Supervised Learning - Foundations Project: ReCell

Problem Statement

Business Context

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

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

Objective

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

Data Description

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

- brand_name: Name of manufacturing brand
- os: OS on which the device runs
- screen_size: Size of the screen in cm
- 4g: Whether 4G is available or not
- 5g: Whether 5G is available or not
- main_camera_mp: Resolution of the rear camera in megapixels
- selfie_camera_mp: Resolution of the front camera in megapixels
- int_memory: Amount of internal memory (ROM) in GB
- ram: Amount of RAM in GB

- battery: Energy capacity of the device battery in mAh
- weight: Weight of the device in grams
- release_year: Year when the device model was released
- days_used: Number of days the used/refurbished device has been used
- normalized_new_price: Normalized price of a new device of the same model in euros
- normalized_used_price: Normalized price of the used/refurbished device in euros

Importing necessary libraries

In [ ]:

```
# Installing the libraries with the specified version.
# uncomment and run the following line if Google Colab is being used
# !pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 panda
```

In [ ]:

```
# Installing the libraries with the specified version.
# uncomment and run the following lines if Jupyter Notebook is being used
# !pip install scikit-learn==1.2.2 seaborn==0.11.1 matplotlib==3.3.4 numpy==1.24.3 panda
```

In [ ]:

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

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

sns.set()

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

# to build linear regression_model
from sklearn.linear_model import LinearRegression

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

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

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

Note: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.*

Loading the dataset

In [ ]:

```
# loading the dataset Google Colab
from google.colab import drive
```

```
drive.mount('/content/drive')
```

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

In [ ]:

```
# loading data
data = pd.read_csv('/content/drive/MyDrive/Google_colab/used_device_data.csv')
```

Data Overview

- Observations
- Sanity checks

In [ ]:

```
#observing the data
data.head()
```

Out[ ]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	ram
0	Honor	Android	14.50	yes	no	13.0	5.0	64.0	3.0
1	Honor	Android	17.30	yes	yes	13.0	16.0	128.0	8.0
2	Honor	Android	16.69	yes	yes	13.0	8.0	128.0	8.0
3	Honor	Android	25.50	yes	yes	13.0	8.0	64.0	6.0
4	Honor	Android	15.32	yes	no	13.0	8.0	64.0	3.0

In [ ]:

```
#checking the shape of the data
data.shape
```

Out[ ]:

```
(3454, 15)
```

There are 3454 rows and 15 columns in the dataset

In [ ]:

```
#To get information about the datatypes
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
```

This shows the datatypes of the columns.It looks relevant

In [ ]:

```
#To get the statistical information about the dataset
data.describe(include='all')
```

Out[ ]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memc
count	3454	3454	3454.000000	3454	3454	3275.000000	3452.000000	3450.0000
unique	34	4	NaN	2	2	NaN	NaN	N
top	Others	Android	NaN	yes	no	NaN	NaN	N
freq	502	3214	NaN	2335	3302	NaN	NaN	N
mean	NaN	NaN	13.713115	NaN	NaN	9.460208	6.554229	54.5730
std	NaN	NaN	3.805280	NaN	NaN	4.815461	6.970372	84.9723
min	NaN	NaN	5.080000	NaN	NaN	0.080000	0.000000	0.0100
25%	NaN	NaN	12.700000	NaN	NaN	5.000000	2.000000	16.0000
50%	NaN	NaN	12.830000	NaN	NaN	8.000000	5.000000	32.0000
75%	NaN	NaN	15.340000	NaN	NaN	13.000000	8.000000	64.0000
max	NaN	NaN	30.710000	NaN	NaN	48.000000	32.000000	1024.0000

Unique_values: There are 34 unique phone brand names. There are 4 unique os types. 4g and 5g has only 2 values yes ,No

Frequency: There are 3214 records that have Android as OS type (Most used OS type) 2335 peoplw have 4g and 3302 have 5g .so it seems like some people have both 4g and 5g as the total records is 3454.

Average: The average screen size used is approximately around 13.71 (Minimum used is approximately around 5.08 and maximum is approximately around 30.71) On an average,the phones are sold after 674 days of usage . On a minimum scale, after approximately 91 days of usage,the phone are brought into this reselling market. On a maximum scale, after approximately 1094 days of usage,the phone are brought into this reselling market. On an average,Normalised new price is around 5.23 and Normalised used price is around 4.36

Missing_values: It seems like (main_camera_mp,int_memory,ram,battery,weight) these columns have some missing or null values.

In [ ]:

```
#check for duplicate values:
data.duplicated().sum()
```

Out[ ]:

```
np.int64(0)
```

There are no duplicate rows in the dataset

In [ ]:

```
#check for missing values:
data.isnull().sum()
```

Out[ ]:

	0
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

It seems like (main_camera_mp,selfie_camers_mp,int_memory,ram,battery,weight) these columns have some missing or null values.

In [ ]:

```
# creating a copy of the data so that original data remains unchanged
df = data.copy()
```

Exploratory Data Analysis (EDA)

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

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

Questions:

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

In [ ]:

```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

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

```
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [ ]:

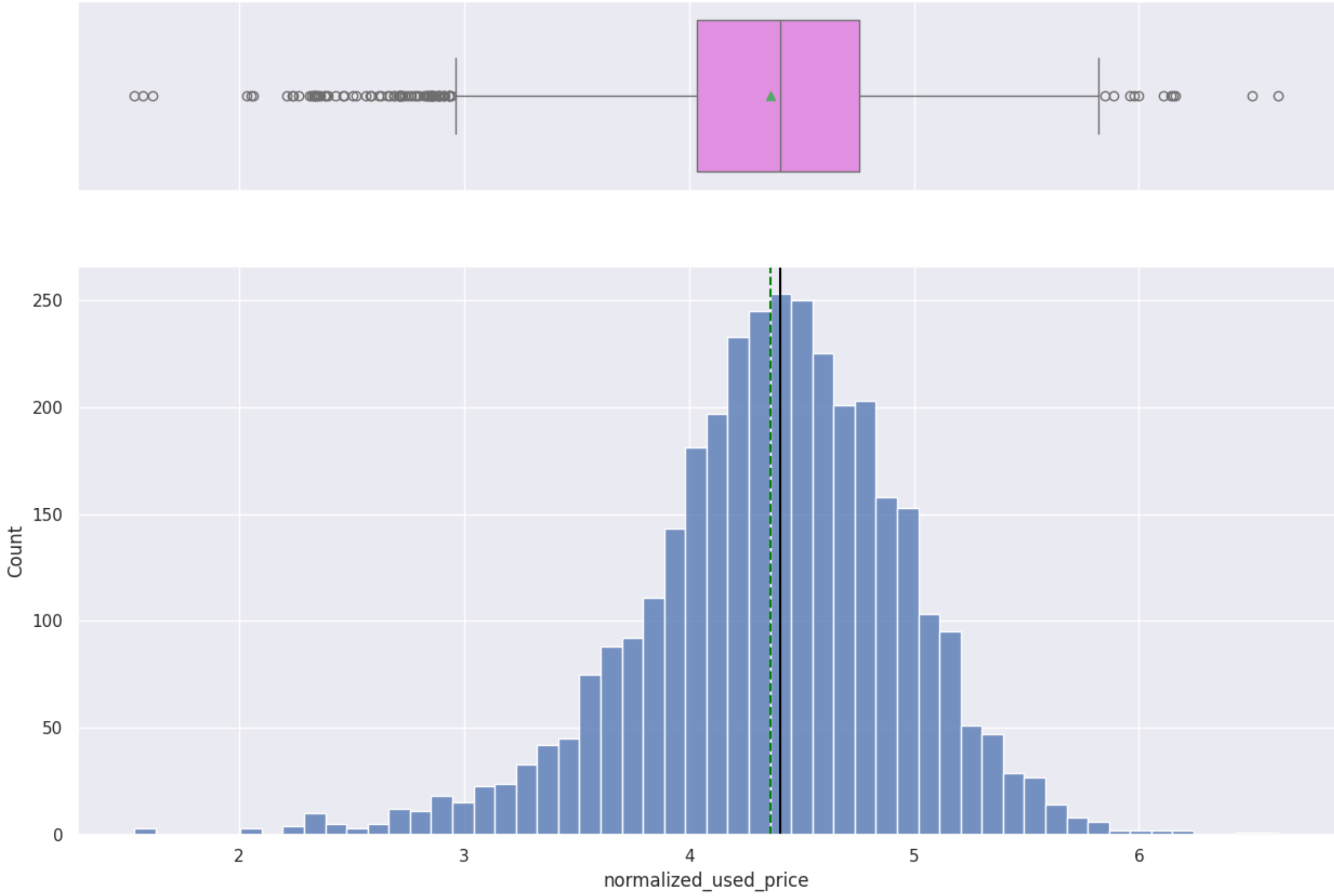
```
# function to create labeled barplots


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

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

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

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

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

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

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

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

What does the distribution of normalized used device prices look like?

Lets explore the normalized_used_price column

In [ ]:

```
#histogram for normalized_used_price
histogram_boxplot(df, "normalized_used_price")
```

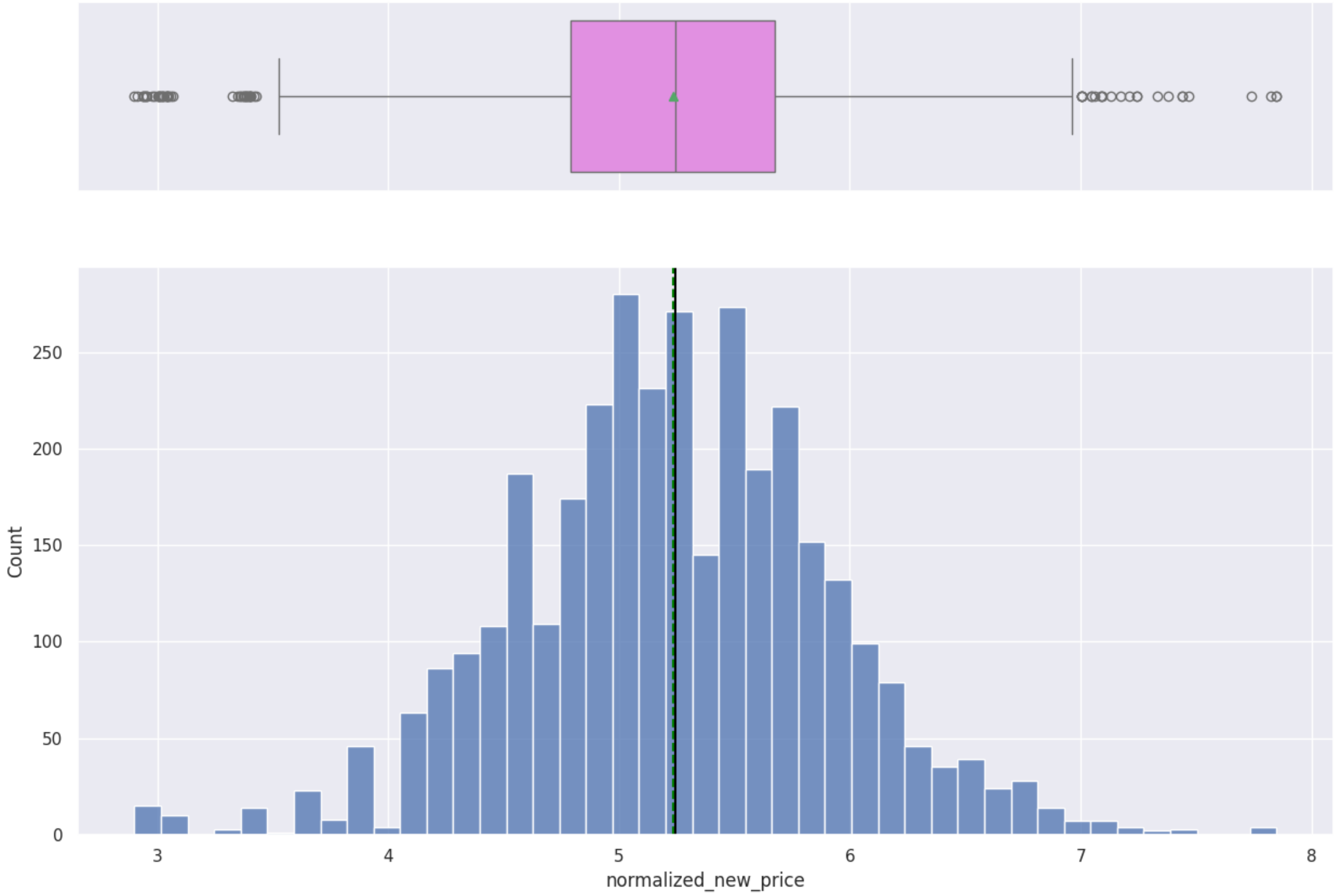


The average of normalized_used_price is approximately around 4.3.There are some outliers in this column .The data looks normally distributed .More than 250 phones have normalised used price in the range around 4 to 5 approximately.

Lets also explore all columns seperately

In [ ]:

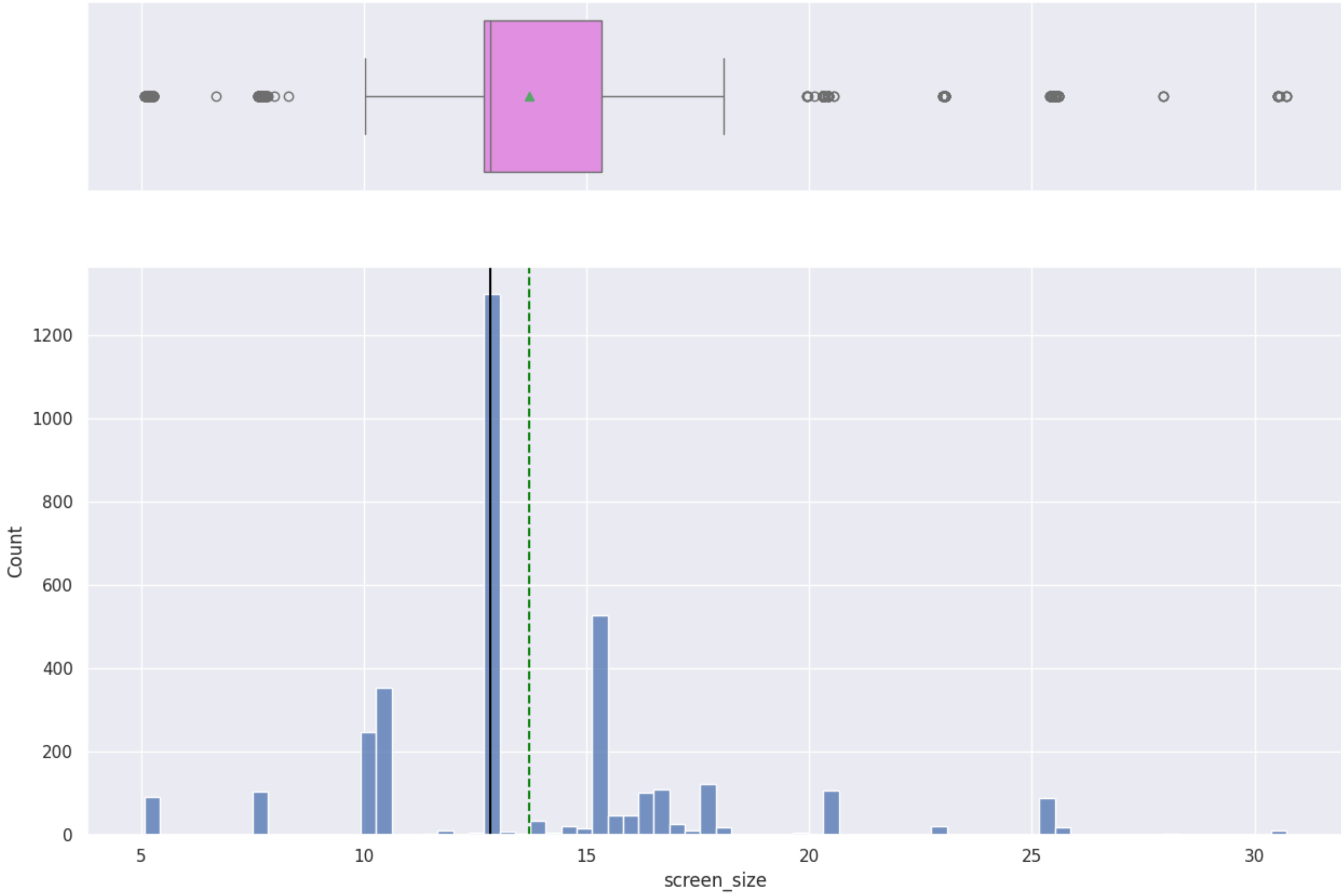
```
#histogram for normalized_new_price
histogram_boxplot(df, "normalized_new_price")
```



There are some outliers in this column.the average is around 5.3.Most phones have normalised new price around 4 to 6

In [ ]:

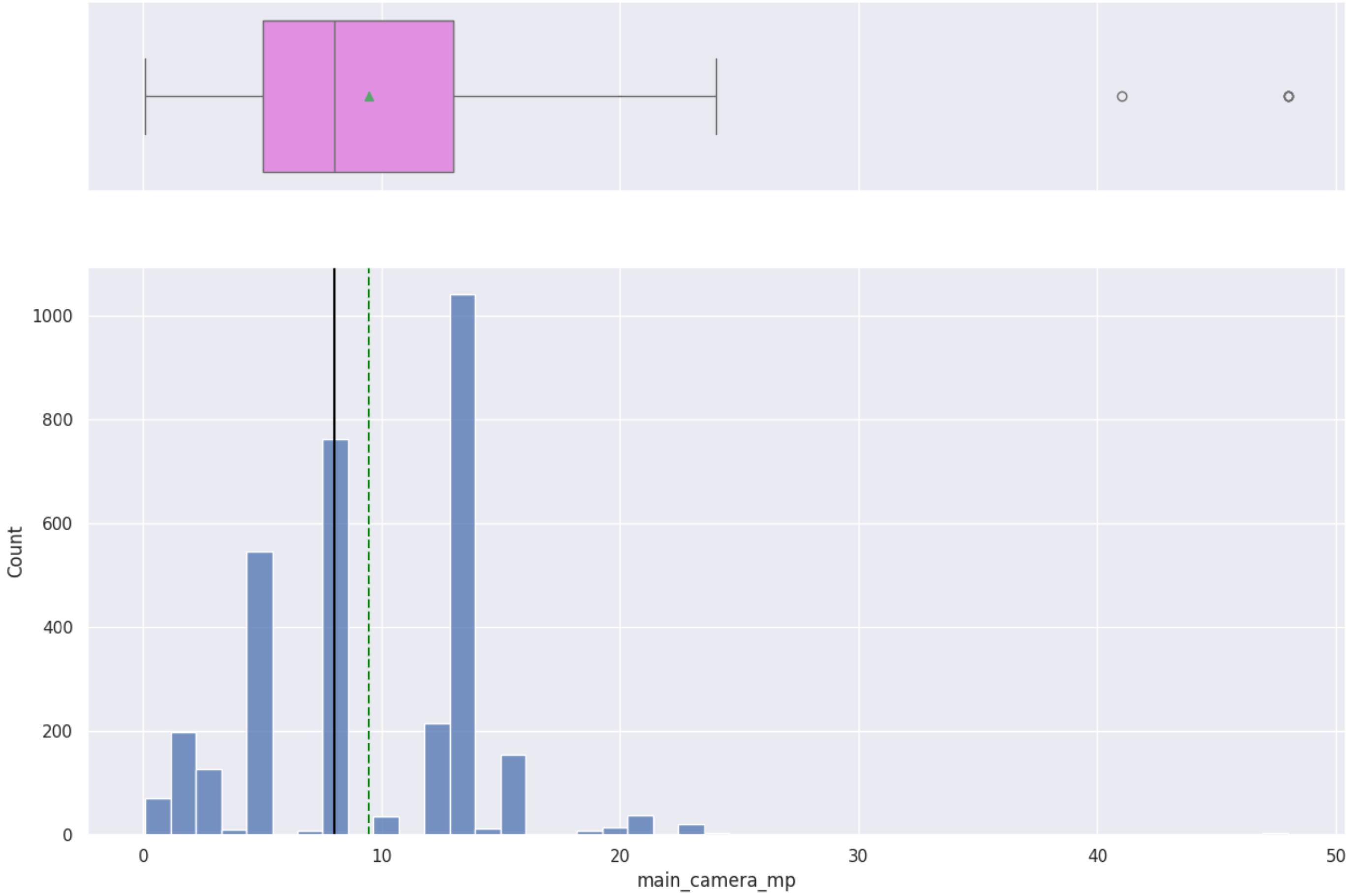
```
#histogram for screen_size
histogram_boxplot(df, "screen_size")
```



The average is around 13 .More than 1200 phones have screen_size around the range of 11 to 13 appromiately

In [ ]:

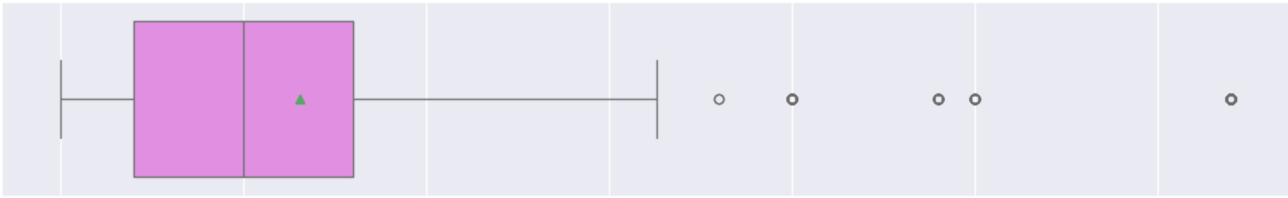
```
#histogram for main_camera_mp
histogram_boxplot(df, "main_camera_mp")
```

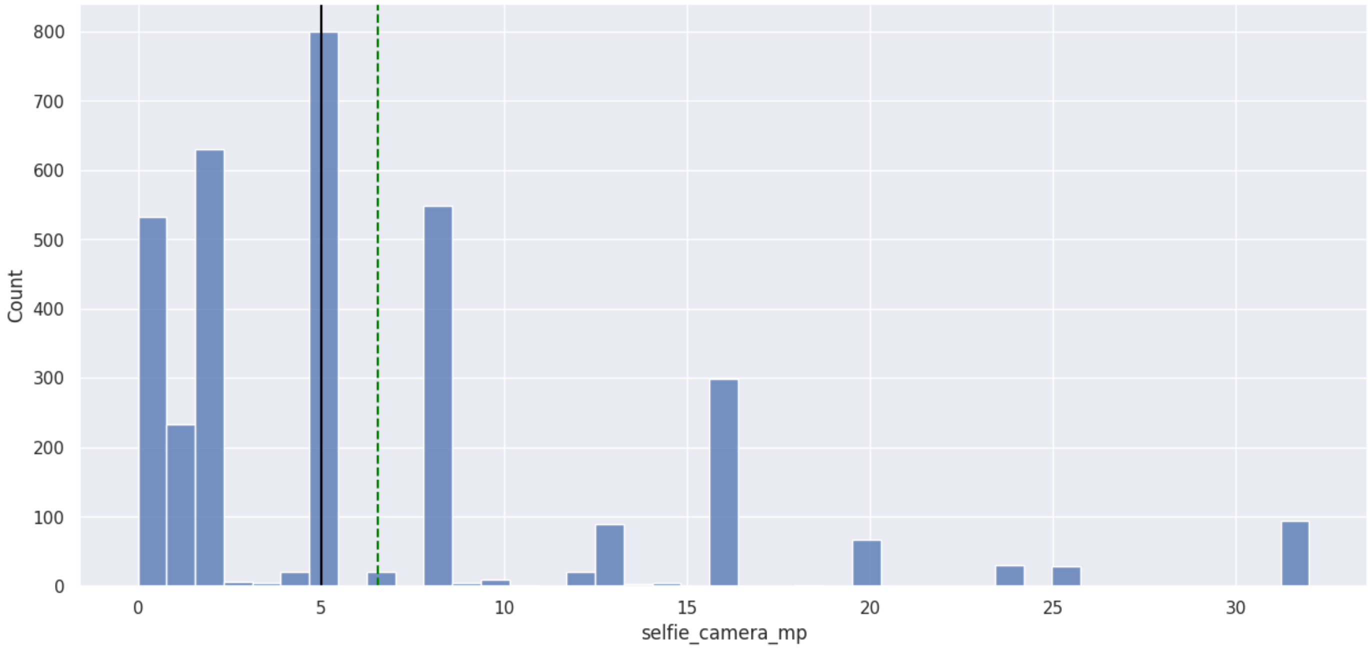


More than 1000 phones have main_camera_mp as around 14.average is around 9.

In [ ]:

```
#histogram for selfie_camera_mp
histogram_boxplot(df, "selfie_camera_mp")
```

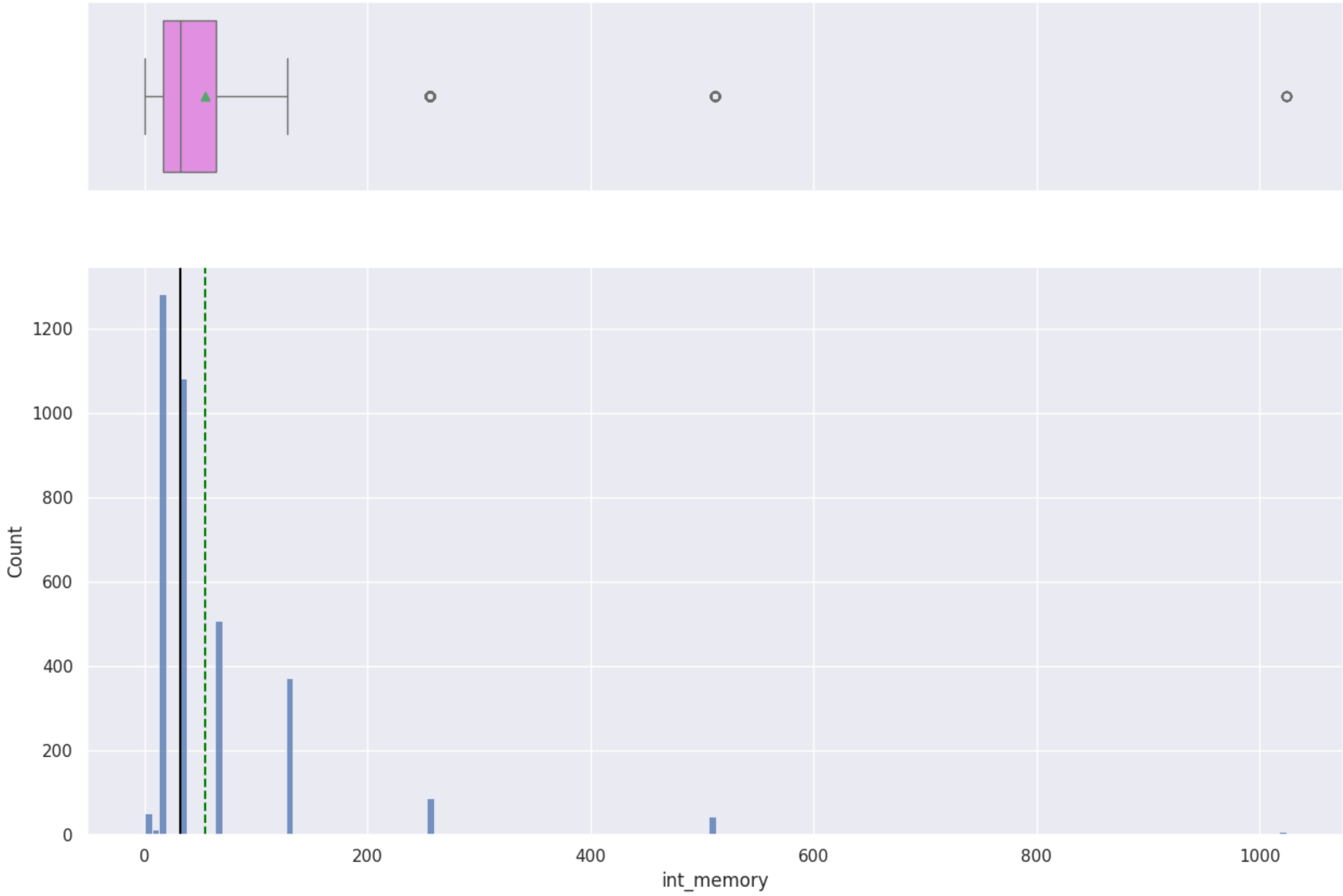




The average is around 6.5 approximately .Around 800 phones (highest count) have 5 as their selfie camers mp.There are not much outliers in the data

In [ ]:

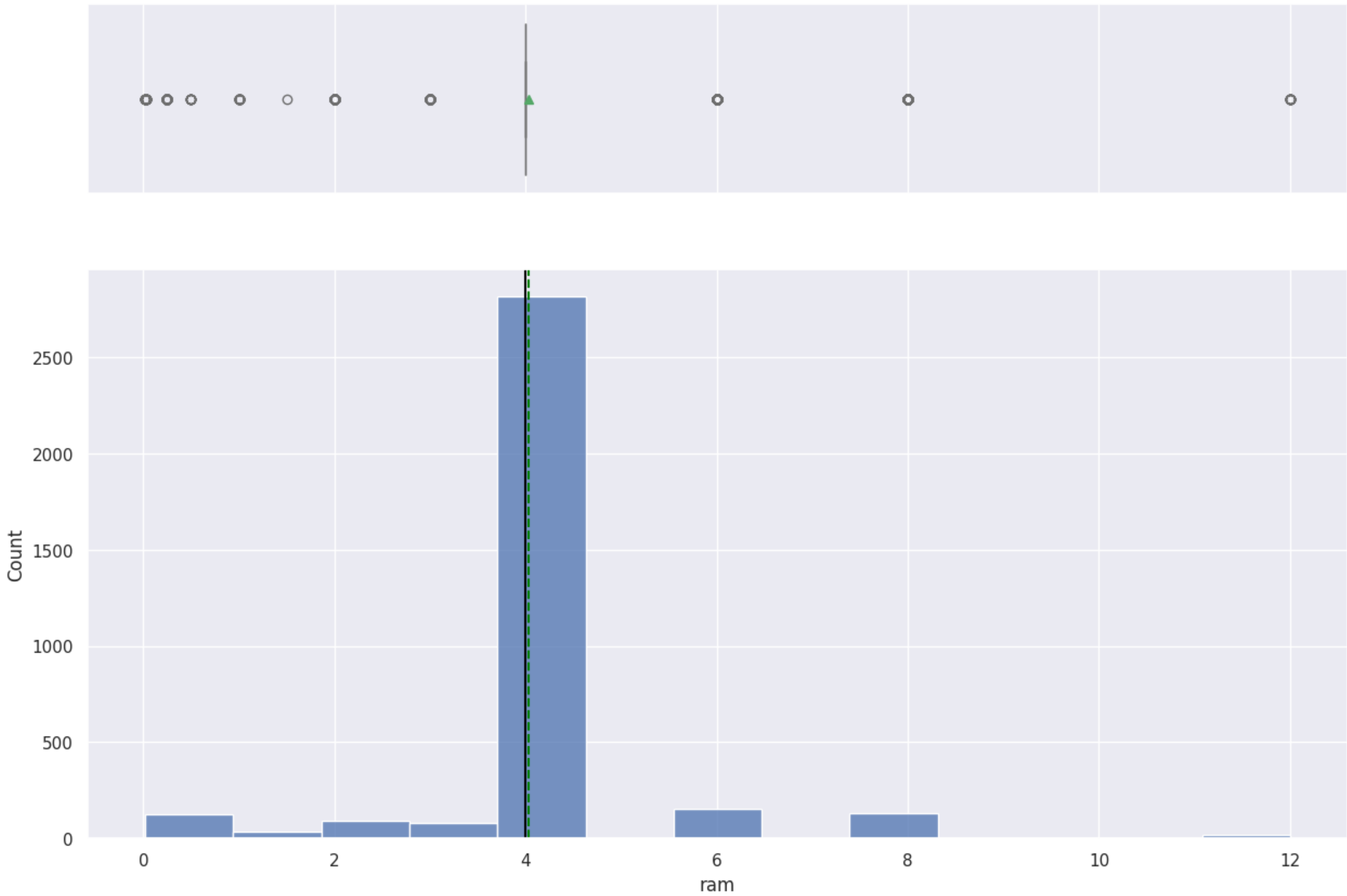
```
#histogram for int_memory
histogram_boxplot(df, "int_memory")
```



Average is around 50 .There are no much outliers in the data

In [ ]:

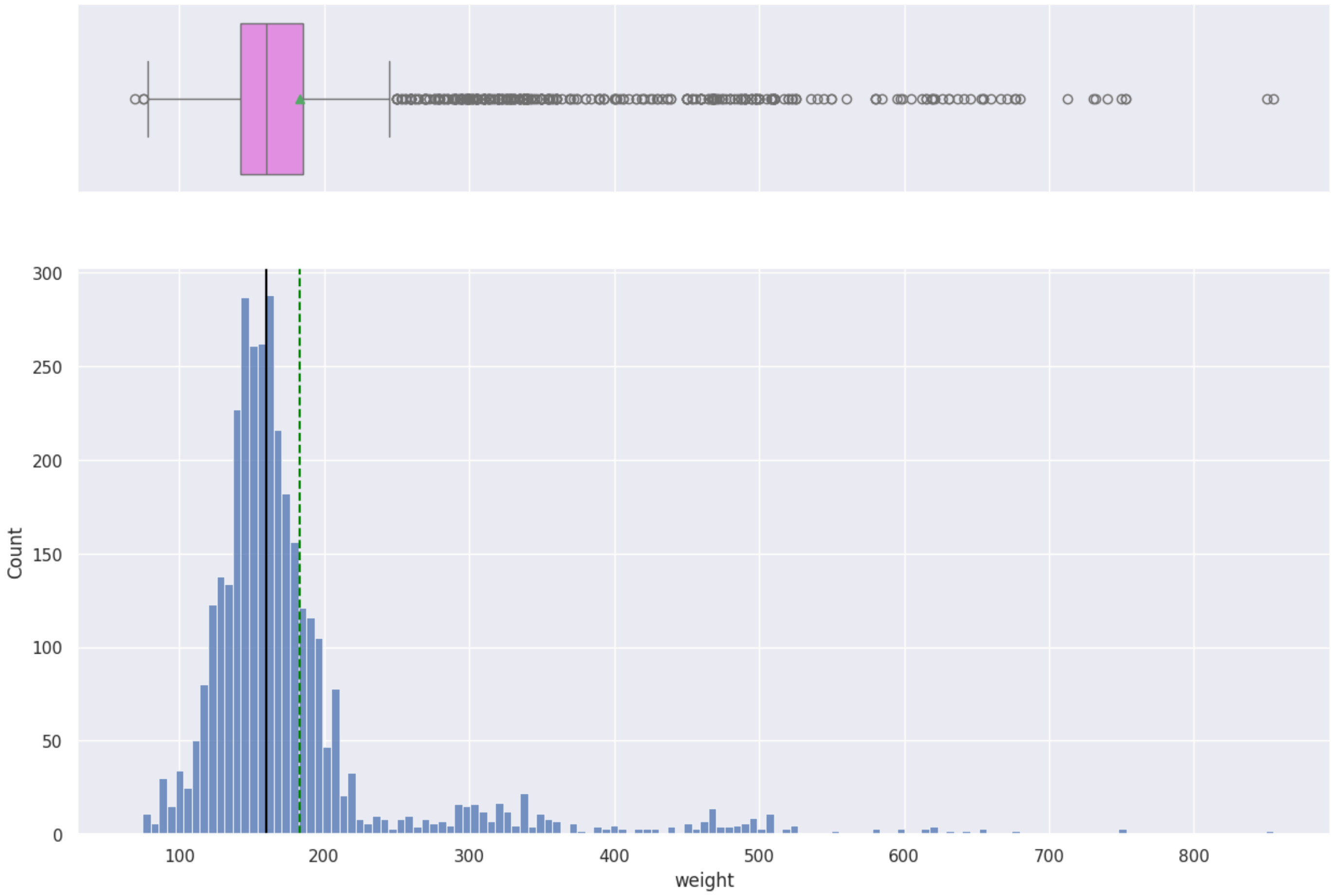
```
#histogram for ram
histogram_boxplot(df, "ram")
```



Average is around 4 and highest count around 2500 phones had 4 as their ram .

In [ ]:

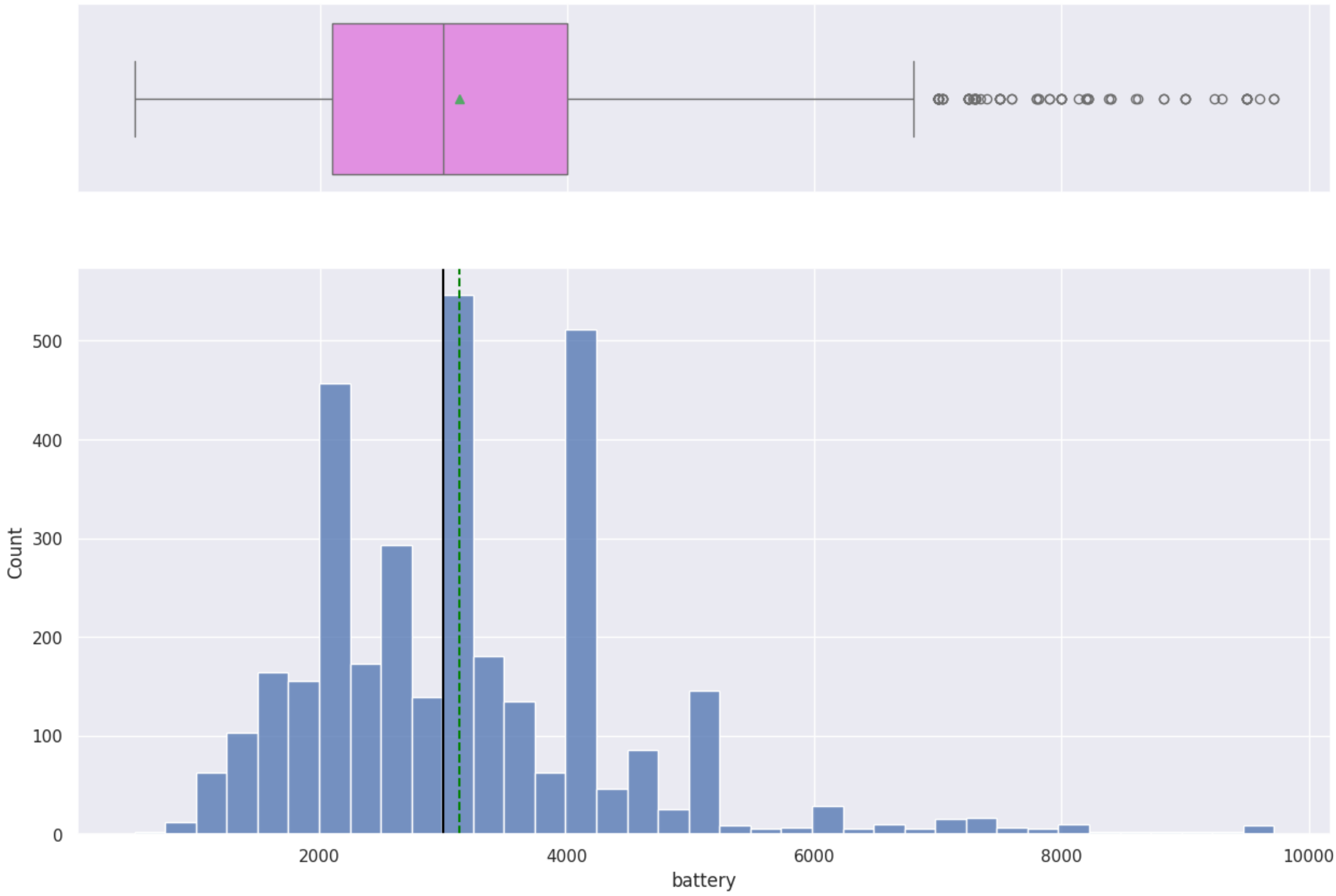
```
#histogram for weight
histogram_boxplot(df, "weight")
```



Average is around 180.Most phones close to 300 had weight of their phones in the range of 160 to 180

In [ ]:

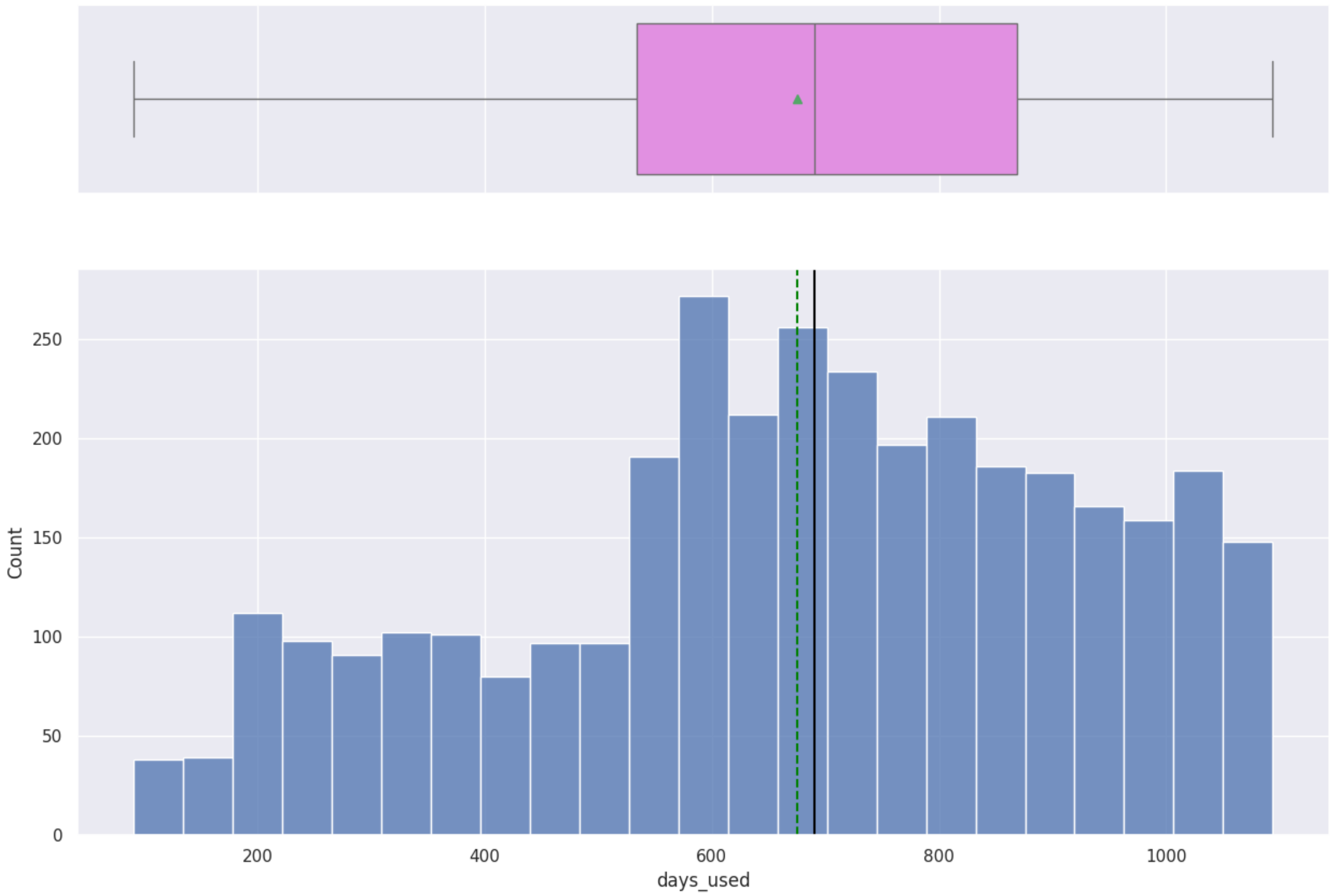
```
#histogram for battery
histogram_boxplot(df, "battery")
```



Average is around 3000 approximately.

In [ ]:

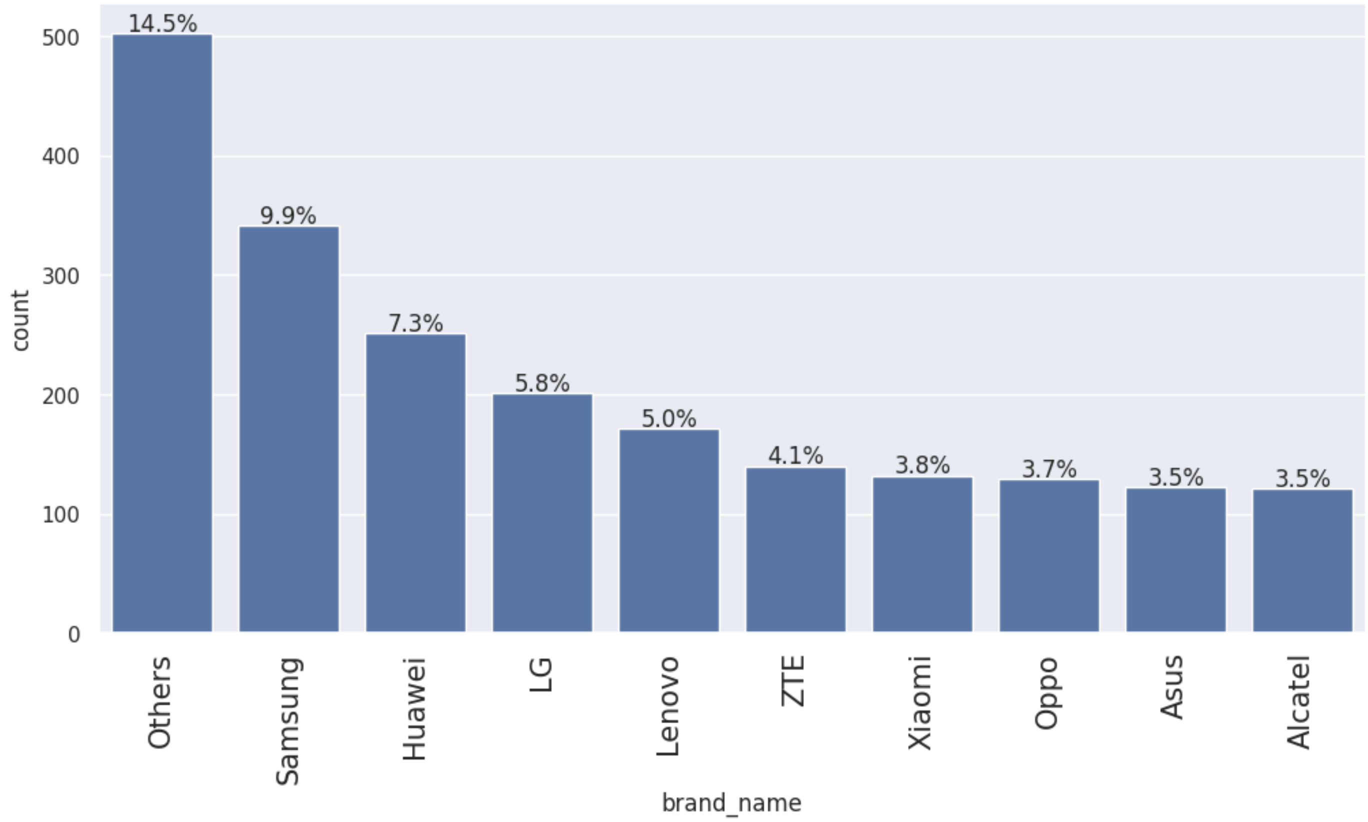
```
#histogram for days_used
histogram_boxplot(df, "days_used")
```



Average is around 670.Around 250 phones (highest) had 600 days of usage before this reselling

In [ ]:

```
#visualizing brand_name
labeled_barplot(df, "brand_name", perc=True, n=10)
```

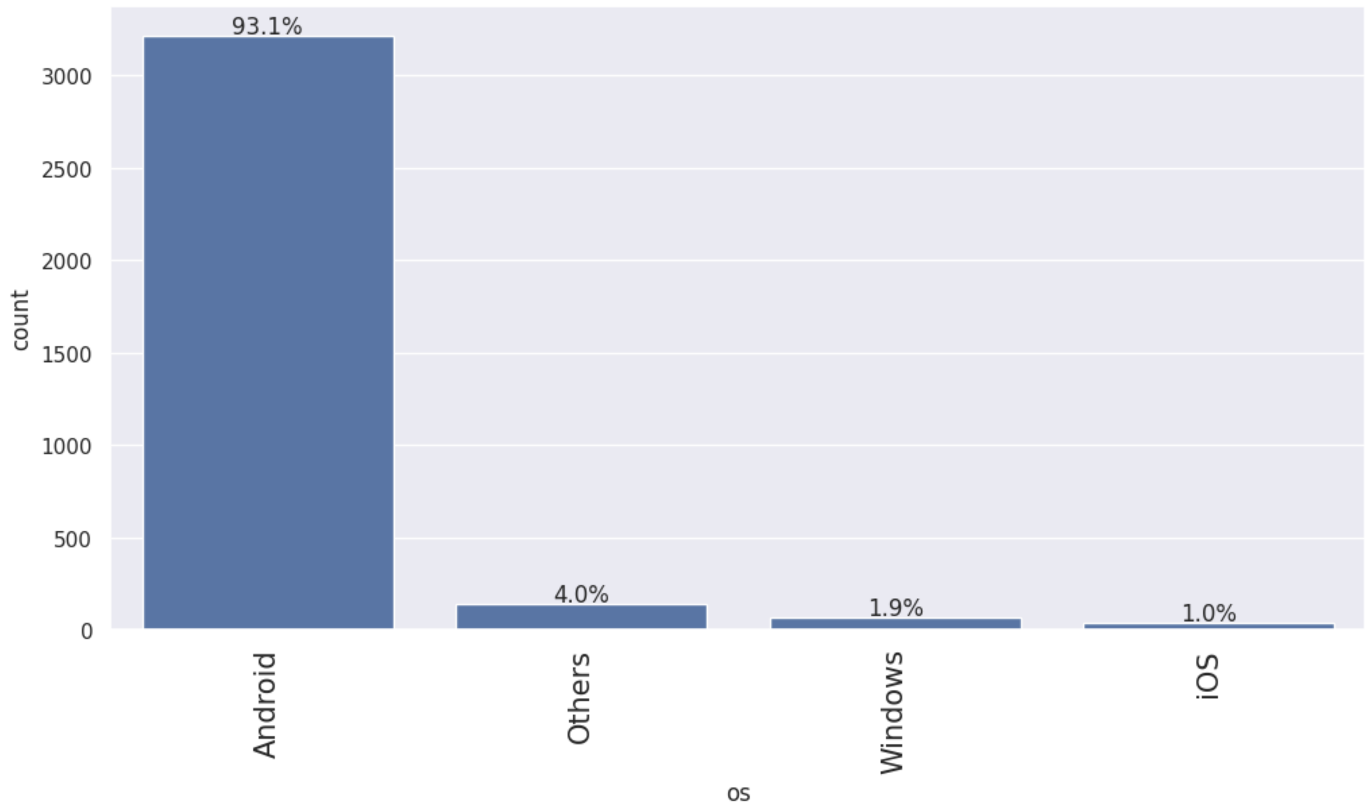


The brands that are sold as part of reselling is mostly the others category and then samsung holds the second place of highest reselling brands .Least is Alcatel .

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

In [ ]:

```
#visualizing os in the used device market
labeled_barplot(df, "os", perc=True, n=10)
```

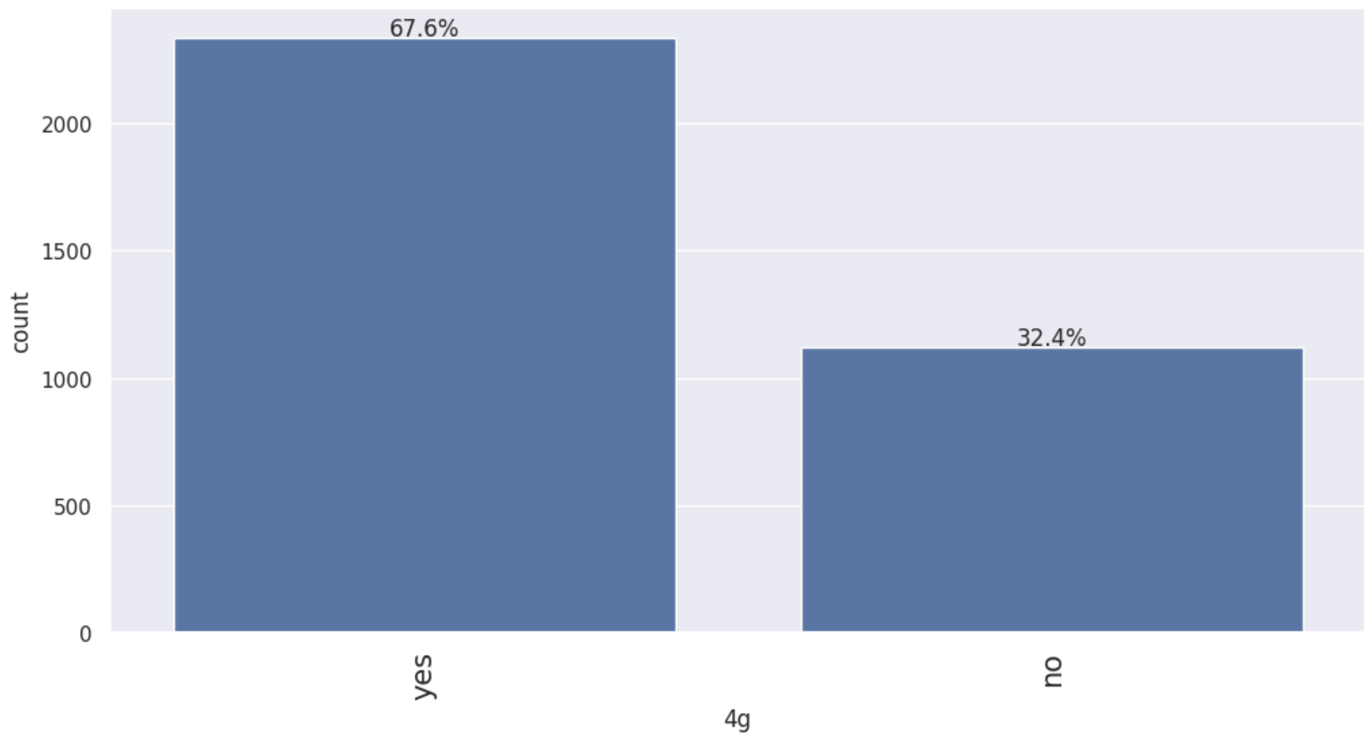


Around 93.1 % of phones that came in to this reselling market (used device market)has android as their os type. ios os type is the least in this market

Lets explore all the columns further

In [ ]:

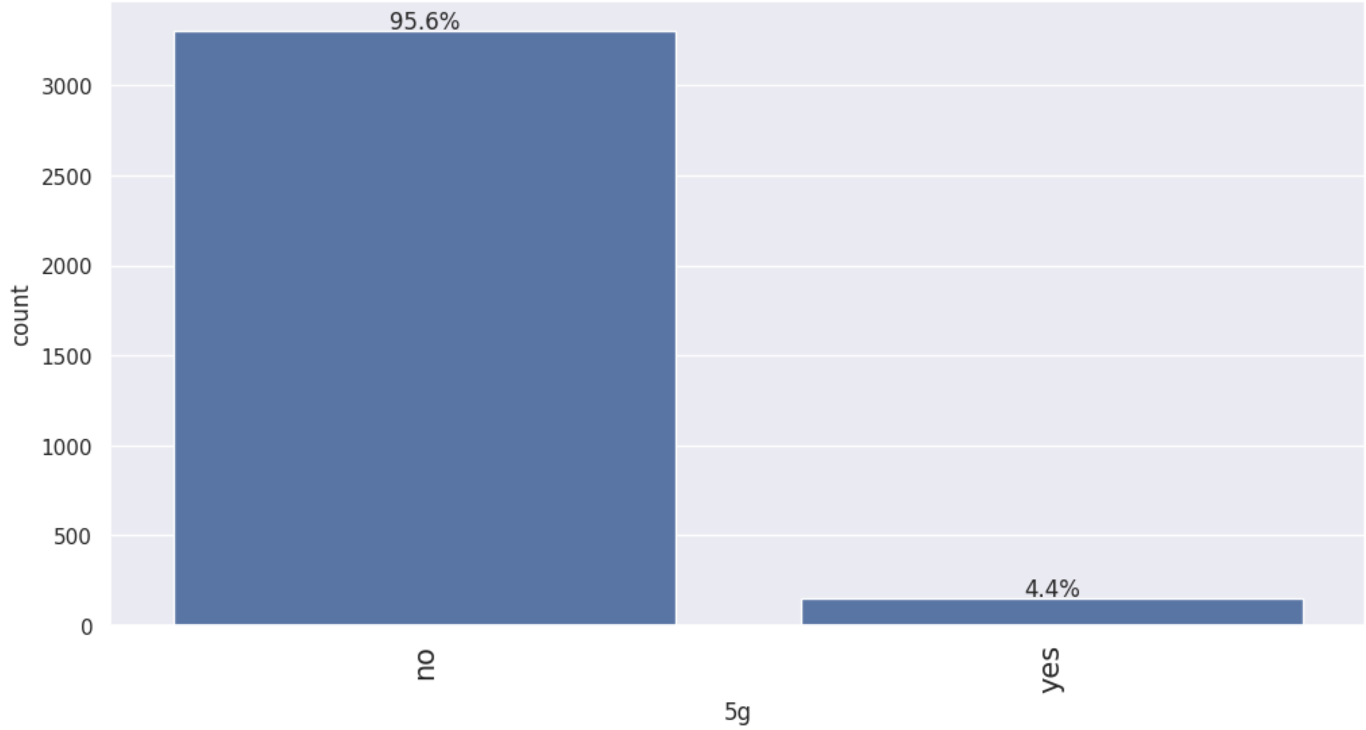
```
#visualizing 4g
labeled_barplot(df, "4g", perc=True, n=10)
```



Around 67.6% of phones in this reselling market had 4g connection and 32.4 percent had no 4g connection

In [ ]:

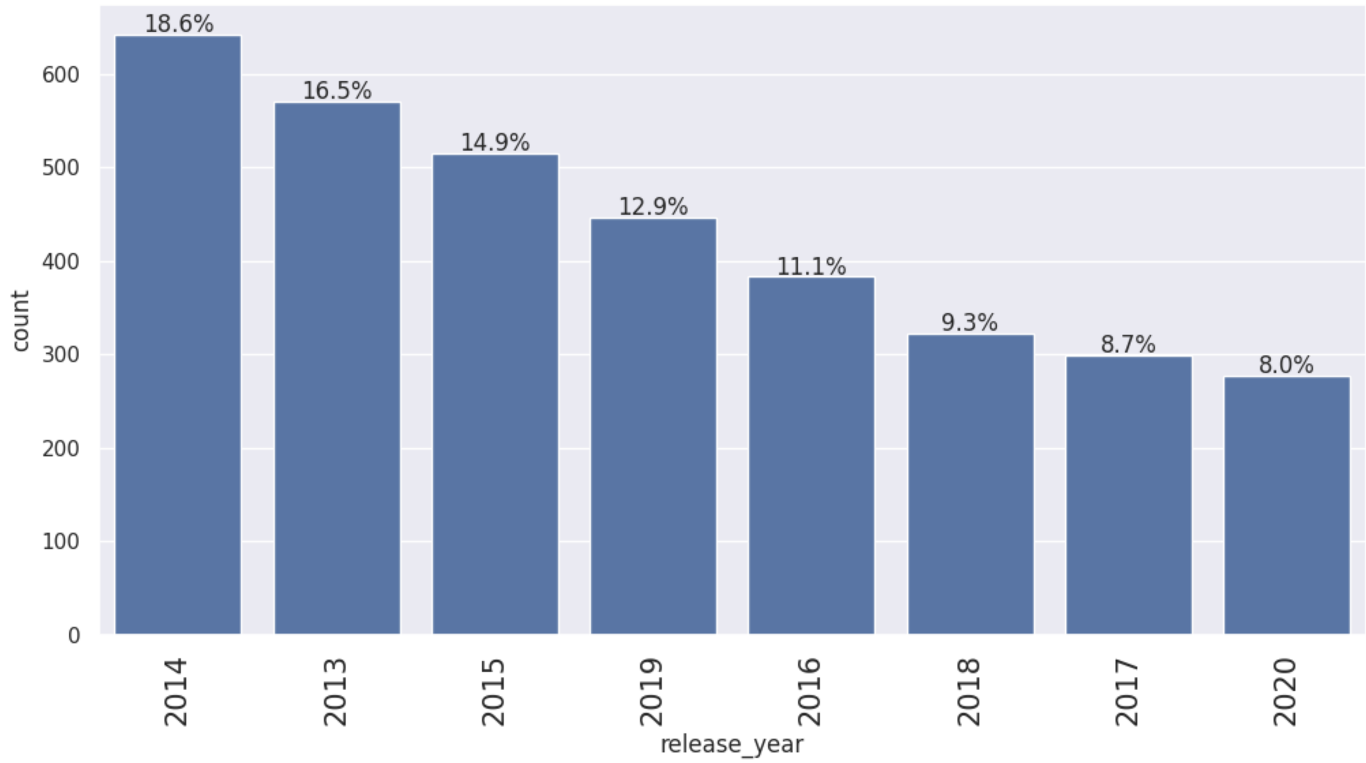
```
#visualizing 5g
labeled_barplot(df, "5g", perc=True, n=10)
```



Around 95 percentage had no 5g connection and only 4 percentage had 5g connection

In [ ]:

```
#visualizing release_year
labeled_barplot(df, "release_year", perc=True, n=10)
```



The phones in the reselling market are mostly released in the year 2014 .The least ones are 2020

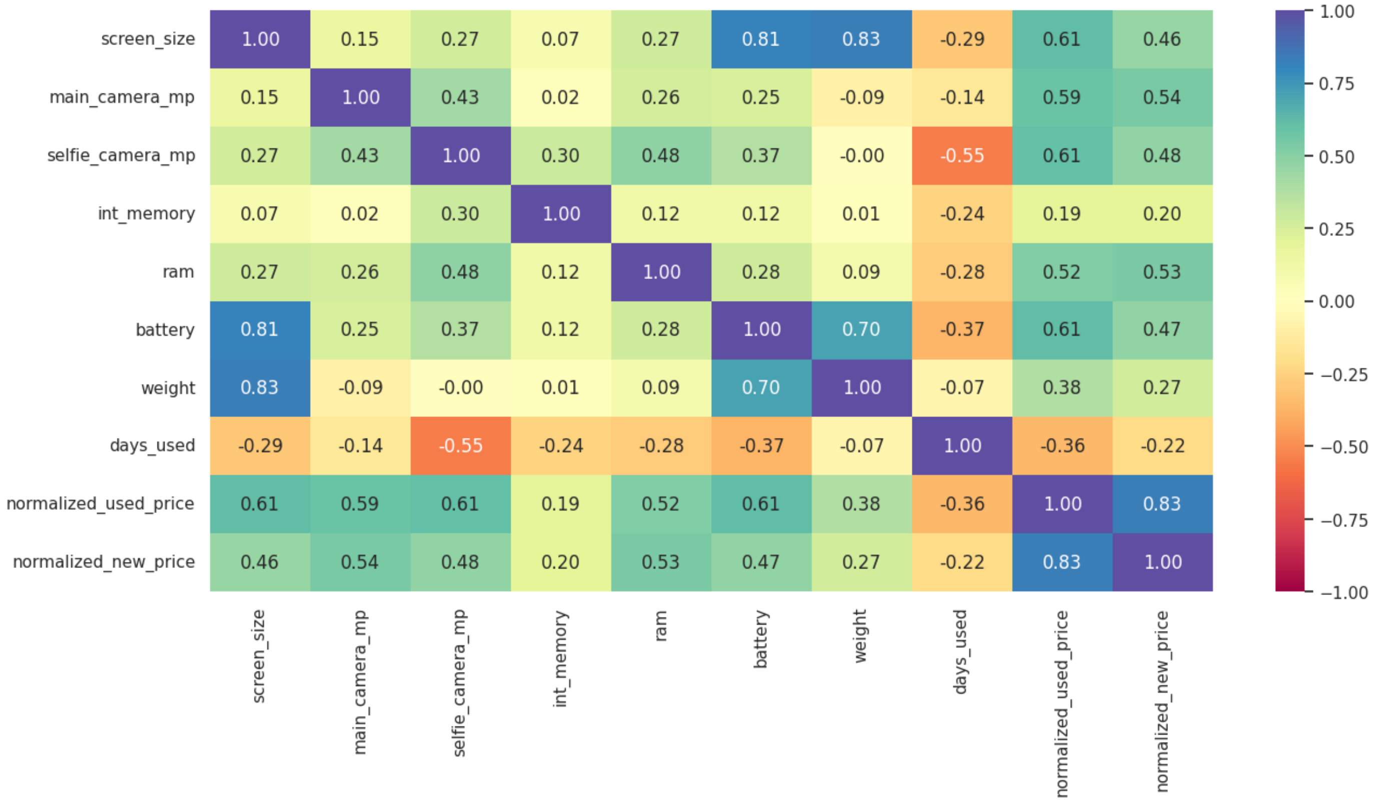
Bivariate Analysis

Which attributes are highly correlated with the normalized price of a used device?

Correlation check

In [ ]:

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

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



normalized_used_price_correlations:

The attributes that are highly positively correlated with normalized_used_price is normalized_new_price which is totally agreed correlation.For each device ,normalized_used_price depends on the normalized_new_price.(0.83)

Screen_size and battery is also positively high correlated with normalized_used_price. As the price of the used device also depends on the battery and screen_size of the device.(0.61)

main_camera_mp and int_memory also seems to have a positive correlation with the normalized_used_price

Other correlations:

Weight and screen_size have high positive correlation

Battery and screen_size have high positive correlation

normalized_new_price and normalized_used_price have high positive correlation

weight and battery have high positive correlation

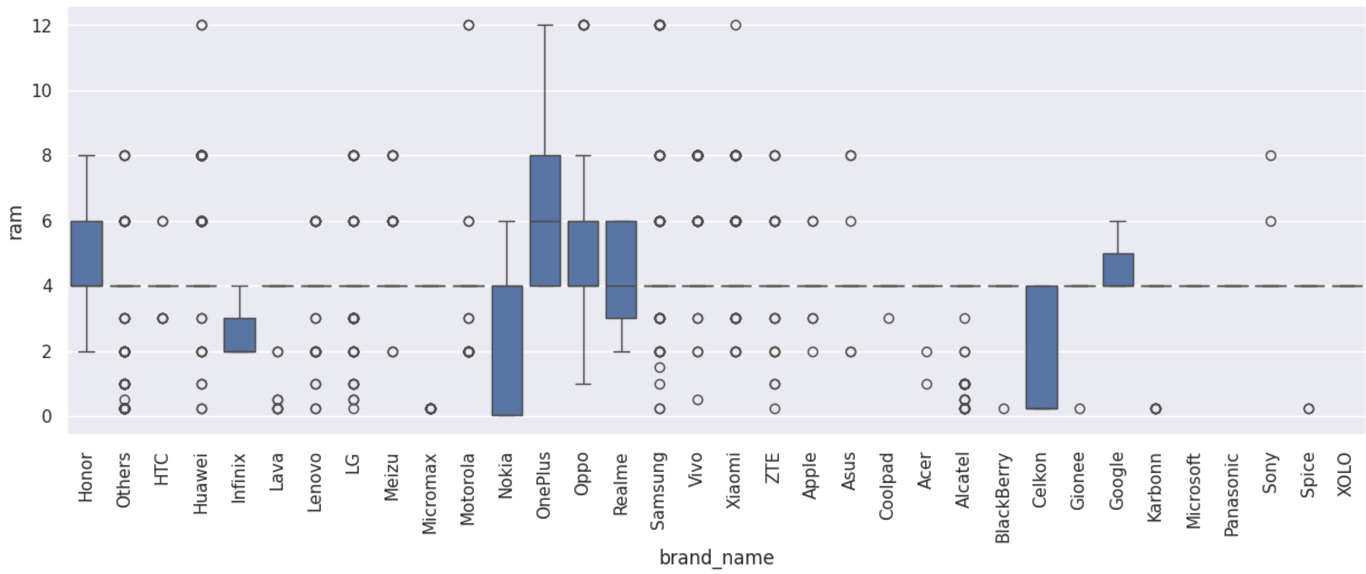
days_used and selfie_camera_mp have high negative correlation

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

RAM vs Brands

In [ ]:

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



Highest RAM is a need nowadays due to the highest usage and storage.

Oneplus have maximum RAM (12) They offer higher RAM devices.And they have the higher median RAM.Seems like honor brand have consistent RAM.

Brands other than honor,huawei,Nokia,Oneplus,Oppo,Realme,Celkon,Google doesnt have a consistent RAM

Average is around 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)?

Large_battery_phones

In [ ]:

```
#dataframe of only large battery greater than 4500
df_large_battery = df[df.battery > 4500]
df_large_battery.shape
```

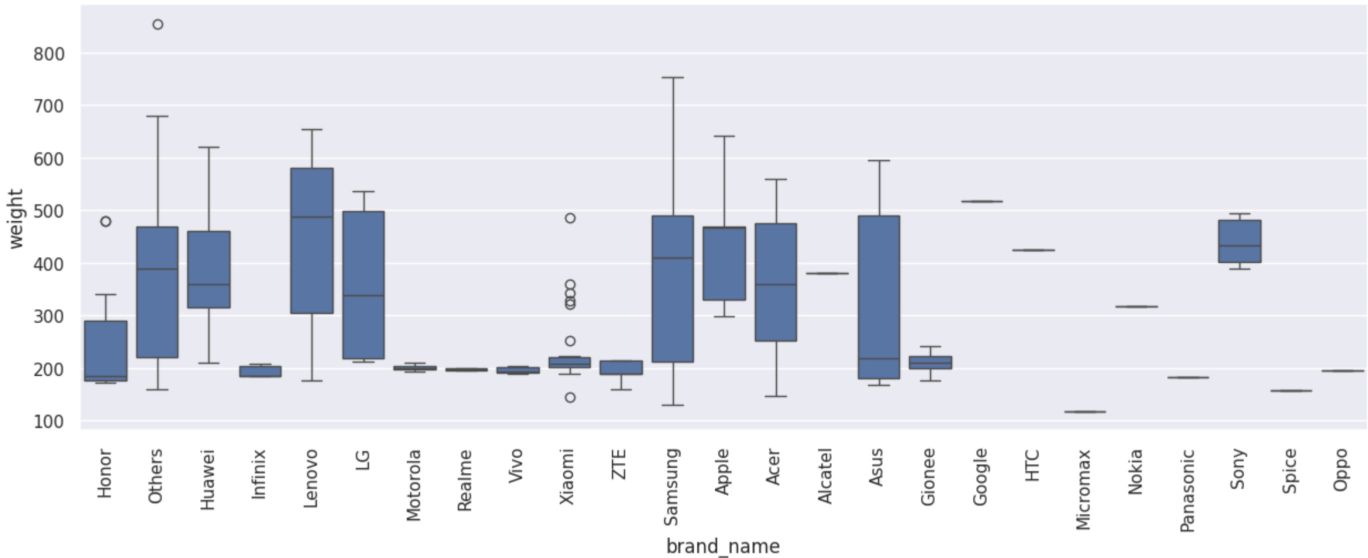
Out[ ]:

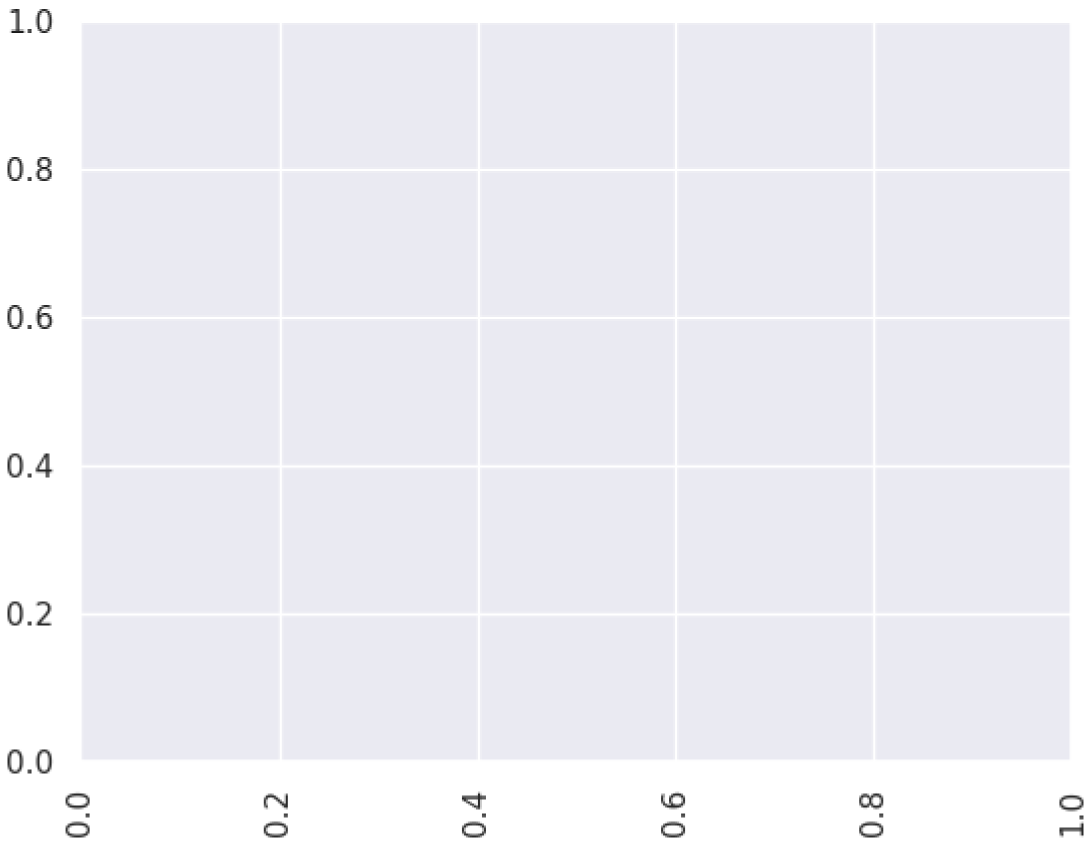
```
(341, 15)
```

There are 341 devices that have battery greater than 4500 mAh

In [ ]:

```
plt.figure(figsize=(15, 5))
sns.boxplot(data=df_large_battery,x='brand_name',y='weight')
plt.xticks(rotation=90)
plt.show()
plt.xticks(rotation=90)
plt.show()
```





So,for all the large battery devices (greater than 4500),samsung has the highest weight .(approximately around 750)

Least weight is in Micromax phones. (around approximately 120)

Mostly all the brands have on an average around 200 to 450 as their weight and large battery.

Brand Xiamo has an inconsistent weight for their devices when they offer large batteries.Sony brand seems to have an consistent weight for their devices while they are providing large batteries.

Lenovo brand has the higher median compared to other devices .This brand's devices have more RAM

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?

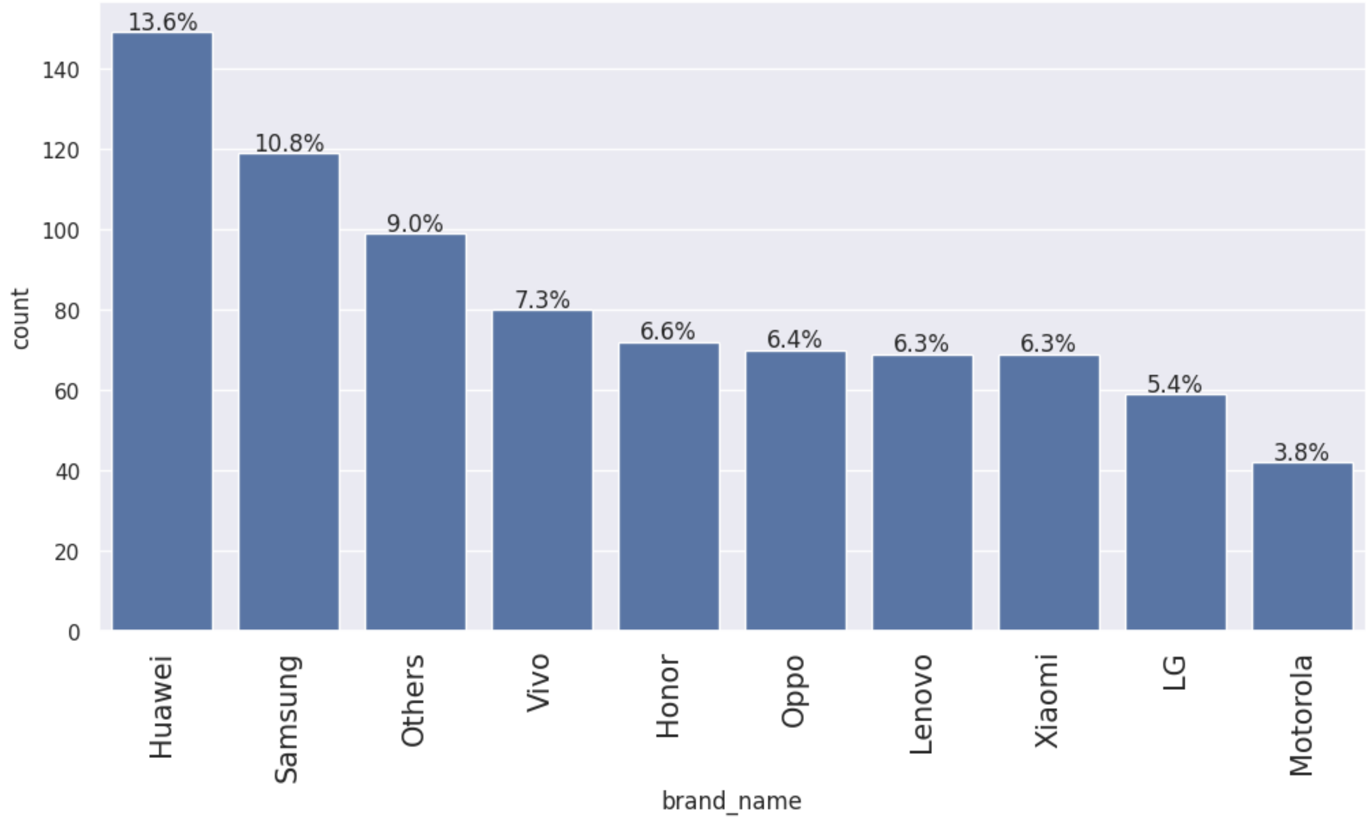
Large_screen_phones

In [ ]:
```
df_large_screen = df[df.screen_size > 6 * 2.54]
df_large_screen.shape
```
Out[ ]:
```
(1099, 15)
```

There are 1099 phones and tablets across different brands with screen size greater than 6 inches

In [ ]:
```
#visualizing brand_name in large screen phones
```

```
labeled_barplot(df_large_screen, "brand_name", perc=True, n=10)
```



People usually prefer large screens ..The brand Huawei provide the largest screen sizes(13.6% of large screen size phones are from Huawei).Samsung stands next with 10.8 %..Brand which offers less screen size is Motoria compared to other brands and their screen sizes

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?

Good_selfie_camera_phones

In [ ]:

```
df_selfie_camera = df[df.selfie_camera_mp > 8]
df_selfie_camera.shape
```

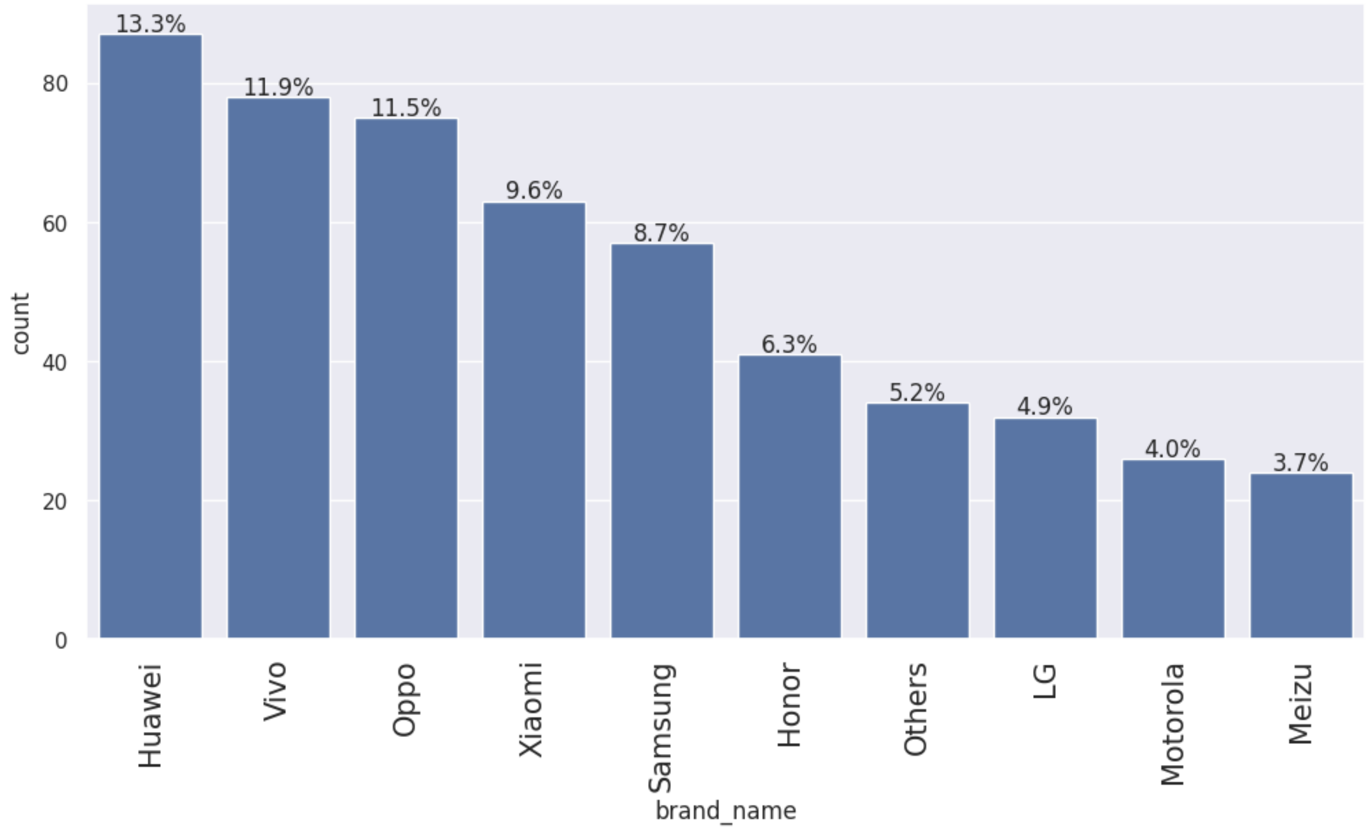
Out[ ]:

```
(655, 15)
```

There are 655 devices that offer greater than 8MP Selfie cameras

In [ ]:

```
#visualizing brand_name in greater selfie camera mp
labeled_barplot(df_selfie_camera, "brand_name", perc=True, n=10)
```



Again ,Huawei stands the brand with most quality selfie camera mp .Vivo takes the second place in providing best selfie camera mp .Meizu stands the brand with less selfie camera mp.

Real_camera_quality_phones

In [ ]:

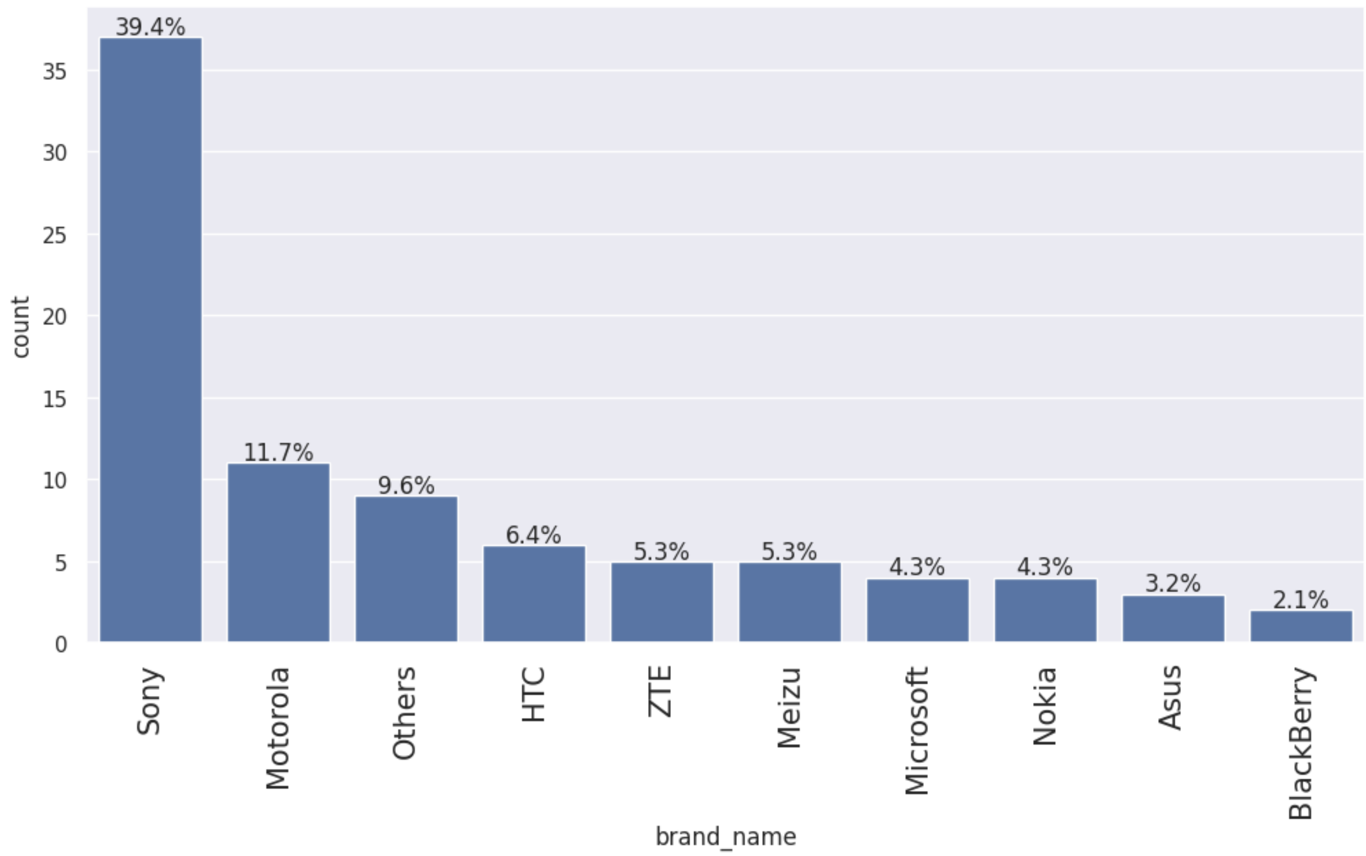
```
df_main_camera = df[df.main_camera_mp > 16]
df_main_camera.shape
```

Out[ ]:

```
(94, 15)
```

In [ ]:

```
#visualizing brand_name in greater main camera mp
labeled_barplot(df_main_camera, "brand_name", perc=True, n=10)
```

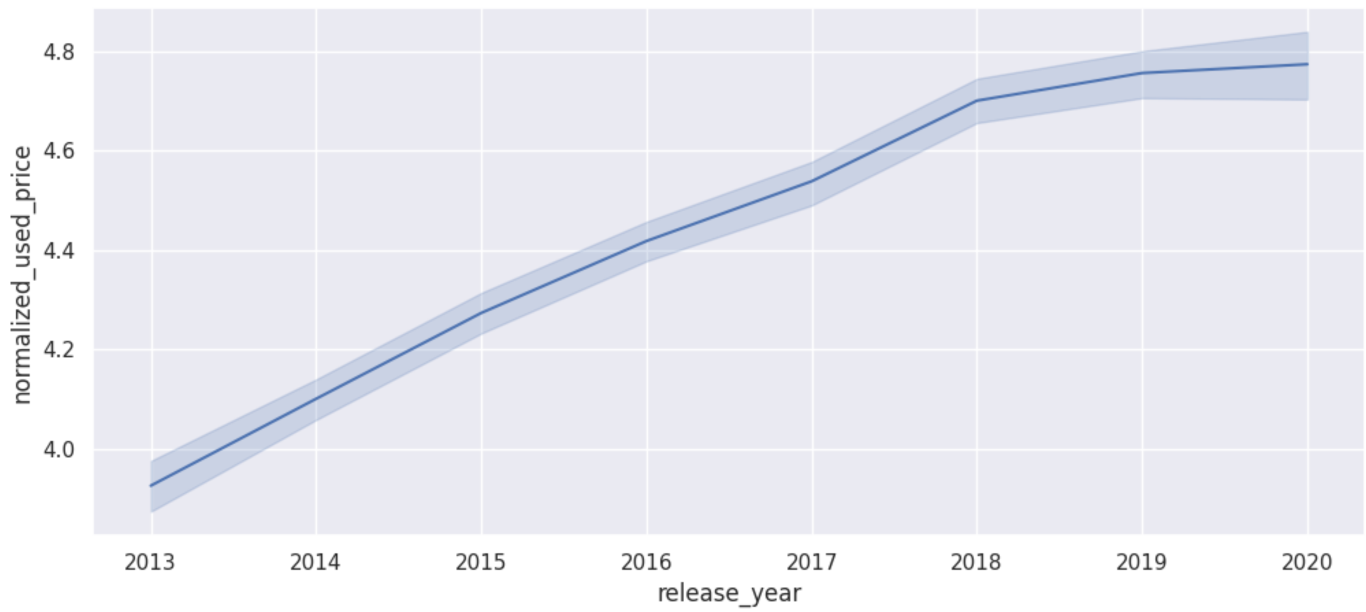


Sony brands stands the highest with 39.4 % for providing best main camera quality.Blackberry brands stands the least with 2.1% compared to other brands in providing best main camera quality

Prices_of_used_phones

In [ ]:

```
plt.figure(figsize=(12, 5))
sns.lineplot(x='release_year',y='normalized_used_price',data=df) ## Complete the code to
plt.show()
```

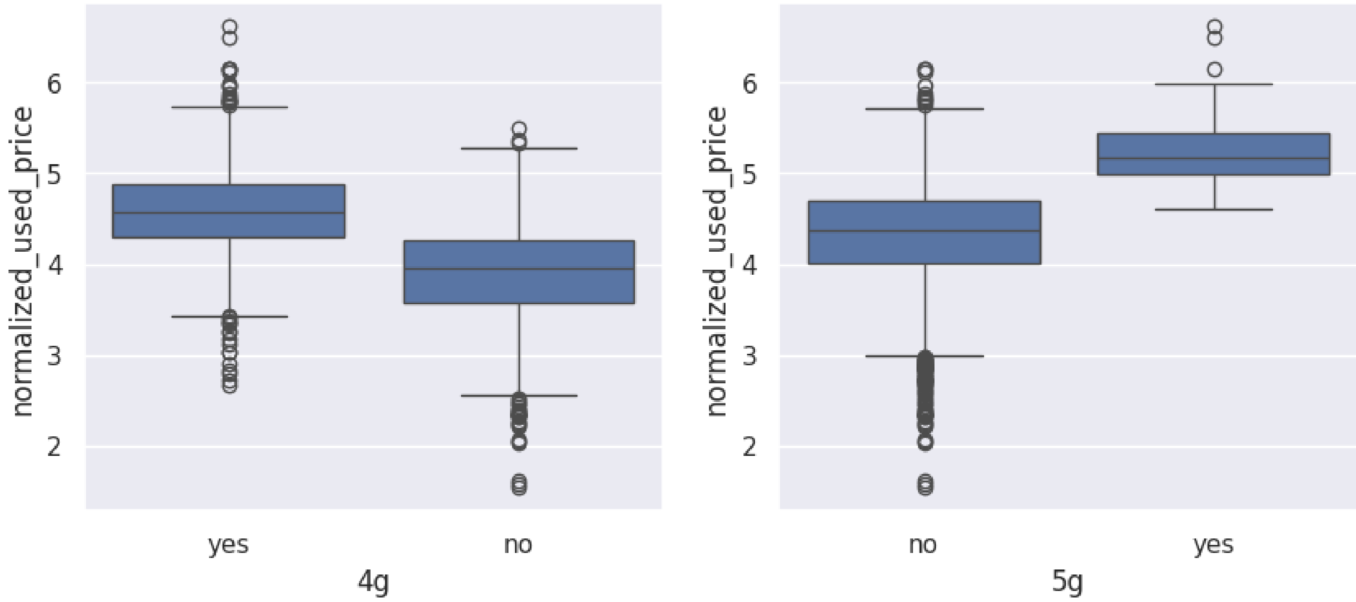


Prices of used phones have got higher over the years from 2013 to 2020.There is no depreciation in the prices after 2013 .

Prices_of_used_phones_4g_5g

In [ ]:

```
plt.figure(figsize=(10, 4))

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

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

plt.show()
```



The prices of used 4g phones is around 4 to 5 and the prices of used 5g phones is around 5 to 6 (normalized prices)

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)

Missing_value_treatment_with_median

In [ ]:

```
# let's create a copy of the data
df1 = df.copy()
```

In [ ]:

```
df1.isnull().sum()
```

Out[ ]:

	0
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

In [ ]:

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

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

In [ ]:

```
# checking for missing values
df1.isnull().sum() ## Complete the code to check missing values after imputing the above
```

Out[ ]:

	0
brand_name	0
os	0
screen_size	0
4g	0

	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
normalized_used_price	0
normalized_new_price	0

dtype: int64

We still have missing values.so treating them with grouping by on brand_name

In [ ]:

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

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

In [ ]:

```
# checking for missing values
df1.isnull().sum() ## Complete the code to check missing values after imputing the above
```

Out[ ]:

	0
brand_name	0
os	0
screen_size	0
4g	0
5g	0
main_camera_mp	10
selfie_camera_mp	0
int_memory	0

	0
ram	0
battery	0
weight	0
release_year	0
days_used	0
normalized_used_price	0
normalized_new_price	0

dtype: int64

We still have missing values in main_camera_mp column.Treating them by replacing it with its median

In [ ]:

```
df1["main_camera_mp"] = df1["main_camera_mp"].fillna(df1["main_camera_mp"].median())
```

In [ ]:

```
# checking for missing values
df1.isnull().sum() ## Complete the code to check missing values after imputing the above
```

Out[ ]:

	0
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

All the missing values are treated

Creating a new column -Feature Engineering

In [ ]:

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

Out[ ]:

	years_since_release
count	3454.000000
mean	5.034742
std	2.298455
min	1.000000
25%	3.000000
50%	5.500000
75%	7.000000
max	8.000000

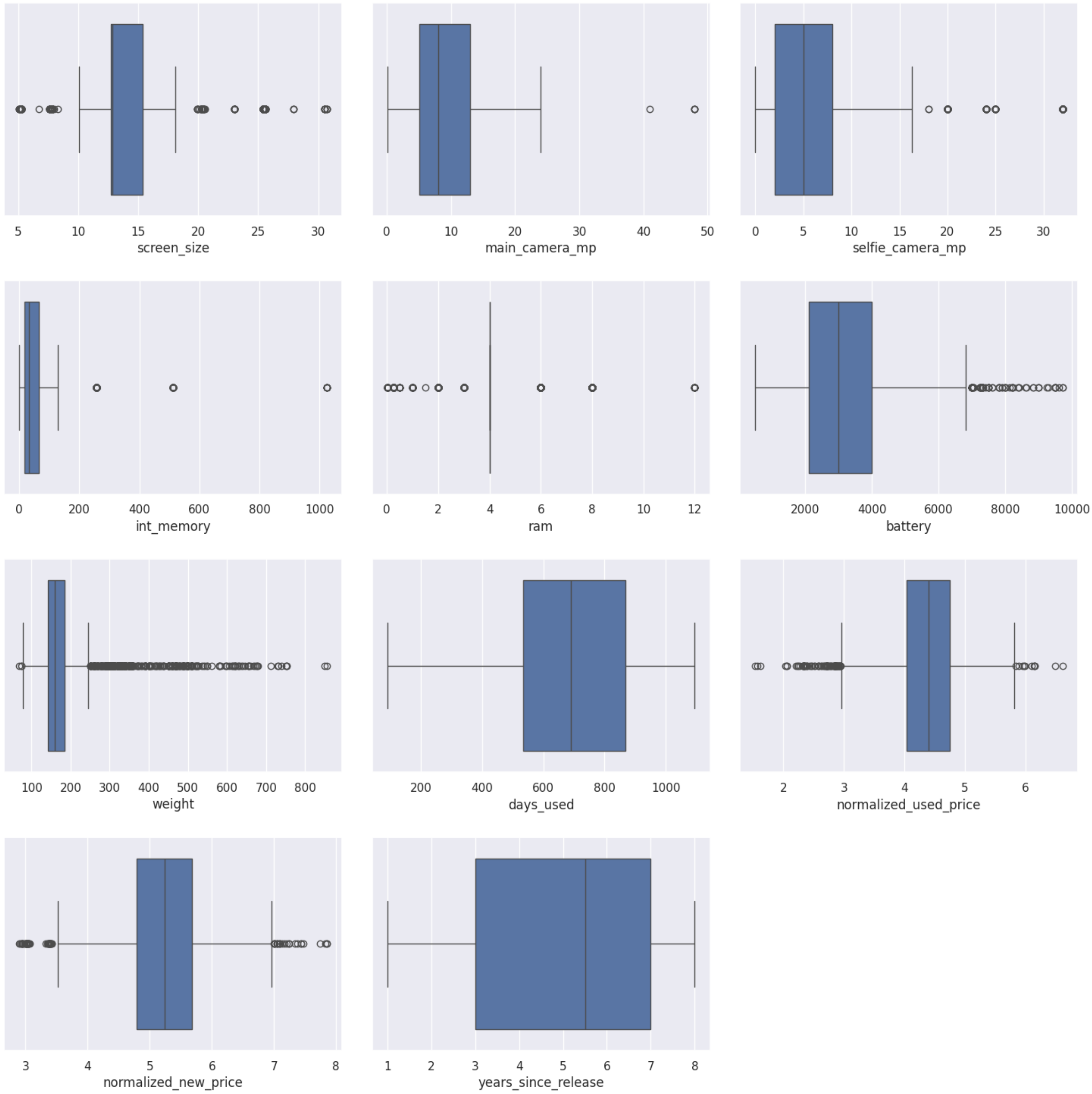
dtype: float64

we have taken 2021 as a highest limit to calculate the years_since_release. On an average ,the phones in this market are 5 years old from its release year.Minimum is 1 year and maximum is 8 years.

Outliers_check

In [ ]:

```
# outlier detection using boxplot
num_cols = df1.select_dtypes(include=np.number).columns.tolist()

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

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

plt.show()
```



EDA

In [ ]:

```
df1.head()
```

Out[ ]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	ram
0	Honor	Android	14.50	yes	no	13.0	5.0	64.0	3.0
1	Honor	Android	17.30	yes	yes	13.0	16.0	128.0	8.0
2	Honor	Android	16.69	yes	yes	13.0	8.0	128.0	8.0
3	Honor	Android	25.50	yes	yes	13.0	8.0	64.0	6.0
4	Honor	Android	15.32	yes	no	13.0	8.0	64.0	3.0

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

Model Building - Linear Regression

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

In [ ]:

```
# defining X and y variables
X = df1.drop(["normalized_used_price"], axis=1)
y = df1["normalized_used_price"]

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

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

   selfie_camera_mp  int_memory  ram  battery  weight  days_used  \
0               5.0        64.0  3.0   3020.0   146.0        127
1              16.0       128.0  8.0   4300.0   213.0        325
2               8.0       128.0  8.0   4200.0   213.0        162
3               8.0        64.0  6.0   7250.0   480.0        345
4               8.0        64.0  3.0   5000.0   185.0        293

   normalized_new_price  years_since_release
0              4.715100                    1
1              5.519018                    1
2              5.884631                    1
3              5.630961                    1
4              4.947837                    1
0    4.307572
1    5.162097
2    5.111084
3    5.135387
4    4.389995
Name: normalized_used_price, dtype: float64
```

The dependent variable is set to y which is the normalized used price.Other attributes are independent variables which are denoted by x

In [ ]:

```
# let's add the intercept to data
X = sm.add_constant(X)
```

In [ ]:

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

Out[ ]:

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	days_us
0	1.0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	1
1	1.0	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	3
2	1.0	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	1
3	1.0	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	3
4	1.0	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	2

5 rows × 49 columns

In [ ]:

```
# converting the input attributes into float type for modeling
X = X.astype(float)
X.head()
```

Out[ ]:

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	days_us
0	1.0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	12
1	1.0	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	32
2	1.0	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	16
3	1.0	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	34
4	1.0	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	29

5 rows × 49 columns

In [ ]:

```
# splitting the data in 70:30 ratio for train to test data

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

We have added constant to the matrix and we have added dummy variables to all the categorical attributes. and we have splitted the train data 70 % and test data 30%

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

In [ ]:

```
olsmodel = sm.OLS(y_train, x_train).fit()
print(olsmodel.summary())
```

```
                            OLS Regression Results
=================================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.845
Model:                               OLS   Adj. R-squared:                  0.842
Method:                    Least Squares   F-statistic:                     268.7
Date:                   Sun, 13 Apr 2025   Prob (F-statistic):               0.00
Time:                           15:17:56   Log-Likelihood:                 123.85
No. Observations:                   2417   AIC:                            -149.7
Df Residuals:                       2368   BIC:                             134.0
Df Model:                             48
Covariance Type:               nonrobust
=========================================================================================
=
                            coef    std err          t      P>|t|      [0.025      0.97
5]
-----------------------------------------------------------------------------------------
-
const                     1.3156      0.071     18.454      0.000       1.176       1.45
5
screen_size               0.0244      0.003      7.163      0.000       0.018       0.03
1
main_camera_mp            0.0208      0.002     13.848      0.000       0.018       0.02
4
selfie_camera_mp          0.0135      0.001     11.997      0.000       0.011       0.01
6
int_memory                0.0001   6.97e-05      1.651      0.099   -2.16e-05       0.00
0
ram                       0.0230      0.005      4.451      0.000       0.013       0.03
3
battery               -1.689e-05   7.27e-06     -2.321      0.020   -3.12e-05   -2.62e-0
6
weight                    0.0010      0.000      7.480      0.000       0.001       0.00
1
days_used              4.216e-05   3.09e-05      1.366      0.172   -1.84e-05       0.00
0
normalized_new_price      0.4311      0.012     35.147      0.000       0.407       0.45
5
years_since_release      -0.0237      0.005     -5.193      0.000      -0.033      -0.01
5
brand_name_Alcatel        0.0154      0.048      0.323      0.747      -0.078       0.10
9
brand_name_Apple         -0.0038      0.147     -0.026      0.980      -0.292       0.28
5
brand_name_Asus           0.0151      0.048      0.314      0.753      -0.079       0.10
9
brand_name_BlackBerry    -0.0300      0.070     -0.427      0.669      -0.168       0.10
8
brand_name_Celkon        -0.0468      0.066     -0.707      0.480      -0.177       0.08
3
brand_name_Coolpad        0.0209      0.073      0.287      0.774      -0.122       0.16
4
brand_name_Gionee         0.0448      0.058      0.775      0.438      -0.068       0.15
8
brand_name_Google        -0.0326      0.085     -0.385      0.700      -0.199       0.13
3
```

brand_name_HTC	-0.0130	0.048	-0.270	0.787	-0.108	0.081
brand_name_Honor	0.0317	0.049	0.644	0.520	-0.065	0.128
brand_name_Huawei	-0.0020	0.044	-0.046	0.964	-0.089	0.085
brand_name_Infinix	0.1633	0.093	1.752	0.080	-0.019	0.346
brand_name_Karbonn	0.0943	0.067	1.405	0.160	-0.037	0.226
brand_name_LG	-0.0132	0.045	-0.291	0.771	-0.102	0.076
brand_name_Lava	0.0332	0.062	0.533	0.594	-0.089	0.155
brand_name_Lenovo	0.0454	0.045	1.004	0.316	-0.043	0.134
brand_name_Meizu	-0.0129	0.056	-0.230	0.818	-0.123	0.097
brand_name_Micromax	-0.0337	0.048	-0.704	0.481	-0.128	0.060
brand_name_Microsoft	0.0952	0.088	1.078	0.281	-0.078	0.268
brand_name_Motorola	-0.0112	0.050	-0.226	0.821	-0.109	0.086
brand_name_Nokia	0.0719	0.052	1.387	0.166	-0.030	0.174
brand_name_OnePlus	0.0709	0.077	0.916	0.360	-0.081	0.223
brand_name_Oppo	0.0124	0.048	0.261	0.794	-0.081	0.106
brand_name_Others	-0.0080	0.042	-0.190	0.849	-0.091	0.075
brand_name_Panasonic	0.0563	0.056	1.008	0.314	-0.053	0.166
brand_name_Realme	0.0319	0.062	0.518	0.605	-0.089	0.153
brand_name_Samsung	-0.0313	0.043	-0.725	0.469	-0.116	0.053
brand_name_Sony	-0.0616	0.050	-1.220	0.223	-0.161	0.037
brand_name_Spice	-0.0147	0.063	-0.233	0.816	-0.139	0.109
brand_name_Vivo	-0.0154	0.048	-0.318	0.750	-0.110	0.080
brand_name_XOLO	0.0152	0.055	0.277	0.782	-0.092	0.123
brand_name_Xiaomi	0.0869	0.048	1.806	0.071	-0.007	0.181
brand_name_ZTE	-0.0057	0.047	-0.121	0.904	-0.099	0.087
os_Others	-0.0510	0.033	-1.555	0.120	-0.115	0.013
os_Windows	-0.0207	0.045	-0.459	0.646	-0.109	0.068
os_iOS	-0.0663	0.146	-0.453	0.651	-0.354	0.221
4g_yes	0.0528	0.016	3.326	0.001	0.022	0.084
5g_yes	-0.0714	0.031	-2.268	0.023	-0.133	-0.01

```
0
==============================================================================
Omnibus:                      223.612   Durbin-Watson:                   1.910
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              422.275
Skew:                          -0.620   Prob(JB):                     2.01e-92
Kurtosis:                       4.630   Cond. No.                     1.78e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[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:

R-squared: 0.85 This explains model explains around 85 percent about the variance of the target variable based on the independent variables

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 0.842, which is good.

constcoefficient: 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 1.3156

Model Performance Check

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

We will be using metric functions defined in sklearn for RMSE, MAE, and R2 .

We will define a function to calculate MAPE and adjusted R2 .

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 which will print out all the above metrics in one go.

In [ ]:

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


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

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

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

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

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

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

    return df_perf
```

In [ ]:

```python
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel_train_perf = model_performance_regression(olsmodel, x_train, y_train)
olsmodel_train_perf
```

```
Training Performance
```

Out[ ]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.229884	0.180326	0.844886	0.841675	4.326841

In [ ]:

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

```
Test Performance
```

Out[ ]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.238358	0.184749	0.842479	0.834659	4.501651

RMSE measures the square root of the average squared differences between the actual and predicted values. A lower RMSE indicates a better model.

Train RMSE =0.22 on average, the model's predictions on the test data are off by about 0.24 units. Test RMSE = 0.23

There is only a slight increase of RMSE in test data whch is not a big concern.

MAE measures the average of the absolute differences between the predicted and actual values.

The training MAE was 0.180326, so the test MAE is slightly higher, but again, this is normal. The difference between the two is minimal, which indicates that the model is consistent across the training and test datasets.

R^2 measures the proportion of the variance in the target variable that is explained by the independent variables in the model. Train R^2 = 0.844886 Test R^2 = 0.842479

The model explains 84 percent about the variance of the target variable and there is a very slight difference between the test and train R^2 which explains the model is robust

Adjusted R^2 is a version of R^2 that adjusts for the number of predictors. Train adjusted R^2 = 0.84 Test adjusted R^2 = 0.83

There is a very slight difference between the test and train adjusted R^2 which explains the model is robust

MAPE measures the average absolute percentage difference between the predicted and actual values

Train MAPE = 4.3 Test MAPE = 4.5

There is a very slight difference between the test and train adjusted R^2 which explains the model is robust

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.

We will be checking the following **Linear Regression assumptions**:

No Multicollinearity

Linearity of variables

Independence of error terms

Normality of error terms

No Heteroscedasticity

TEST FOR MULTICOLLINEARITY

We will test for multicollinearity using VIF.

General Rule of thumb:

If VIF is 1 then there is no correlation between the k th predictor and the remaining predictor variables.

If VIF exceeds 5 or is close to exceeding 5, we say there is moderate multicollinearity.

If VIF is 10 or exceeding 10, it shows signs of high multicollinearity. Let's define a function to check VIF.

In [ ]:

```
#Let's define a function to check VIF.

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

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

In [ ]:

```
checking_vif(x_train)  ## Complete the code to check VIF on train data
```

Out[ ]:

	feature	VIF
0	const	227.744081
1	screen_size	7.677290
2	main_camera_mp	2.285051
3	selfie_camera_mp	2.812473
4	int_memory	1.364152
5	ram	2.282352
6	battery	4.081780
7	weight	6.396749
8	days_used	2.660269
9	normalized_new_price	3.119430
10	years_since_release	4.899007
11	brand_name_Alcatel	3.405693
12	brand_name_Apple	13.057668
13	brand_name_Asus	3.332038

	feature	VIF
14	brand_name_BlackBerry	1.632378
15	brand_name_Celkon	1.774721
16	brand_name_Coolpad	1.468006
17	brand_name_Gionee	1.951272
18	brand_name_Google	1.321778
19	brand_name_HTC	3.410361
20	brand_name_Honor	3.340687
21	brand_name_Huawei	5.983852
22	brand_name_Infinix	1.283955
23	brand_name_Karbonn	1.573702
24	brand_name_LG	4.849832
25	brand_name_Lava	1.711360
26	brand_name_Lenovo	4.558941
27	brand_name_Meizu	2.179607
28	brand_name_Micromax	3.363521
29	brand_name_Microsoft	1.869751
30	brand_name_Motorola	3.274558
31	brand_name_Nokia	3.479849
32	brand_name_OnePlus	1.437034
33	brand_name_Oppo	3.971194
34	brand_name_Others	9.711034
35	brand_name_Panasonic	2.105703
36	brand_name_Realme	1.946812
37	brand_name_Samsung	7.539866
38	brand_name_Sony	2.943161
39	brand_name_Spice	1.688863
40	brand_name_Vivo	3.651437
41	brand_name_XOLO	2.138070
42	brand_name_Xiaomi	3.719689
43	brand_name_ZTE	3.797581
44	os_Others	1.859863
45	os_Windows	1.596034
46	os_iOS	11.784684
47	4g_yes	2.467681
48	5g_yes	1.813900

There are multiple columns with very high VIF values, indicating presence of strong multicollinearity

We will systematically drop numerical columns with VIF > 5

We will ignore the VIF values for dummy variables and the constant (intercept)

Removing Multicollinearity To remove multicollinearity

Drop every column one by one that has a VIF score greater than 5.

Look at the adjusted R-squared and RMSE of all these models.

Drop the variable that makes the least change in adjusted R-squared.

Check the VIF scores again.

Continue till you get all VIF scores under 5.

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

In [ ]:

```
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)
```

```python
    return temp
```

In [ ]:

```python
col_list = ['screen_size','weight'] ## Complete the code to specify the columns with hig

res = treating_multicollinearity(x_train, y_train, col_list) ## Complete the code to che
res
```

Out[ ]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	screen_size	0.838381	0.234703
1	weight	0.838071	0.234928

In [ ]:

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

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

```
VIF after dropping  screen_size
```

Out[ ]:

	feature	VIF
0	const	202.673906
1	main_camera_mp	2.281835
2	selfie_camera_mp	2.809009
3	int_memory	1.362043
4	ram	2.282350
5	battery	3.842989
6	weight	2.993855
7	days_used	2.648929
8	normalized_new_price	3.077650
9	years_since_release	4.730315
10	brand_name_Alcatel	3.405533
11	brand_name_Apple	13.000338
12	brand_name_Asus	3.326698
13	brand_name_BlackBerry	1.631042
14	brand_name_Celkon	1.774528
15	brand_name_Coolpad	1.467719
16	brand_name_Gionee	1.941437

	feature	VIF
17	brand_name_Google	1.319334
18	brand_name_HTC	3.399980
19	brand_name_Honor	3.340354
20	brand_name_Huawei	5.981046
21	brand_name_Infinix	1.283526
22	brand_name_Karbonn	1.573494
23	brand_name_LG	4.832548
24	brand_name_Lava	1.711092
25	brand_name_Lenovo	4.553789
26	brand_name_Meizu	2.176424
27	brand_name_Micromax	3.358629
28	brand_name_Microsoft	1.868243
29	brand_name_Motorola	3.262356
30	brand_name_Nokia	3.464643
31	brand_name_OnePlus	1.437004
32	brand_name_Oppo	3.965445
33	brand_name_Others	9.652572
34	brand_name_Panasonic	2.104853
35	brand_name_Realme	1.943845
36	brand_name_Samsung	7.523421
37	brand_name_Sony	2.937375
38	brand_name_Spice	1.683302
39	brand_name_Vivo	3.650625
40	brand_name_XOLO	2.137844
41	brand_name_Xiaomi	3.713988
42	brand_name_ZTE	3.788971
43	os_Others	1.625212
44	os_Windows	1.595936
45	os_iOS	11.678957
46	4g_yes	2.466915
47	5g_yes	1.810289

Dropping high p-value variables (if needed) We will drop the predictor variables having a p-value greater than 0.05 as they do not significantly impact the target variable.

But sometimes p-values change after dropping a variable. So, we'll not drop all variables at once.

Instead, we will do the following:

Build a model, check the p-values of the variables, and drop the column with the highest p-value.

Create a new model without the dropped feature, check the p-values of the variables, and drop the column with the highest p-value.

Repeat the above two steps till there are no columns with p-value > 0.05.

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

In [ ]:

```
# initial list of columns
predictors = x_train2.copy()  ## Complete the code to check for p-values on the right da
cols = predictors.columns.tolist()

# setting an initial max p-value
max_p_value = 1

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

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

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

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

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

selected_features = cols
print(selected_features)
```

```
['const', 'main_camera_mp', 'selfie_camera_mp', 'ram', 'weight', 'normalized_new_price',
'years_since_release', 'brand_name_Karbonn', 'brand_name_Samsung', 'brand_name_Sony', 'b
rand_name_Xiaomi', 'os_Others', 'os_iOS', '4g_yes', '5g_yes']
```

In [ ]:

```
x_train3 = x_train2[selected_features]
x_test3 = x_test2[selected_features]
olsmod2 = sm.OLS(y_train, x_train3).fit()
print(olsmod2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.839
Model:                               OLS   Adj. R-squared:                  0.838
```

```
Method:                 Least Squares   F-statistic:                     895.7
Date:                Sun, 13 Apr 2025   Prob (F-statistic):               0.00
Time:                        15:17:58   Log-Likelihood:                 80.645
No. Observations:                2417   AIC:                            -131.3
Df Residuals:                    2402   BIC:                            -44.44
Df Model:                          14
Covariance Type:            nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                      1.5000      0.048     30.955      0.000       1.405       1.595
main_camera_mp             0.0210      0.001     14.714      0.000       0.018       0.024
selfie_camera_mp           0.0138      0.001     12.858      0.000       0.012       0.016
ram                        0.0207      0.005      4.151      0.000       0.011       0.030
weight                     0.0017      6e-05     27.672      0.000       0.002       0.002
normalized_new_price       0.4415      0.011     39.337      0.000       0.419       0.463
years_since_release       -0.0292      0.003     -8.589      0.000      -0.036      -0.023
brand_name_Karbonn         0.1156      0.055      2.111      0.035       0.008       0.223
brand_name_Samsung        -0.0374      0.016     -2.270      0.023      -0.070      -0.005
brand_name_Sony           -0.0670      0.030     -2.197      0.028      -0.127      -0.007
brand_name_Xiaomi          0.0801      0.026      3.114      0.002       0.030       0.130
os_Others                 -0.1276      0.027     -4.667      0.000      -0.181      -0.074
os_iOS                    -0.0900      0.045     -1.994      0.046      -0.179      -0.002
4g_yes                     0.0502      0.015      3.326      0.001       0.021       0.080
5g_yes                    -0.0673      0.031     -2.194      0.028      -0.127      -0.007
==============================================================================
Omnibus:                      246.183   Durbin-Watson:                   1.902
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              483.879
Skew:                          -0.658   Prob(JB):                    8.45e-106
Kurtosis:                       4.753   Cond. No.                     2.39e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 2.39e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [ ]:

```python
x_train3 = x_train2[selected_features]  ## Complete the code to specify the train data f
x_test3 = x_test2[selected_features]  ## Complete the code to specify the test data from
```

In [ ]:

```python
olsmodel2 = sm.OLS(y_train, x_train3).fit()## Complete the code fit OLS() on updated dat
print(olsmodel2.summary())
```

```
                            OLS Regression Results
=================================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.839
Model:                               OLS   Adj. R-squared:                  0.838
Method:                    Least Squares   F-statistic:                     895.7
Date:                   Sun, 13 Apr 2025   Prob (F-statistic):               0.00
Time:                           15:17:58   Log-Likelihood:                 80.645
No. Observations:                   2417   AIC:                            -131.3
Df Residuals:                       2402   BIC:                            -44.44
Df Model:                             14
Covariance Type:               nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
```

```
------------------------------------------------------------------------------------------
const                     1.5000      0.048     30.955      0.000       1.405       1.595
main_camera_mp            0.0210      0.001     14.714      0.000       0.018       0.024
selfie_camera_mp          0.0138      0.001     12.858      0.000       0.012       0.016
ram                       0.0207      0.005      4.151      0.000       0.011       0.030
weight                    0.0017      6e-05     27.672      0.000       0.002       0.002
normalized_new_price      0.4415      0.011     39.337      0.000       0.419       0.463
years_since_release      -0.0292      0.003     -8.589      0.000      -0.036      -0.023
brand_name_Karbonn        0.1156      0.055      2.111      0.035       0.008       0.223
brand_name_Samsung       -0.0374      0.016     -2.270      0.023      -0.070      -0.005
brand_name_Sony          -0.0670      0.030     -2.197      0.028      -0.127      -0.007
brand_name_Xiaomi         0.0801      0.026      3.114      0.002       0.030       0.130
os_Others                -0.1276      0.027     -4.667      0.000      -0.181      -0.074
os_iOS                   -0.0900      0.045     -1.994      0.046      -0.179      -0.002
4g_yes                    0.0502      0.015      3.326      0.001       0.021       0.080
5g_yes                   -0.0673      0.031     -2.194      0.028      -0.127      -0.007
==============================================================================
Omnibus:                      246.183   Durbin-Watson:                   1.902
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              483.879
Skew:                          -0.658   Prob(JB):                    8.45e-106
Kurtosis:                       4.753   Cond. No.                     2.39e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 2.39e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [ ]:

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

Training Performance

Out[ ]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.23403	0.182751	0.83924	0.838235	4.395407

In [ ]:

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

Test Performance

Out[ ]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.241434	0.186649	0.838387	0.836013	4.556349

Now no feature has p-value greater than 0.05, so we'll consider the features in x_train3 as the final set of predictor variables and olsmod2 as the final model to move forward with Now adjusted R-squared is 0.836, i.e., our model is able to explain ~83% of the variance The adjusted R-squared in olsmodel (where we considered the variables without multicollinearity) was 0.84 This shows that the variables we dropped were not affecting the model much RMSE and MAE values are comparable for train and test sets, indicating that the model is not overfitting

Now we'll check the rest of the assumptions on olsmodel2.

Linearity of variables

Independence of error terms

Normality of error terms

No Heteroscedasticity

TEST FOR LINEARITY AND INDEPENDENCE

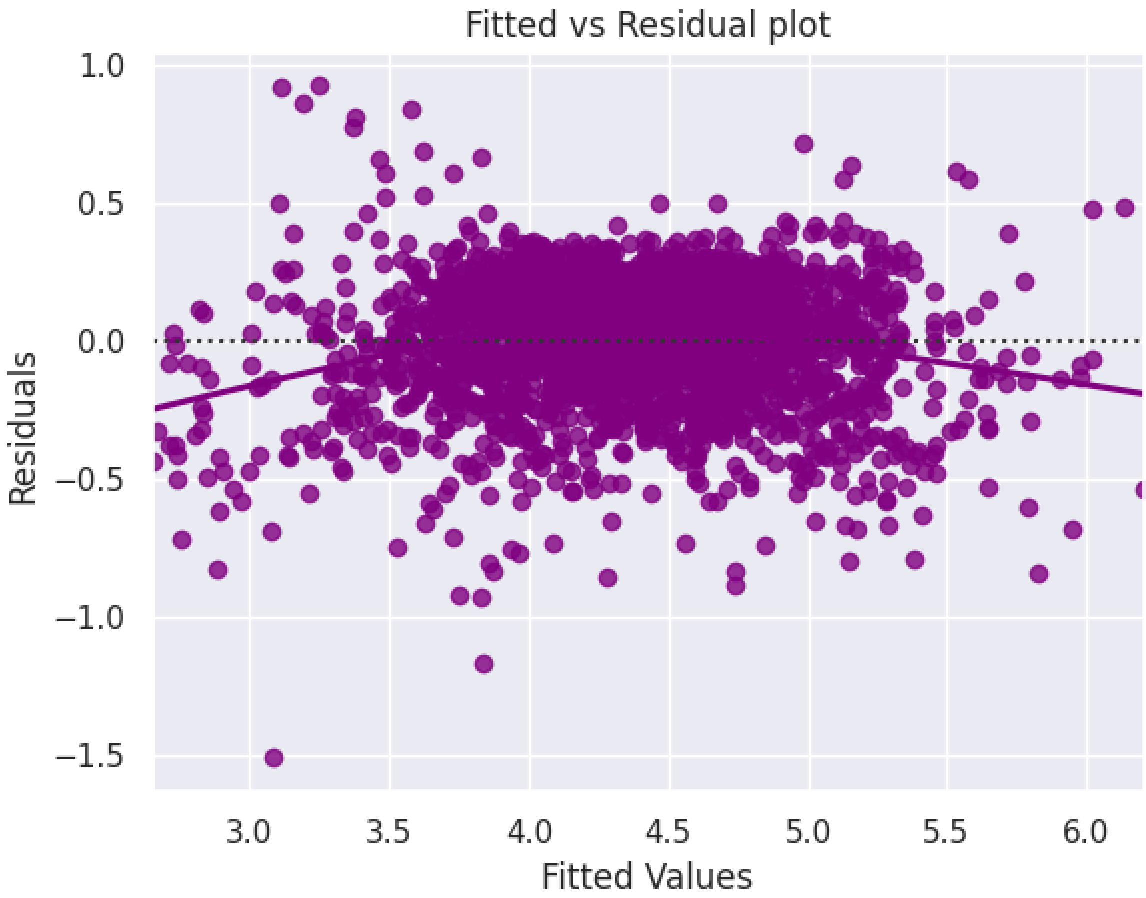
We will test for linearity and independence by making a plot of fitted values vs residuals and checking for patterns.

If there is no pattern, then we say the model is linear and residuals are independent.

Otherwise, the model is showing signs of non-linearity and residuals are not independent.

In [ ]:

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

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

df_pred.head()
# 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()
```



There is no pattern in the above graph , hence we say the model is linear and residuals are independent.

TEST FOR NORMALITY

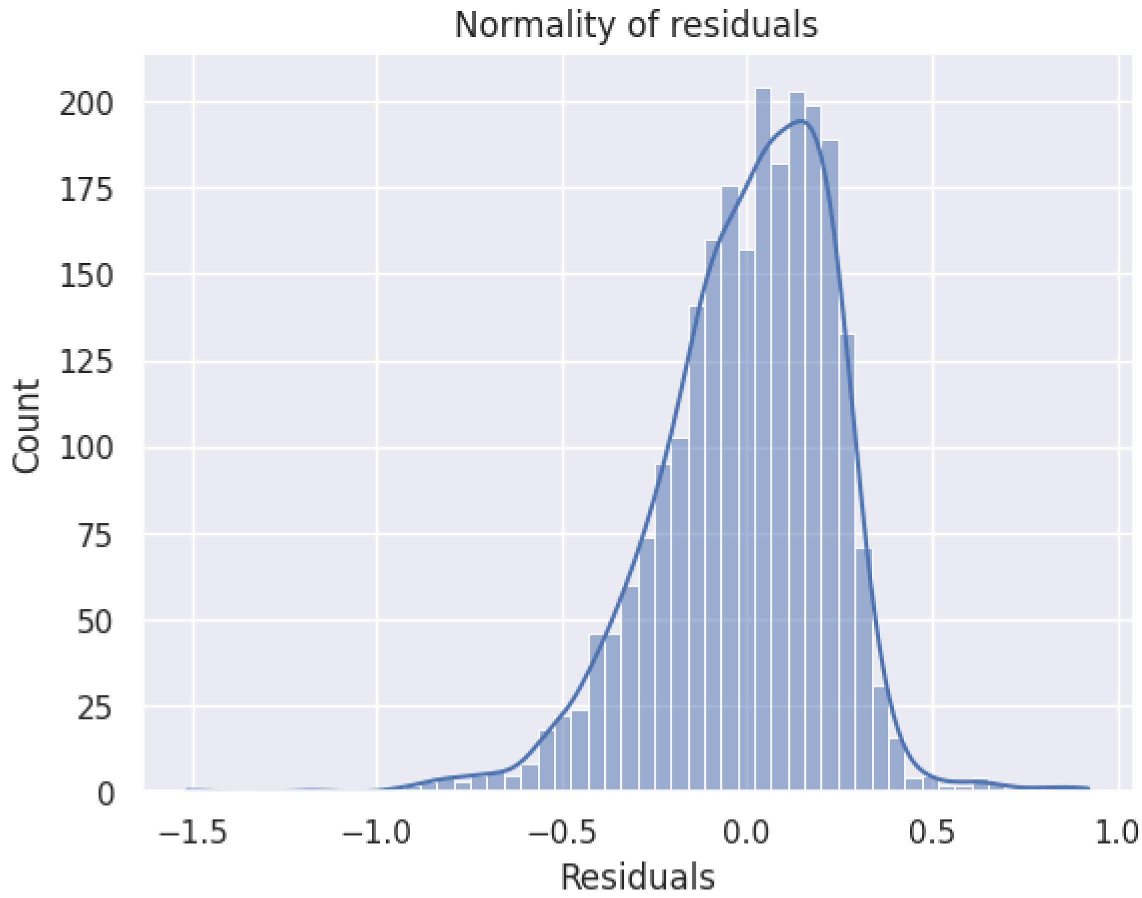
We will test for normality by checking the distribution of residuals, by checking the Q-Q plot of residuals, and by using the Shapiro-Wilk test.

If the residuals follow a normal distribution, they will make a straight line plot, otherwise not.

If the p-value of the Shapiro-Wilk test is greater than 0.05, we can say the residuals are normally distributed.

In [ ]:

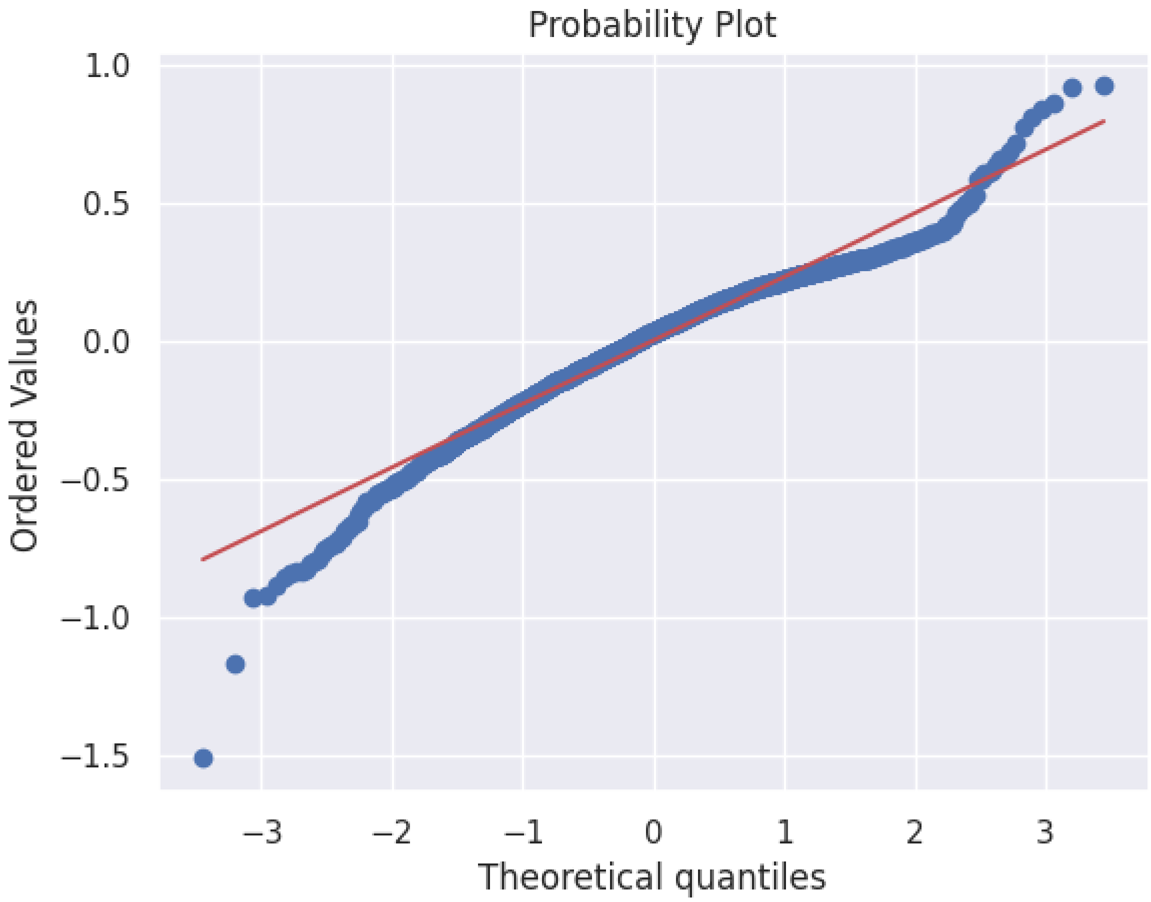
```
sns.histplot(data=df_pred, x="Residuals", kde=True)## Complete the code to plot the dist
plt.title("Normality of residuals")
plt.show()
import pylab
import scipy.stats as stats
```



The histogram does have a bell shape.now lets check Q-Q plot

In [ ]:

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



This forms more or less a straight line.Lets examine the shapiro wilk's test

In [ ]:

```
stats.shapiro(df_pred["Residuals"]) ## Complete the code to apply the Shapiro-Wilks test
```

Out[ ]:

```
ShapiroResult(statistic=np.float64(0.967695082990057), pvalue=np.float64(6.983856712612358e-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

We will test for homoscedasticity by using the goldfeldquandt test.

If we get a p-value greater than 0.05, we can say that the residuals are homoscedastic. Otherwise, they are heteroscedastic.

In [ ]:

```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

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

Out[ ]:

```
[('F statistic', np.float64(1.0087504199106758)),
 ('p-value', np.float64(0.4401970650667301))]
```

Since p value is greater than 0.05 ,the assumption is satisfied

Final Model

In [ ]:

```
x_train_final = x_train3.copy()
x_test_final = x_test3.copy()
olsmodel_final = sm.OLS(y_train, x_train_final).fit()
print(olsmodel_final.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.839
Model:                               OLS   Adj. R-squared:                  0.838
Method:                    Least Squares   F-statistic:                     895.7
Date:                   Sun, 13 Apr 2025   Prob (F-statistic):               0.00
Time:                           15:18:24   Log-Likelihood:                 80.645
No. Observations:                   2417   AIC:                            -131.3
Df Residuals:                       2402   BIC:                            -44.44
Df Model:                             14
Covariance Type:               nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                     1.5000      0.048     30.955      0.000       1.405       1.595
main_camera_mp            0.0210      0.001     14.714      0.000       0.018       0.024
selfie_camera_mp          0.0138      0.001     12.858      0.000       0.012       0.016
ram                       0.0207      0.005      4.151      0.000       0.011       0.030
weight                    0.0017      6e-05     27.672      0.000       0.002       0.002
normalized_new_price      0.4415      0.011     39.337      0.000       0.419       0.463
years_since_release      -0.0292      0.003     -8.589      0.000      -0.036      -0.023
brand_name_Karbonn        0.1156      0.055      2.111      0.035       0.008       0.223
brand_name_Samsung       -0.0374      0.016     -2.270      0.023      -0.070      -0.005
brand_name_Sony          -0.0670      0.030     -2.197      0.028      -0.127      -0.007
brand_name_Xiaomi         0.0801      0.026      3.114      0.002       0.030       0.130
os_Others                -0.1276      0.027     -4.667      0.000      -0.181      -0.074
os_iOS                   -0.0900      0.045     -1.994      0.046      -0.179      -0.002
4g_yes                    0.0502      0.015      3.326      0.001       0.021       0.080
5g_yes                   -0.0673      0.031     -2.194      0.028      -0.127      -0.007
==============================================================================
Omnibus:                      246.183   Durbin-Watson:                   1.902
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              483.879
Skew:                          -0.658   Prob(JB):                    8.45e-106
Kurtosis:                       4.753   Cond. No.                     2.39e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 2.39e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [ ]:

```
# 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[ ]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.23403	0.182751	0.83924	0.838235	4.395407

In [ ]:

```
# 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[ ]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.241434	0.186649	0.838387	0.836013	4.556349

The model is able to explain ~83% of the variation in the data

The train and test RMSE and MAE are low and comparable. So, our model is not suffering from overfitting

The MAPE on the test set suggests we can predict within 4.3% of the normalized_used_price

Hence, we can conclude the model olsmodel_final is good for prediction as well as inference purposes

Actionable Insights and Recommendations

Conclusions:

The model is able to explain ~83% of the variation in the data and within 4.5% of the normalized_used_price on the test data, which is good

This indicates that the model is good for prediction as well as inference purposes

Devices that were more expensive when released tend to retain a higher proportion of their value when resold.

Each additional year since release of the product reduces the normalized used price by ~2.9%.

Higher camera megapixels (main_camera_mp and selfie_camera_mp) are positively associated with resale value of the phone.

More RAM also increases the resale price.

Heavier phones have slightly higher resale price

Karbonn (+0.1156) and Xiaomi (+0.0801) have good resale price

Samsung (−0.0374) and Sony (−0.0670) have less resale price

Devices with 4G have ~5% higher resale value.

5G support lowers normalized used phone prices

-

Recommendations:

Since,Devices that were more expensive when released tend to retain a higher proportion of their value when resold.Recell can focus more on higher rate end phones and tablets for the market

Each additional year since release of the product reduces the normalized used price by ~2.9%. It is better to take the devices into the market which has released within and year so their prices would stay up

Higher camera megapixels (main_camera_mp and selfie_camera_mp) are positively associated with resale value of the phone.So even on the used devices ,People need better camera quality .Recell needs to ensure this pointer when taking a device into the market.

More RAM also increases the resale price.People also prefer better storage even on used devices.Recell needs to ensure this pointer when taking a device into the market.

Heavier phones have slightly higher resale price.This may be due to large battery or people like to have moderate weight for the devices to avoid breakage or damages .Recell needs to ensure this pointer when taking a device into the market.

Karbonn (+0.1156) and Xiaomi (+0.0801) have good resale price.Recell should have these brands as most preffered brands when taking a device into the market.

Samsung (−0.0374) and Sony (−0.0670) have less resale price.Recell should mostly try to avoid these brands when taking a device into the market.

Devices with 4G have ~5% higher resale value.5G support lowers normalized used phone prices.This may be due to the vast usage of 4g among the people.Recell needs to ensure this pointer when taking a device into the market.