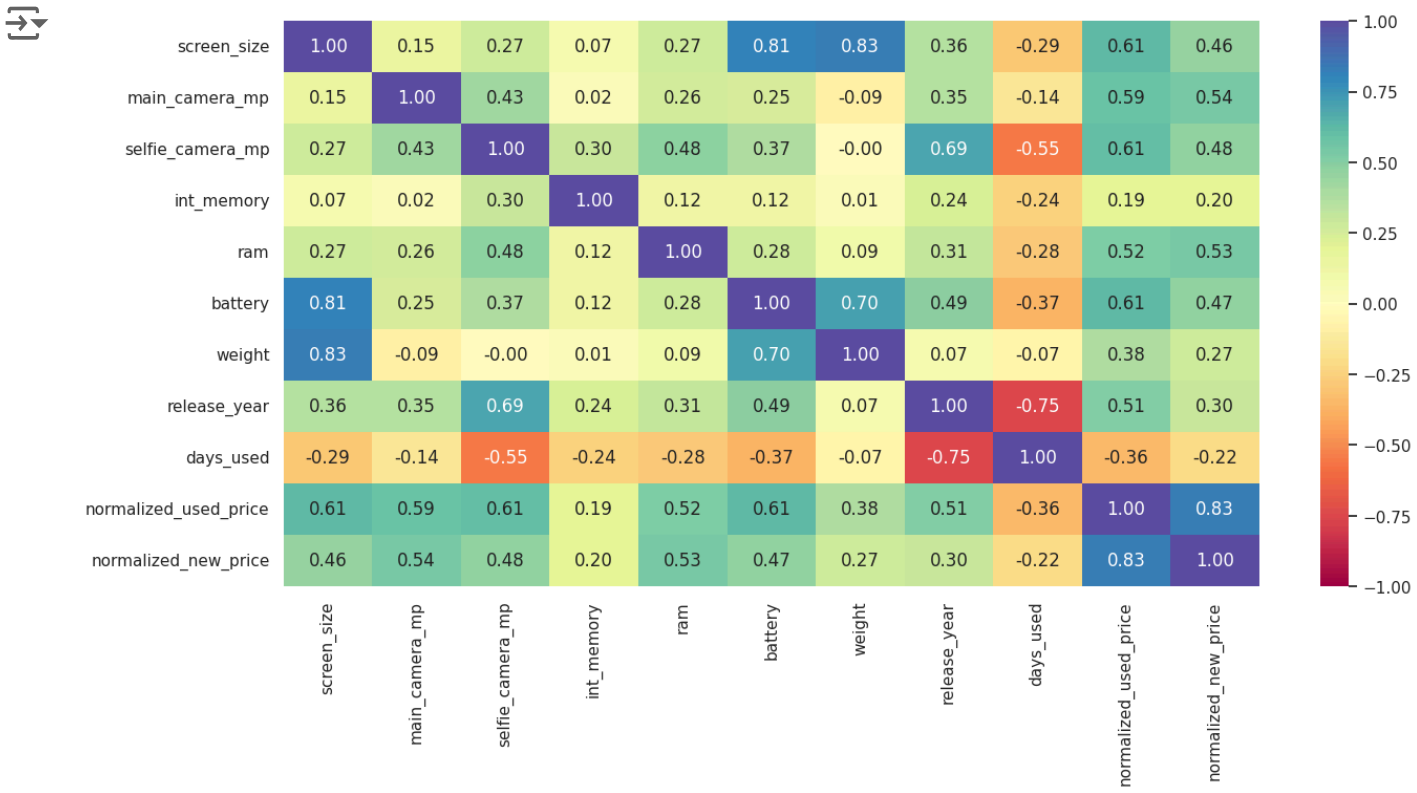
```
brand_name
Acer          3.901961
Alcatel       3.407025
Apple         4.000000
Asus          4.049180
BlackBerry    3.829545
Celkon        1.613636
Coolpad       3.954545
Gionee        3.933036
Google        4.533333
HTC           4.000000
Honor         4.603448
Huawei        4.655378
Infinix       2.600000
Karbonn       3.353448
LG            3.936567
Lava          3.277778
Lenovo        3.885965
Meizu         4.451613
Micromax      3.679487
Microsoft     4.000000
Motorola      3.943396
Nokia         2.420294
OnePlus       6.363636
Oppo          4.961240
Others        3.777888
Panasonic     4.000000
Realme        4.195122
Samsung       4.199413
```

```
Sony          4.069767
Spice         3.750000
Vivo          4.756410
XOLO          4.000000
Xiaomi        4.583333
ZTE           4.023214
Name: ram, dtype: float64
```

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
sns.barplot(data=df, y="ram", x="brand_name")
plt.xticks(rotation=90)

plt.subplot(1, 2, 2)
sns.boxplot(data=df, y="ram", x="brand_name")
plt.xticks(rotation=90)

plt.show()
```



```
numeric_columns = df.select_dtypes(include=np.number).columns.tolist()

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



Observations:

- used_price is highly correlated with new_price which means the higher the new price the higher the used price tends to increase.

- used_price also moderately correlated with ram and selfie_camera_mp meaning the higher the phone ram or selfie camera pixels the higher the price of the used phone.
- used_price is moderately correlated negatively with days_used which can mean that the longer the duration the phone was used the lower the price of such used phone.

Data Preprocessing

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

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

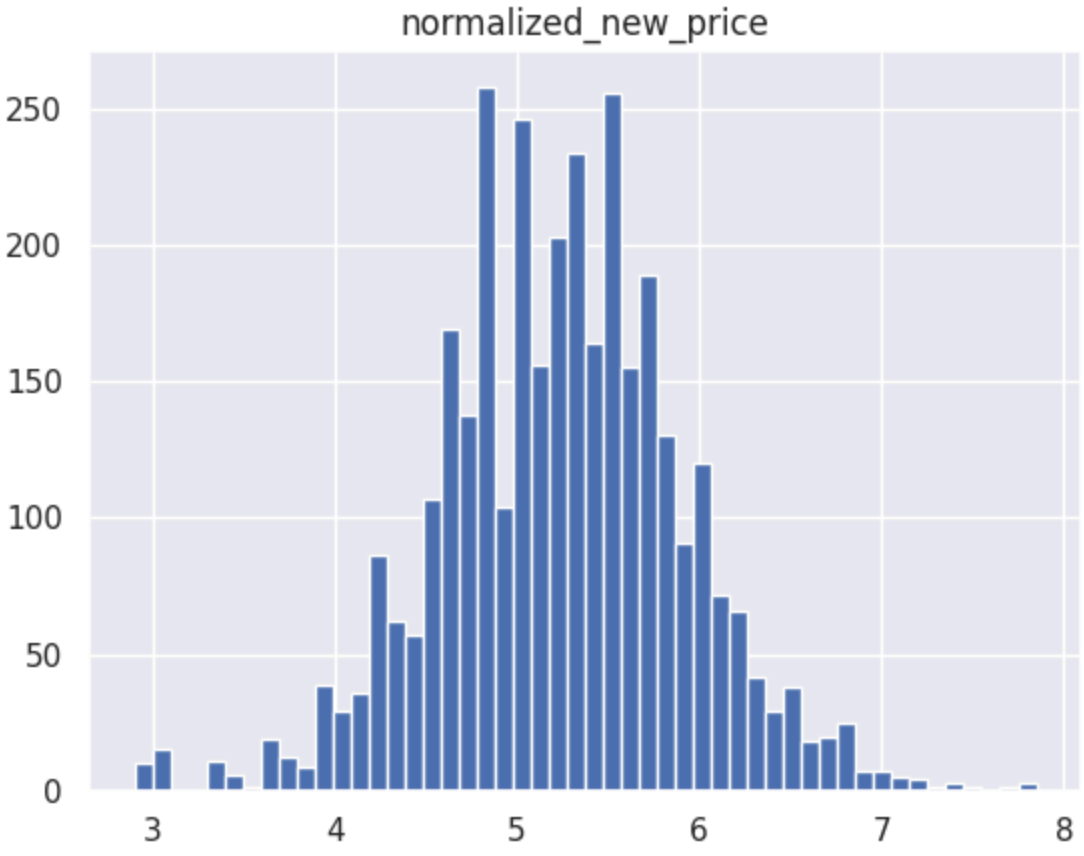
```
brand_name                0
os                        0
screen_size               0
4g                        0
5g                        0
main_camera_mp          179
selfie_camera_mp          2
int_memory                4
ram                       4
battery                   6
weight          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

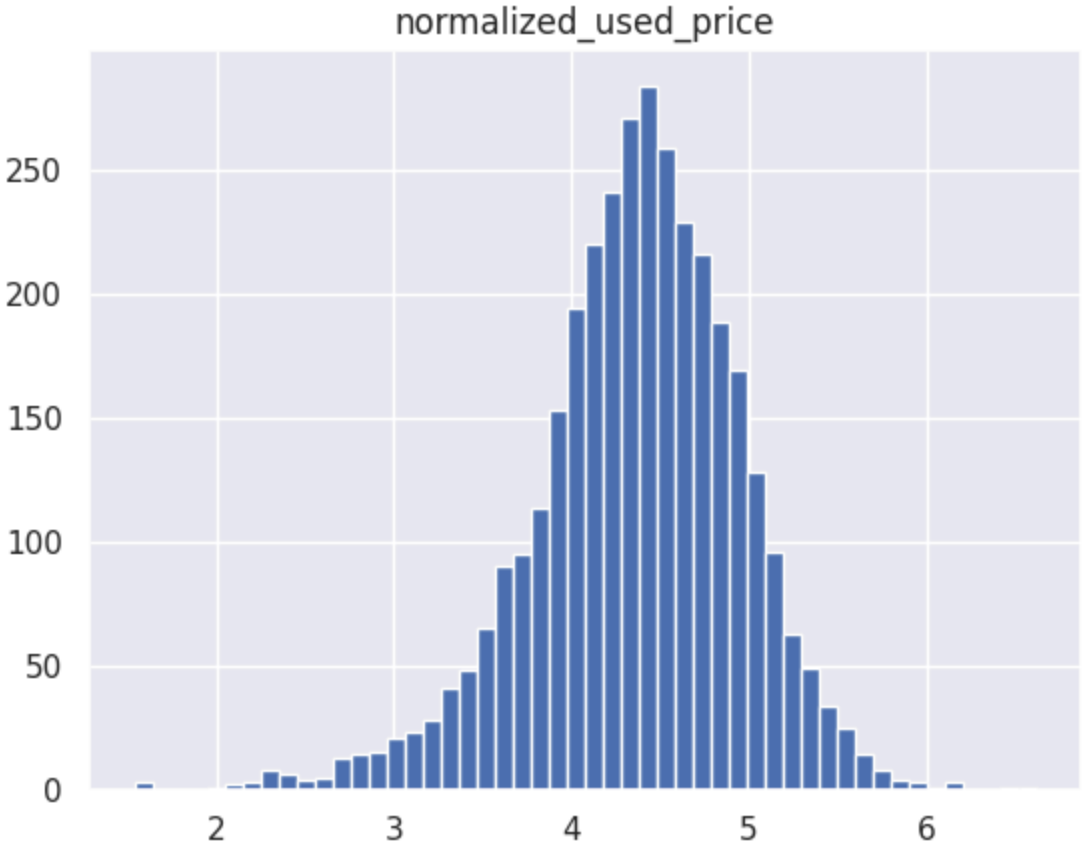
```
col_to_fill = [
    "main_camera_mp",
    "selfie_camera_mp",
    "int_memory",
    "ram",
    "battery",
    "weight",
]

df[col_to_fill] = df[col_to_fill].apply(lambda x: x.fillna(x.median()), axis=0)
```

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

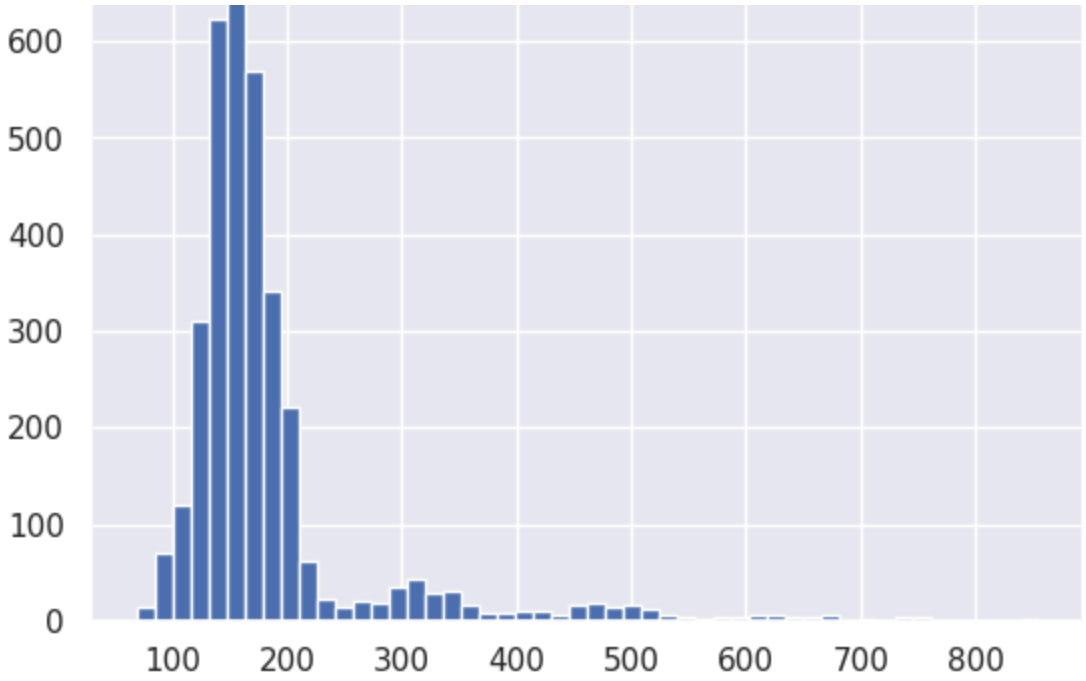
```
brand_name             0
os                     0
screen_size            0
4g                     0
5g                     0
main_camera_mp         0
selfie_camera_mp       0
int_memory             0
ram                    0
battery                0
weight                 0
release_year         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

```
cols_to_log = ["normalized_new_price", "normalized_used_price", "weight"]

for colname in cols_to_log:
    plt.hist(df[colname], bins=50)
    plt.title(colname)
    plt.show()
```


normalized_new_price
250
200
150
100
50
0
3
4
5
6
7
8

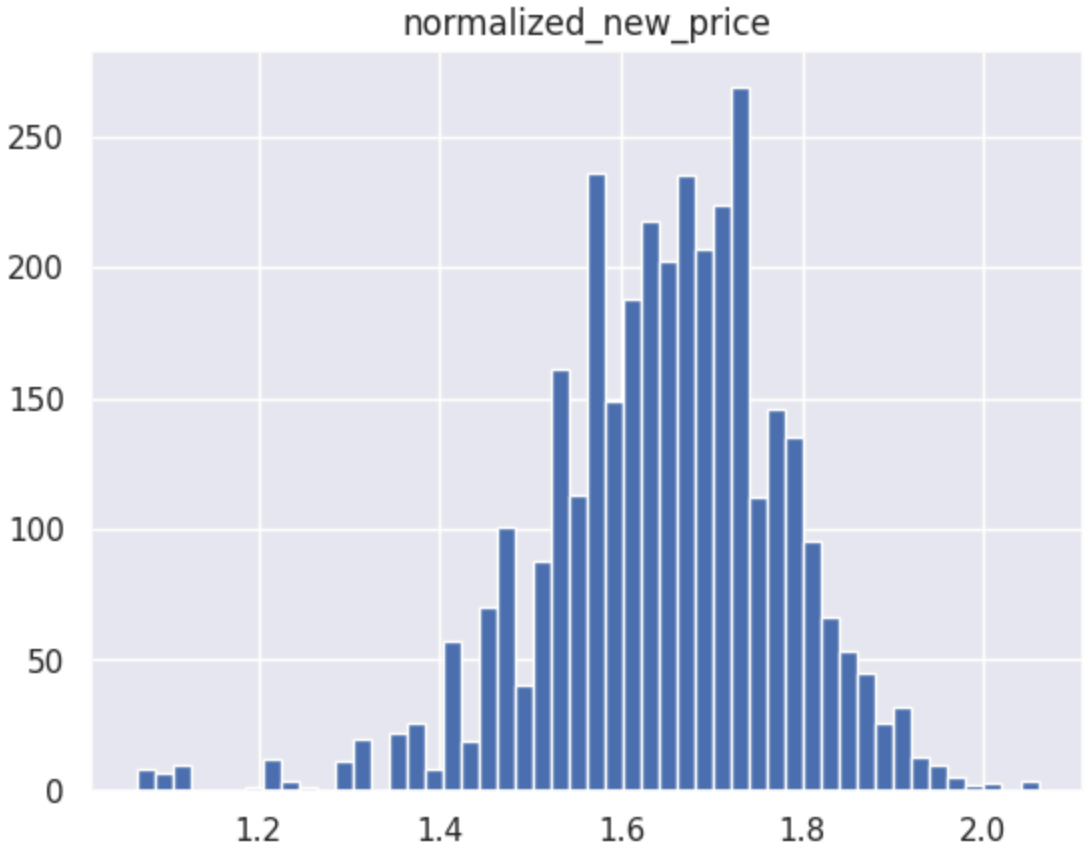

normalized_used_price
250
200
150
100
50
0
2
3
4
5
6

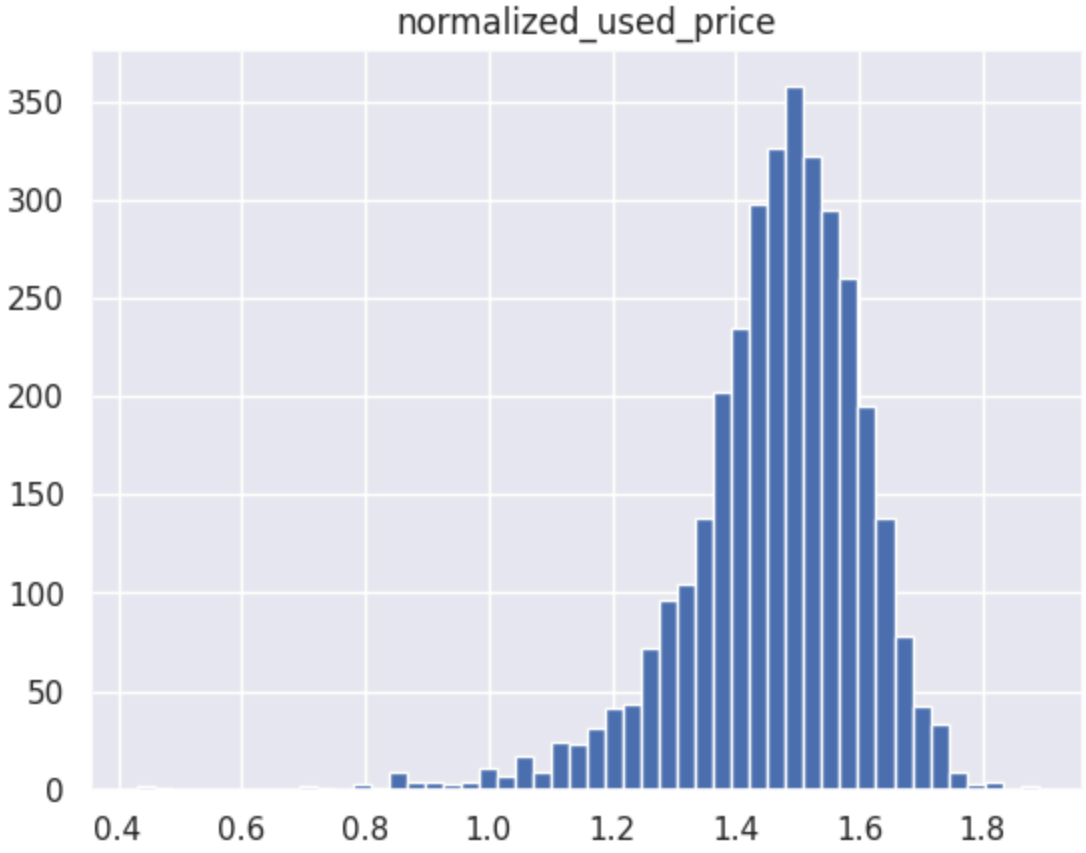

weight
700



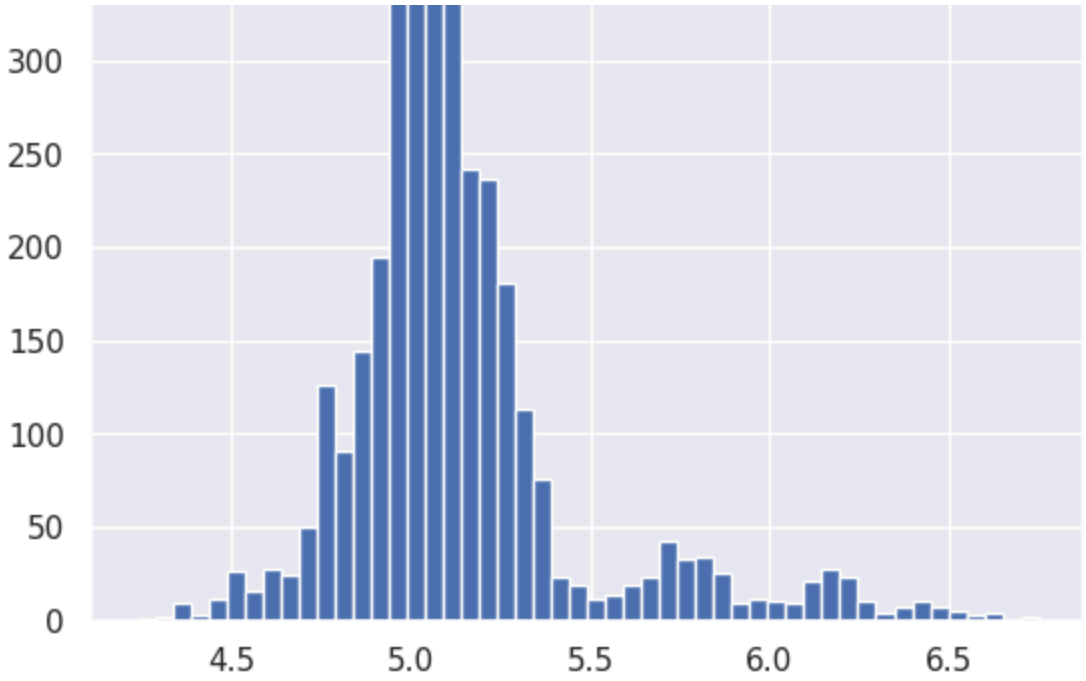

```
1 for colname in cols_to_log:
2     df[colname + "_log"] = np.log(df[colname])
3
4 df.drop(cols_to_log, axis=1, inplace=True)
```

```
1 for colname in cols_to_log:
2     plt.hist(df[colname + "_log"], bins=50)
3     plt.title(colname)
4     plt.show()
```


normalized_new_price
250
200
150
100
50
0
1.2
1.4
1.6
1.8
2.0


normalized_used_price
350
300
250
200
150
100
50
0
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8


weight
350



EDA

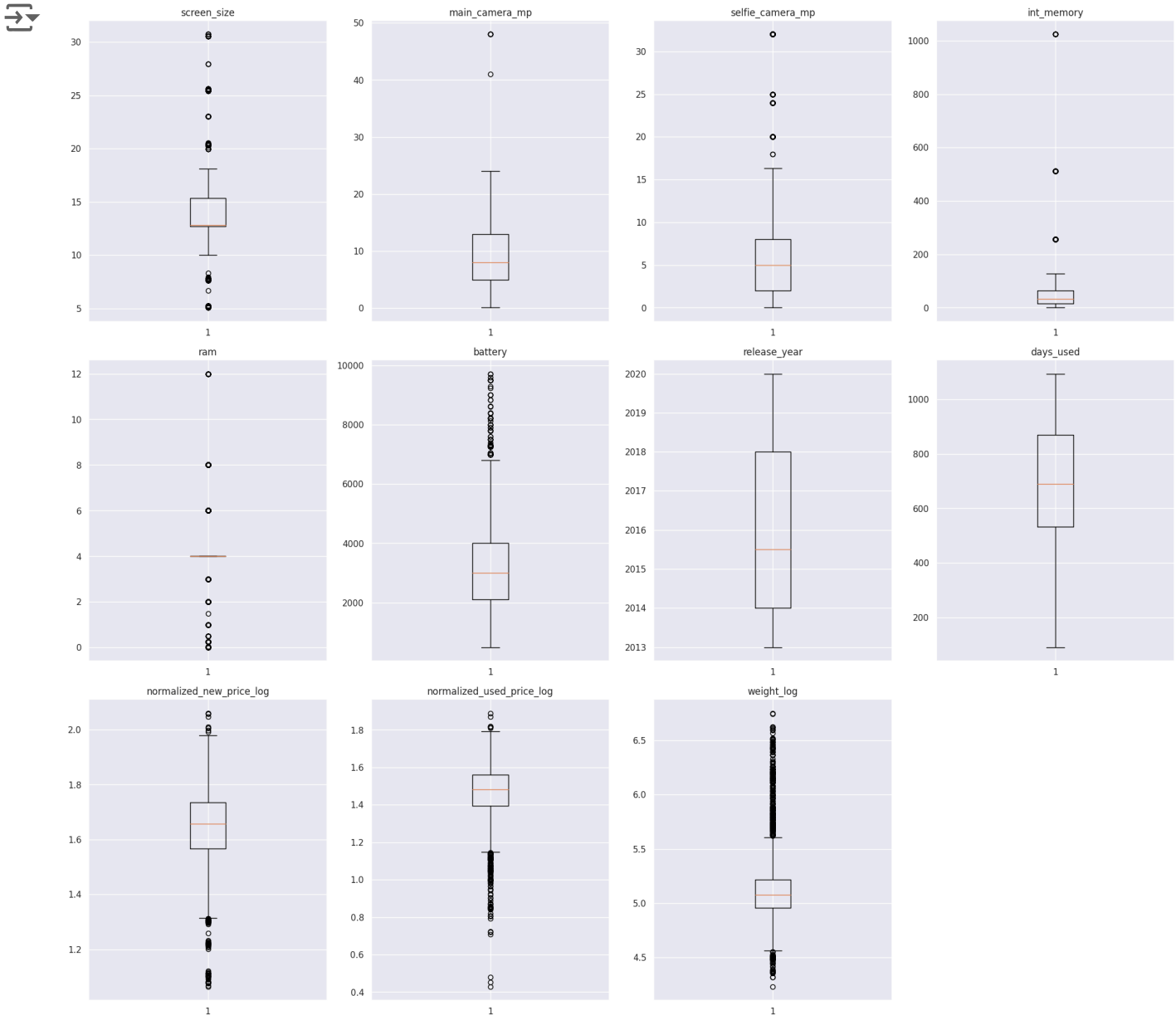
- It is a good idea to explore the data once again after manipulating it.

```
1 df1 = pd.get_dummies(df, columns=["brand_name", "os", "4g", "5g"], drop_first=True)
```

```
1 df1.head()
```

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	release_yea
0	14.50	13.00	5.00	64.00	3.00	3020.00	202
1	17.30	13.00	16.00	128.00	8.00	4300.00	202
2	16.69	13.00	8.00	128.00	8.00	4200.00	202
3	25.50	13.00	8.00	64.00	6.00	7250.00	202
4	15.32	13.00	8.00	64.00	3.00	5000.00	202

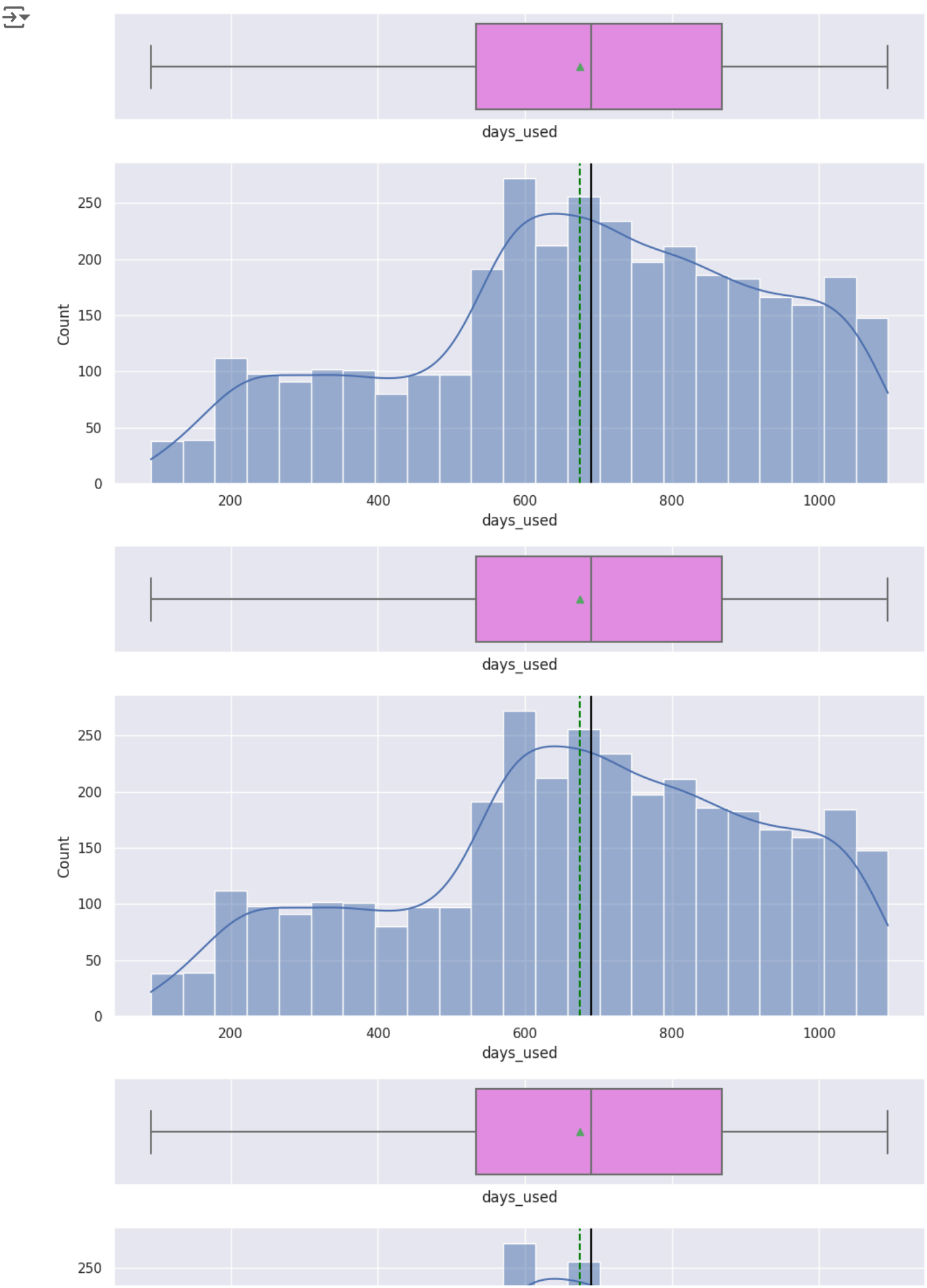
```
plt.figure(figsize=(20, 30))

for i, var in enumerate(df.select_dtypes(include=np.number).columns.tolist()):
    plt.subplot(5, 4, i + 1)
    plt.boxplot(df[var], whis=1.5)
    plt.tight_layout()
    plt.title(var)

plt.show()
```

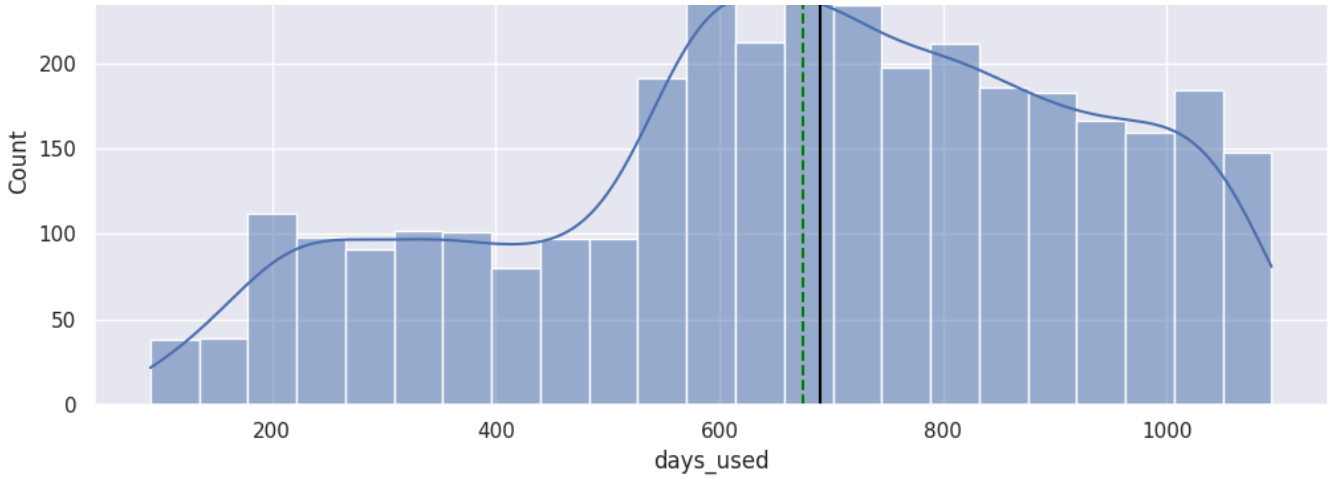


Observations:

- Outliers are present in all the numerical columns except for release_year and days_used.
- In this case we will not treat the outliers, since treating the outliers can lead to loss of information.

```
1 for col in ["used_price_log", "new_price_log", "weight_log"]:
2     histogram_boxplot(df, col)
```


days_used
Count
0
50
100
150
200
250
200
400
600
800
1000
days_used
days_used
Count
0
50
100
150
200
250
200
400
600
800
1000
days_used
days_used
250



Model Building - Linear Regression

```
1 X = df1.drop(["normalized_used_price_log"], axis=1)
2 y = df1["normalized_used_price_log"]
```

```
1 X.head()
```

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	release_yea
0	14.50	13.00	5.00	64.00	3.00	3020.00	202
1	17.30	13.00	16.00	128.00	8.00	4300.00	202
2	16.69	13.00	8.00	128.00	8.00	4200.00	202
3	25.50	13.00	8.00	64.00	6.00	7250.00	202
4	15.32	13.00	8.00	64.00	3.00	5000.00	202

```
1 y.head()
```

```
0   1.46
1   1.64
2   1.63
3   1.64
4   1.48
Name: normalized_used_price_log, dtype: float64
```

```
1 X_train, X_test, y_train, y_test = train_test_split(
2     X, y, test_size=0.30, random_state=42)
```

```
1 print("Number of rows in train data =", X_train.shape[0])
2 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
```

```
1 linearregression = LinearRegression()
2 linearregression.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
1 coef_df = pd.DataFrame(
2     np.append(linearregression.coef_, linearregression.intercept_),
3     index=X_train.columns.tolist() + ["Intercept"],
4     columns=["Coefficients"],)
5
6 coef_df
```

	Coefficients
screen_size	0.01
main_camera_mp	0.01
selfie_camera_mp	0.00
int_memory	-0.00
ram	0.01
battery	-0.00
release_year	0.01
days_used	0.00
normalized_new_price_log	0.55
weight_log	0.06
brand_name_Alcatel	-0.02
brand_name_Apple	-0.01
brand_name_Asus	0.00
brand_name_BlackBerry	0.02
brand_name_Celkon	-0.07
brand_name_Coolpad	0.00
brand_name_Gionee	-0.00
brand_name_Google	-0.03
brand_name_HTC	-0.01
brand_name_Honor	-0.01
brand_name_Huawei	-0.01
brand_name_Infinix	0.04
brand_name_Karbonn	-0.01
brand_name_LG	-0.01
brand_name_Lava	0.00
brand_name_Lenovo	-0.00
brand_name_Meizu	-0.01
brand_name_Micromax	-0.01
brand_name_Microsoft	0.02
brand_name_Motorola	-0.01

brand_name_Nokia	0.02
brand_name_OnePlus	0.00
brand_name_Oppo	-0.00
brand_name_Others	-0.02
brand_name_Panasonic	-0.01
brand_name_Realme	0.02
brand_name_Samsung	-0.01
brand_name_Sony	-0.02
brand_name_Spice	-0.00
brand_name_Vivo	-0.01
brand_name_XOLO	-0.01
brand_name_Xiaomi	0.01
brand_name_ZTE	-0.01
os_Others	-0.03
os_Windows	-0.01
os_iOS	-0.01
4g_yes	0.01
5g_yes	-0.02
Intercept	-15.98

Model Performance Check

```
def adj_r2_score(predictors, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = predictors.shape[0]
    k = predictors.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))


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


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

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

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

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

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

    return df_perf
```

```
print("Training Performance\n")
linearregression_train_perf = model_performance_regression(
    linearregression, X_train, y_train
)
linearregression_train_perf
```

Training Performance

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.06	0.04	0.84	0.84	3.14

```
print("Test Performance\n")
linearregression_test_perf = model_performance_regression(
    linearregression, X_test, y_test
)
linearregression_test_perf
```

Test Performance

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.06	0.05	0.83	0.82	3.27

Observations:

The training $R2$ is 84%, indicating that the model explains 83% of the variation in the train data. we can tell the model is not underfitted.

MAE and RMSE on the train and test sets are comparable, which shows that the model is not overfitting.

MAE indicates that our current model is able to predict used price within a mean error of 0.05 euros on the test data.

MAPE on the test set suggests we can predict within 3.27% of the used price.

```
X_train1 = sm.add_constant(X_train)
# adding constant to the test data
X_test1 = sm.add_constant(X_test)

olsmod0 = sm.OLS(y_train, X_train1).fit()
print(olsmod0.summary())
```

```
                                OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price_log   R-squared:                       0.840
Model:                                   OLS   Adj. R-squared:                  0.836
Method:                        Least Squares   F-statistic:                     258.5
Date:                       Thu, 26 Oct 2023   Prob (F-statistic):              0.00
Time:                               22:36:15   Log-Likelihood:                 3426.2
No. Observations:                       2417   AIC:                            -6754.
Df Residuals:                           2368   BIC:                            -6471.
Df Model:                                 48
```

```
Covariance Type:                  nonrobust
==============================================================================
                                 coef    std err          t      P>|t|      [0.025
------------------------------------------------------------------------------
const                        -15.9832      2.338     -6.838      0.000     -20.567
screen_size                    0.0064      0.001      7.377      0.000       0.005
main_camera_mp                 0.0050      0.000     12.983      0.000       0.004
selfie_camera_mp               0.0022      0.000      7.211      0.000       0.002
int_memory                 -8.741e-06   1.71e-05     -0.512      0.609   -4.22e-05    2
ram                            0.0074      0.001      5.554      0.000       0.005
battery                    -5.773e-06   1.88e-06     -3.076      0.002   -9.45e-06   -2
release_year                   0.0080      0.001      6.911      0.000       0.006
days_used                   1.202e-05   7.83e-06      1.536      0.125   -3.32e-06    2
normalized_new_price_log       0.5495      0.016     34.232      0.000       0.518
weight_log                     0.0568      0.009      6.413      0.000       0.039
brand_name_Alcatel            -0.0163      0.013     -1.280      0.201      -0.041
brand_name_Apple              -0.0129      0.038     -0.343      0.732      -0.087
brand_name_Asus                0.0027      0.013      0.212      0.832      -0.022
brand_name_BlackBerry          0.0180      0.018      0.978      0.328      -0.018
brand_name_Celkon             -0.0666      0.017     -3.830      0.000      -0.101
brand_name_Coolpad             0.0011      0.018      0.061      0.951      -0.034
brand_name_Gionee             -0.0028      0.015     -0.184      0.854      -0.032
brand_name_Google             -0.0306      0.021     -1.451      0.147      -0.072
brand_name_HTC                -0.0084      0.013     -0.659      0.510      -0.033
brand_name_Honor              -0.0096      0.013     -0.741      0.459      -0.035
brand_name_Huawei             -0.0149      0.012     -1.267      0.205      -0.038
brand_name_Infinix             0.0426      0.029      1.471      0.141      -0.014
brand_name_Karbonn            -0.0054      0.017     -0.311      0.756      -0.040
brand_name_LG                 -0.0132      0.012     -1.102      0.270      -0.037
brand_name_Lava                0.0005      0.016      0.028      0.977      -0.031
brand_name_Lenovo             -0.0049      0.012     -0.404      0.686      -0.028
brand_name_Meizu              -0.0102      0.014     -0.711      0.477      -0.038
brand_name_Micromax           -0.0145      0.013     -1.155      0.248      -0.039
brand_name_Microsoft           0.0226      0.021      1.083      0.279      -0.018
brand_name_Motorola           -0.0094      0.013     -0.727      0.467      -0.035
brand_name_Nokia               0.0151      0.013      1.126      0.260      -0.011
brand_name_OnePlus             0.0035      0.019      0.187      0.851      -0.033
brand_name_Oppo               -0.0028      0.013     -0.222      0.825      -0.027
brand_name_Others             -0.0162      0.011     -1.450      0.147      -0.038
brand_name_Panasonic          -0.0051      0.016     -0.321      0.748      -0.036
brand_name_Realme              0.0153      0.016      0.950      0.342      -0.016
brand_name_Samsung            -0.0147      0.011     -1.284      0.199      -0.037
brand_name_Sony               -0.0172      0.013     -1.274      0.203      -0.044
brand_name_Spice              -0.0045      0.017     -0.257      0.797      -0.039
brand_name_Vivo               -0.0110      0.013     -0.851      0.395      -0.036
brand_name_XOLO               -0.0148      0.015     -1.012      0.312      -0.044
brand_name_Xiaomi              0.0128      0.013      1.015      0.310      -0.012
```

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.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# we will define a function to check VIF
def checking_vif(predictors):
    vif = pd.DataFrame()
    vif["feature"] = predictors.columns

    # calculating VIF for each feature
    vif["VIF"] = [
        round(variance_inflation_factor(predictors.values, i), 2)
        for i in range(len(predictors.columns))
    ]
    return vif
```

```
checking_vif(X_train1)
```

	feature	VIF
0	const	3763803.01
1	screen_size	7.69
2	main_camera_mp	2.25
3	selfie_camera_mp	2.86
4	int_memory	1.33
5	ram	2.24
6	battery	4.07
7	release_year	4.79
8	days_used	2.58
9	normalized_new_price_log	3.33
10	weight_log	6.36
11	brand_name_Alcatel	3.47
12	brand_name_Apple	11.24
13	brand_name_Asus	3.65
14	brand_name_BlackBerry	1.62
15	brand_name_Celkon	1.88
16	brand_name_Coolpad	1.58
17	brand_name_Gionee	2.08
18	brand_name_Google	1.39
19	brand_name_HTC	3.47
20	brand_name_Honor	3.57
21	brand_name_Huawei	6.41
22	brand_name_Infinix	1.19
23	brand_name_Karbonn	1.64
24	brand_name_LG	5.37
25	brand_name_Lava	1.83
26	brand_name_Lenovo	4.71
27	brand_name_Meizu	2.41
28	brand_name_Micromax	3.80
29	brand_name_Microsoft	2.09

30	brand_name_Motorola	3.48
31	brand_name_Nokia	3.75
32	brand_name_OnePlus	1.59
33	brand_name_Oppo	4.30
34	brand_name_Others	10.84
35	brand_name_Panasonic	1.90
36	brand_name_Realme	1.96
37	brand_name_Samsung	8.03
38	brand_name_Sony	2.88
39	brand_name_Spice	1.64
40	brand_name_Vivo	3.74
41	brand_name_XOLO	2.17
42	brand_name_Xiaomi	4.08
43	brand_name_ZTE	4.35
44	os_Others	1.81
45	os_Windows	1.74
46	os_iOS	10.06
47	4g_yes	2.55
48	5g_yes	1.83

```
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
```

```
col_list = [
    "brand_name_Apple",
    "os_iOS",
    "brand_name_Others",
    "brand_name_Samsung",
    "brand_name_Huawei",
    "brand_name_LG",
]

res = treating_multicollinearity(X_train1, y_train, col_list)
res
```