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 [198…

```
# this will help in making the Python code more structured automatically (good coding
%load_ext nb_black

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

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

sns.set()

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

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

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

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

```
The nb_black extension is already loaded. To reload it, use:
  %reload_ext nb_black
```

Loading the dataset

In [199…

```
data = pd.read_csv("used_device_data.csv")
data.head()
```

Out[199]:

	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

Data Overview

- Observations
- Sanity checks

Check the shape of the dataset

In [200…

```
data.shape
```

Out[200]:

```
(3454, 15)
```

The data has 3454 rows and 15 columns

Get the info regarding column datatypes

In [201…

```
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
```

In [202…
```
# checking for missing values
data.isnull().sum()
```

Out[202]:
```
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 [203…
```
data.duplicated().sum()
```

Out[203]:
```
0
```

Columns brand_name, os, 4g and 5g are categorical while all others are numerical data types

Column normalized_used_price is the dependent variable

All rows have data for most of the columns

Some missing data in columns main_camera_mp, selfie_camera_mp, int_memory,ram,battery and weight

There are no duplicates in the data

Get summary statistics for the numerical columns

In [204…
```
data.describe(include="all").T
```

Out[204]:

	count	unique	top	freq	mean	std	min	25%
brand_name	3454	34	Others	502	NaN	NaN	NaN	NaN
os	3454	4	Android	3214	NaN	NaN	NaN	NaN
screen_size	3454.0	NaN	NaN	NaN	13.713115	3.80528	5.08	12.7
4g	3454	2	yes	2335	NaN	NaN	NaN	NaN
5g	3454	2	no	3302	NaN	NaN	NaN	NaN
main_camera_mp	3275.0	NaN	NaN	NaN	9.460208	4.815461	0.08	5.0
selfie_camera_mp	3452.0	NaN	NaN	NaN	6.554229	6.970372	0.0	2.0
int_memory	3450.0	NaN	NaN	NaN	54.573099	84.972371	0.01	16.0
ram	3450.0	NaN	NaN	NaN	4.036122	1.365105	0.02	4.0
battery	3448.0	NaN	NaN	NaN	3133.402697	1299.682844	500.0	2100.0
weight	3447.0	NaN	NaN	NaN	182.751871	88.413228	69.0	142.0
release_year	3454.0	NaN	NaN	NaN	2015.965258	2.298455	2013.0	2014.0
days_used	3454.0	NaN	NaN	NaN	674.869716	248.580166	91.0	533.5
normalized_used_price	3454.0	NaN	NaN	NaN	4.364712	0.588914	1.536867	4.033931
normalized_new_price	3454.0	NaN	NaN	NaN	5.233107	0.683637	2.901422	4.790342

There are 34 different brands, 4 different os out of which 'Android' is the most popular.

Most devices are 4g enabled and lack 5g capabilities.

This data is for phones released between 2013 and 2022.

We can see the range of all the numerical columns in the data. Looking at the mean, median, min and max of numerical data, there appear to be outliers in the data.

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 [205…

```
# creating a copy of the data to avoid any changes to original data
df = data.copy()
```

In [206…

```
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 co
    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 [207…

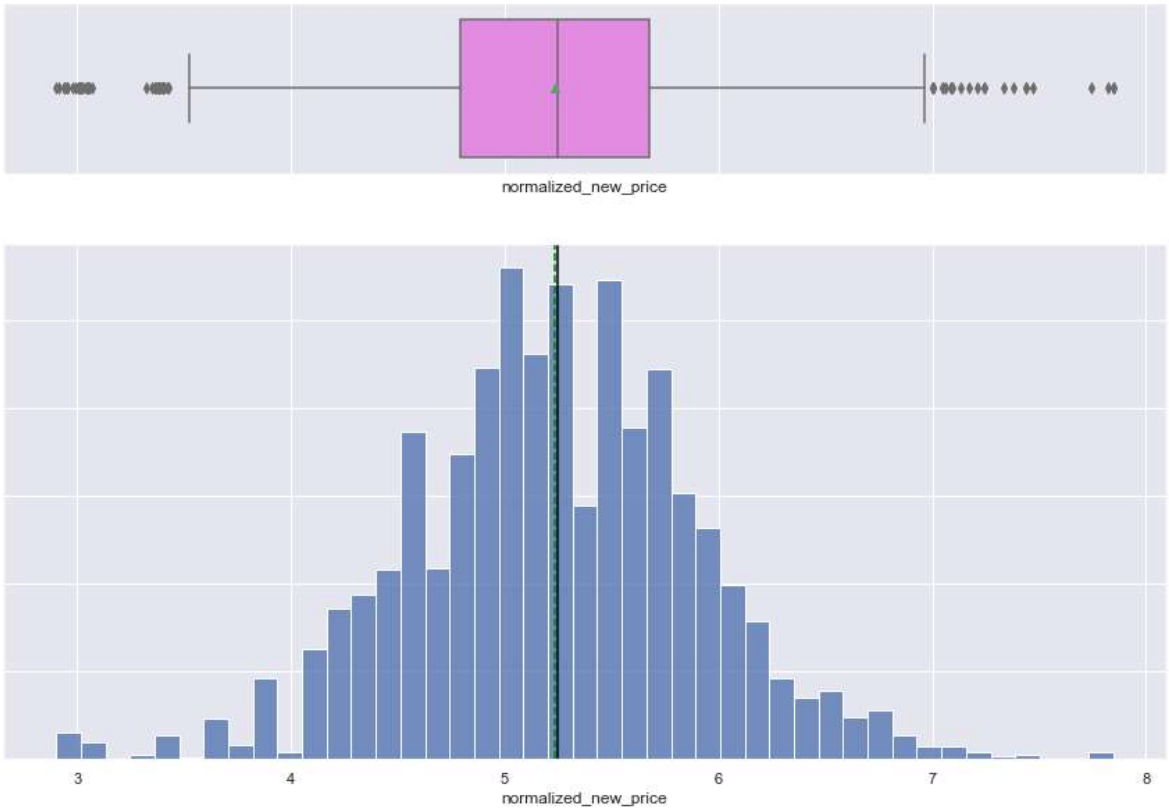
```
# function to create labeled barplots


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,
        palette="Paired",
        order=data[feature].value_counts().index[:n],
    )

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

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

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

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

Univarate Analysis

In [208…

```
histogram_boxplot(df, "normalized_new_price")
```

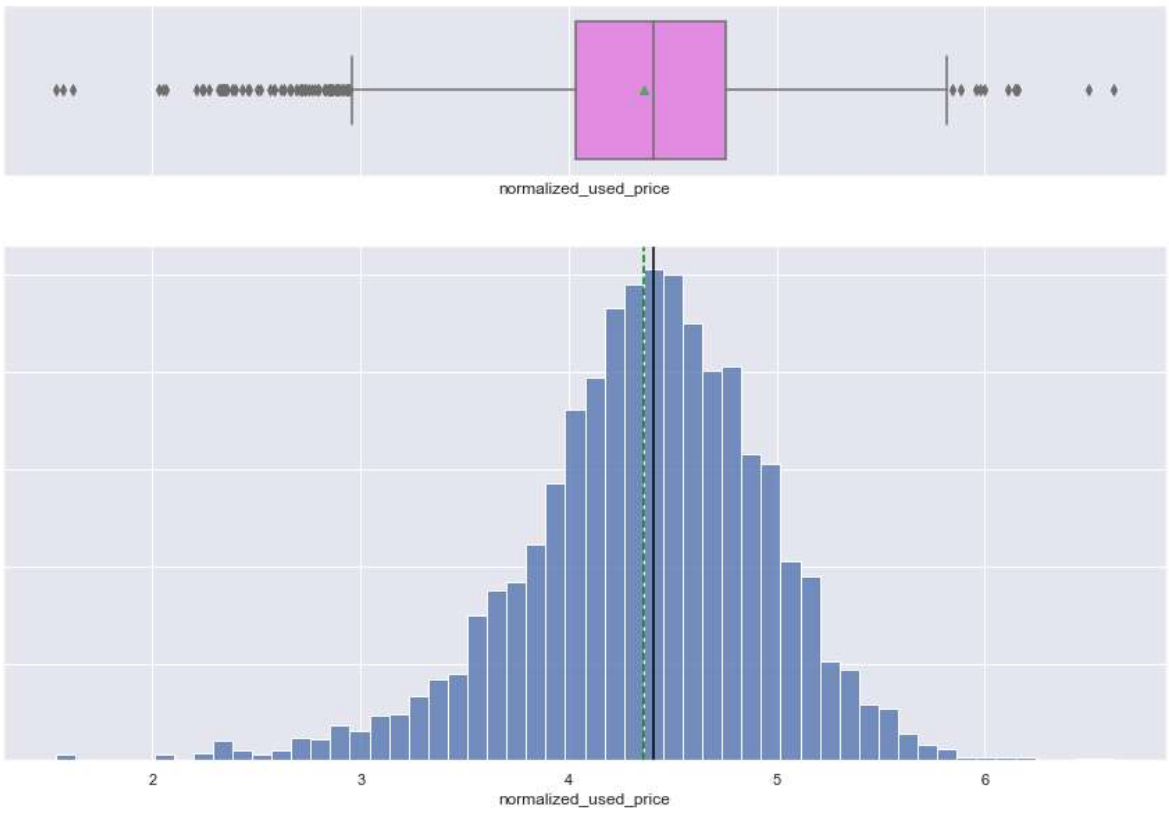


There are outliers on both sides of the boxplot

The distribution of the new price appears to be normal.

In [209…

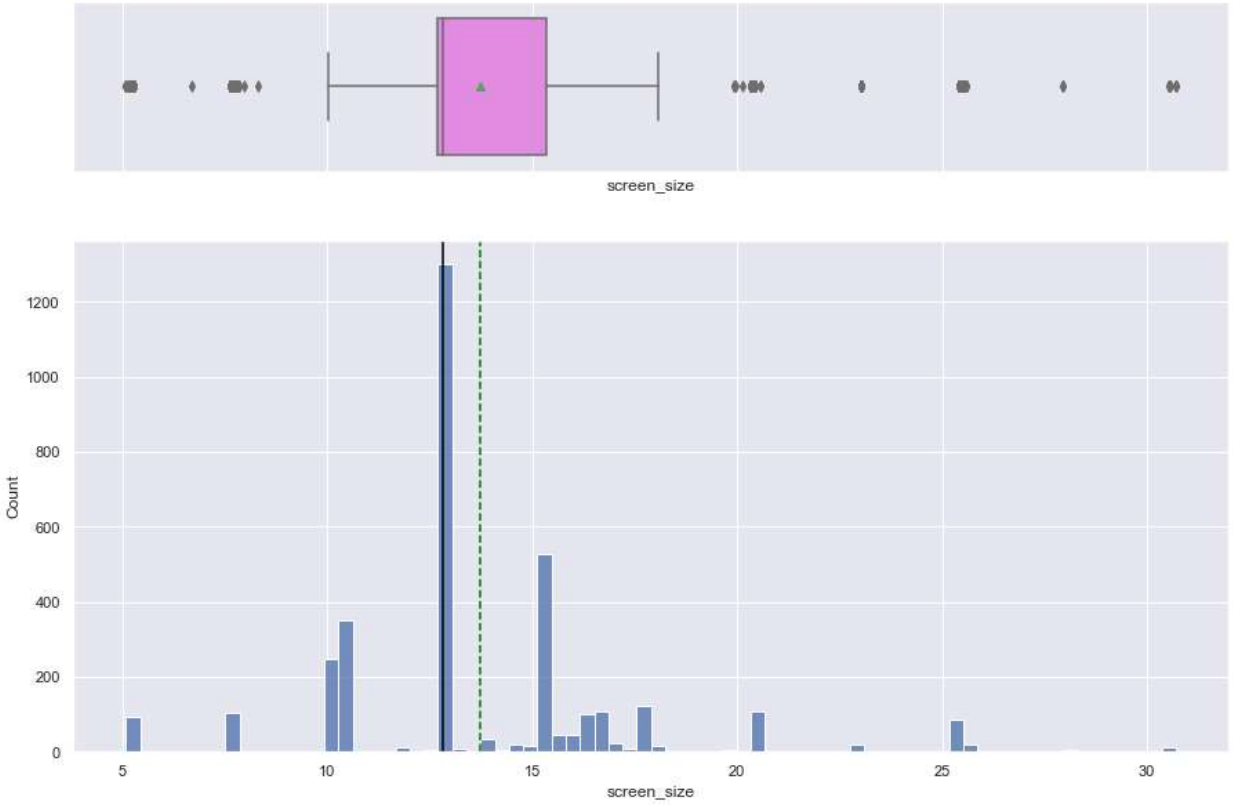
```
histogram_boxplot(df, "normalized_used_price")
```



There are outliers on both sides of the data with significant number on left than right.

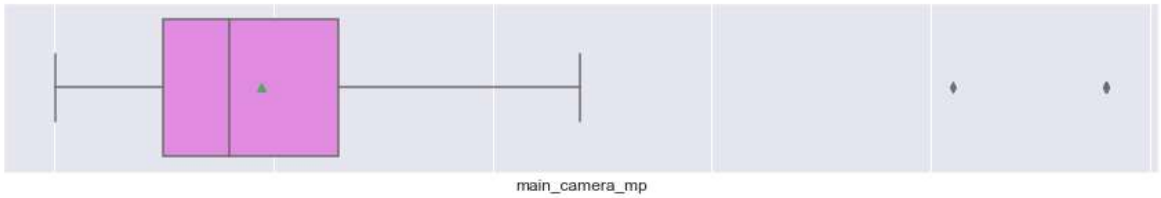
The data appears to be left skewed.

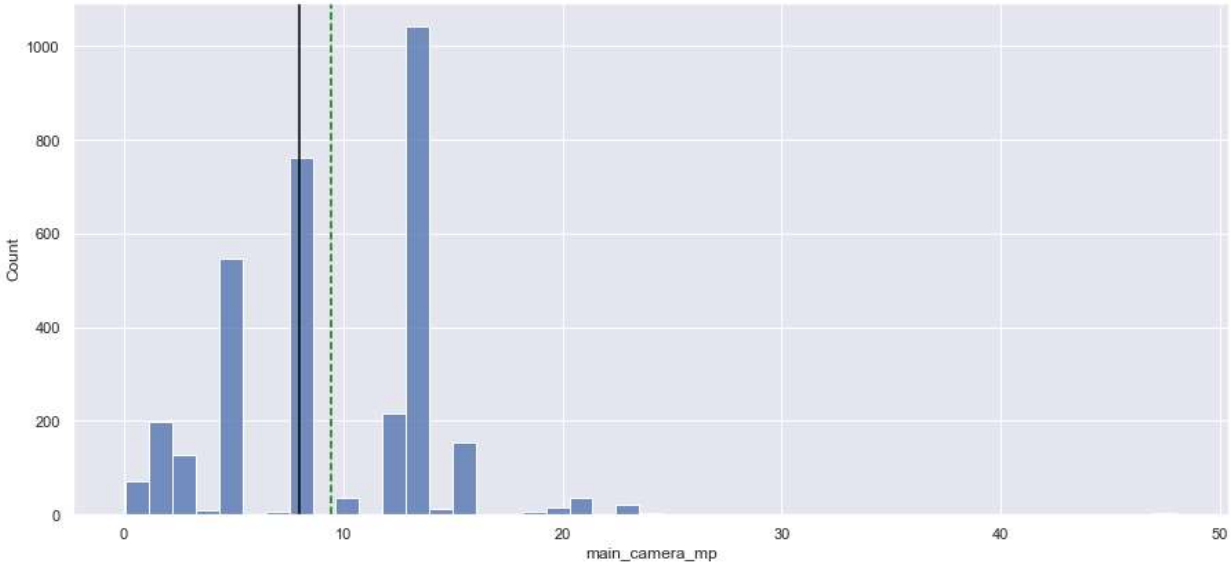
In [210…
```
histogram_boxplot(df, "screen_size")
```



This distribution appears to be right skewed with outliers on both sides of the boxplot.

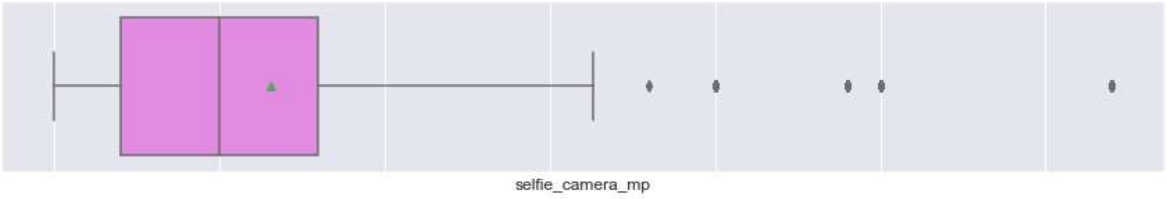
In [211…
```
histogram_boxplot(df, "main_camera_mp")
```

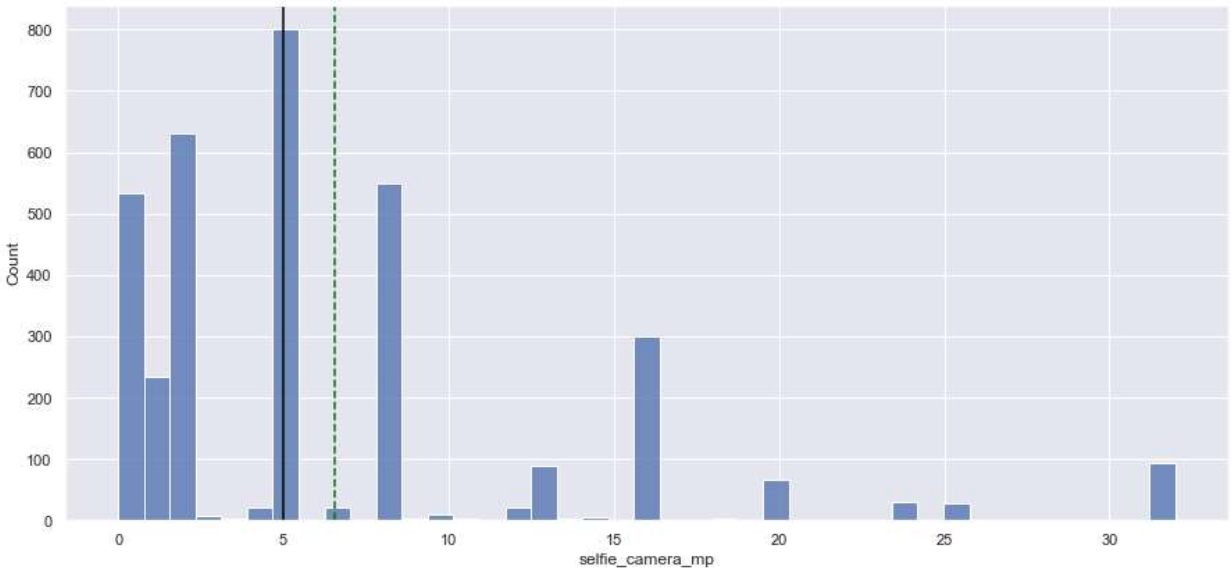




There are some outliers on right of the boxplot. Data appears to be right skewed.

In [212…
```
histogram_boxplot(df, "selfie_camera_mp")
```

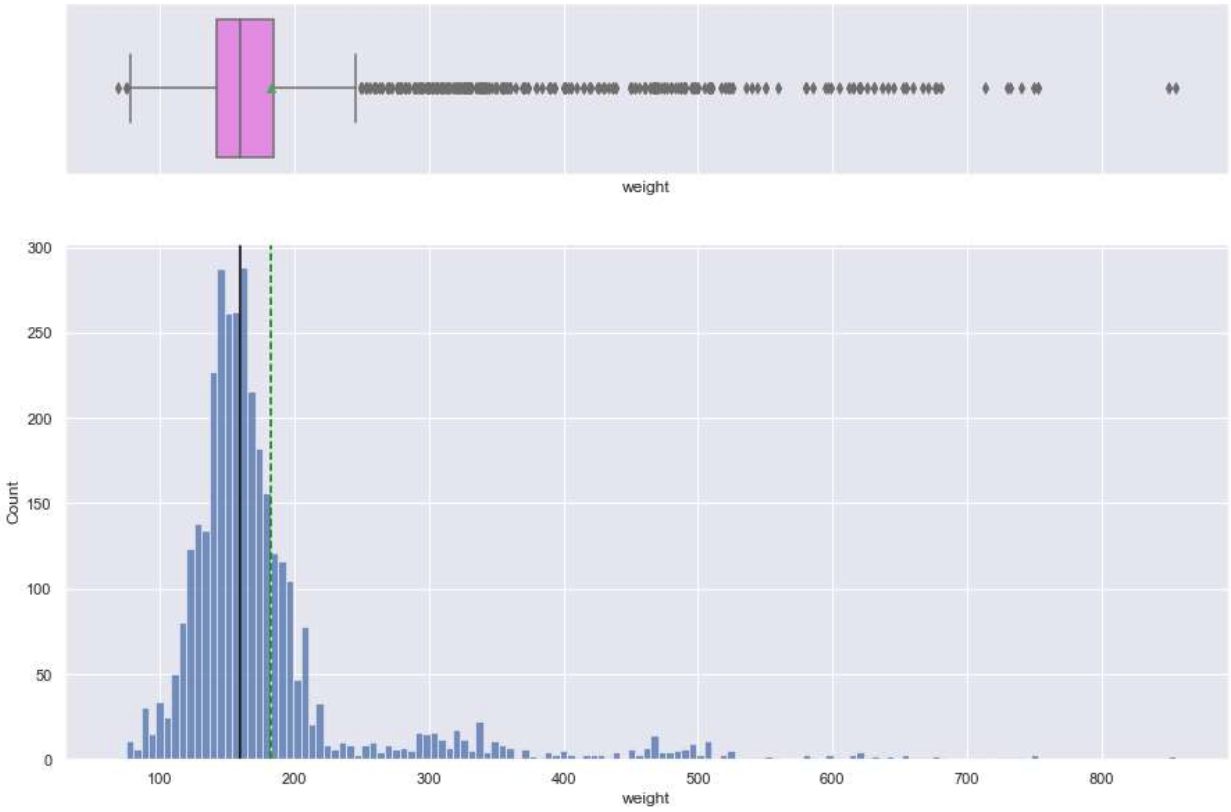




There are some outliers on right of the boxplot. Data appears to be right skewed.

In [213…

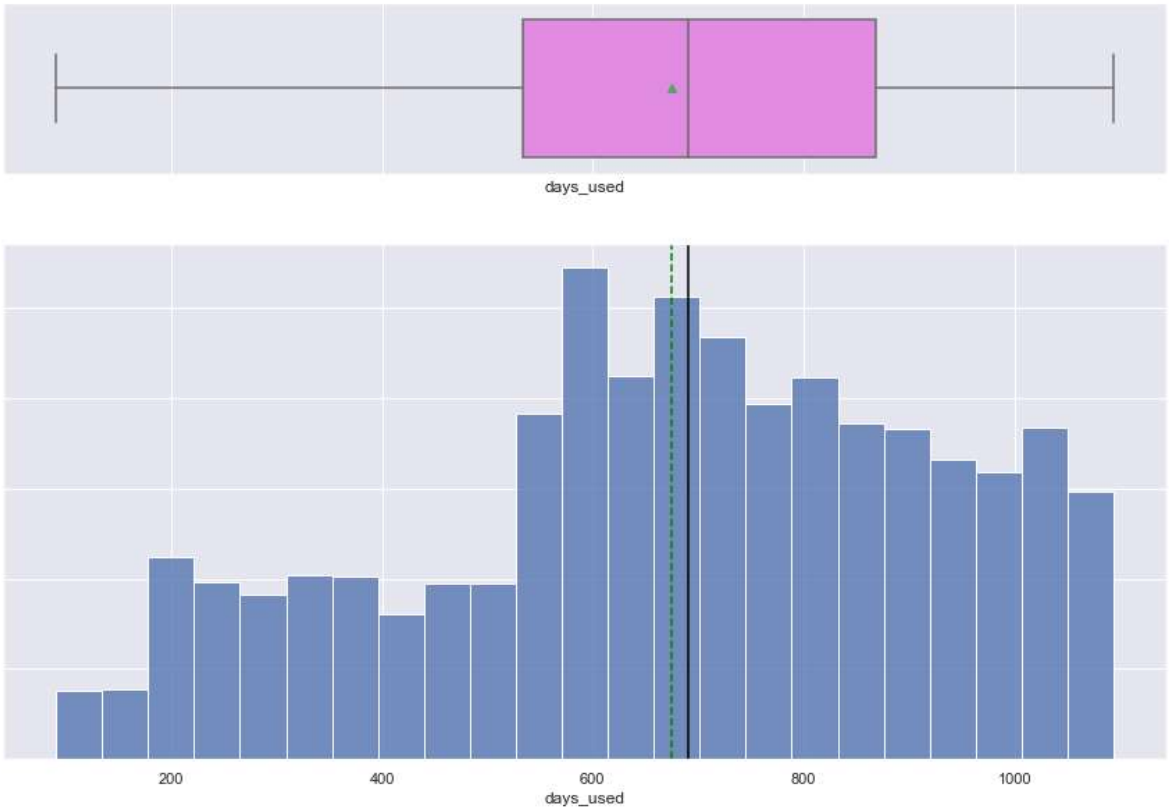
```
histogram_boxplot(df, "weight")
```



There are outliers on both sides of the boxplot, significantly on the right. Data appears to be right skewed.

In [214…

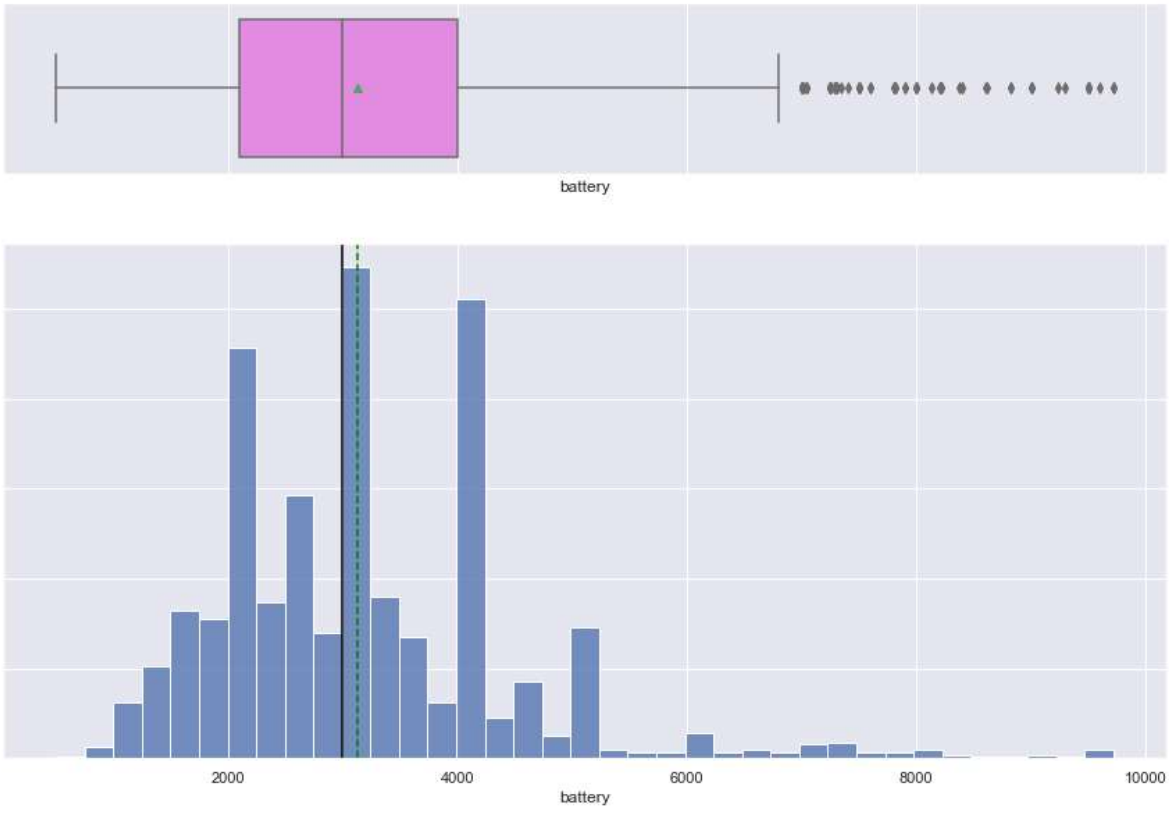
```
histogram_boxplot(df, "days_used")
```



There appear to be no outliers in the data and it appears to be left skewed.

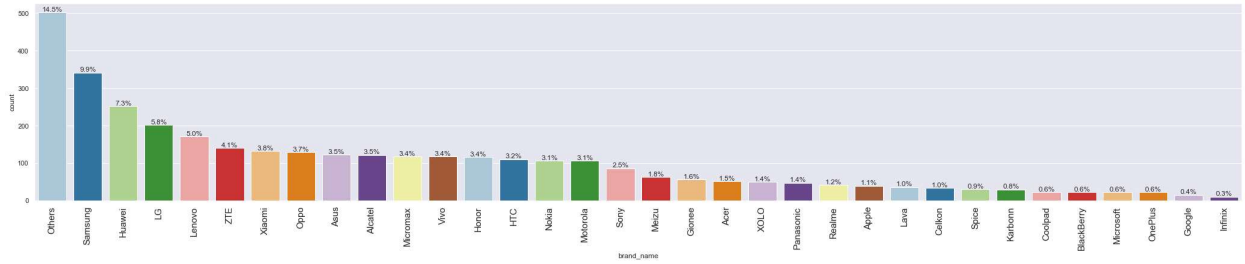
In [215…

```
histogram_boxplot(df, "battery")
```



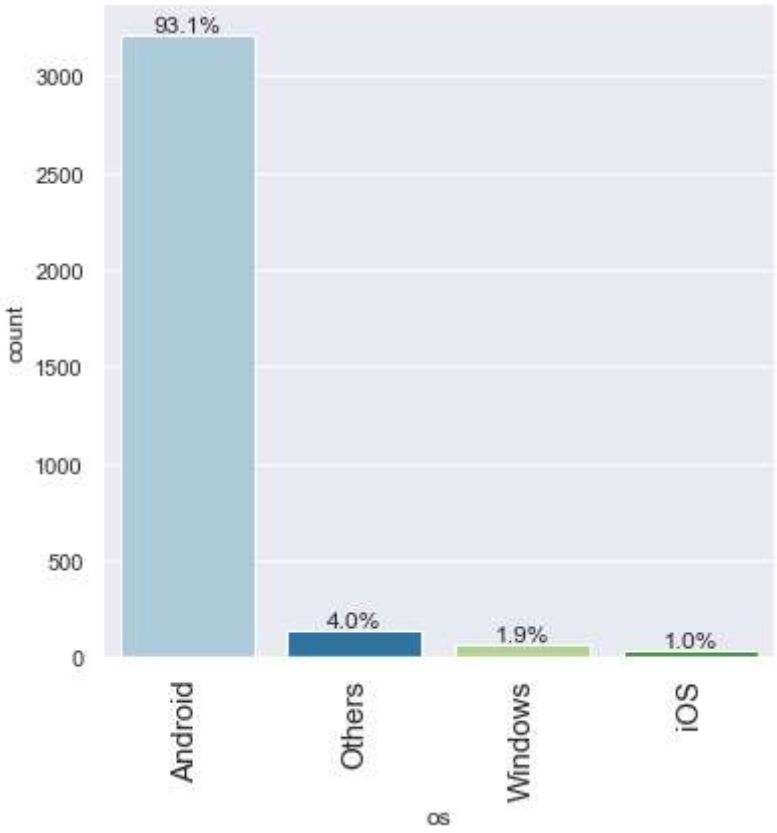
There are a significant amount of outliers on the right and data appears to be right skewed.

In [216…
```
labeled_barplot(df, "brand_name", perc=True)
```



Others(non-famous brands) own a major share of the data. Samsung is the known brand with largest representation. Few very well known brands like Apple, Google, Microsoft etc do not have a signifcant representation.

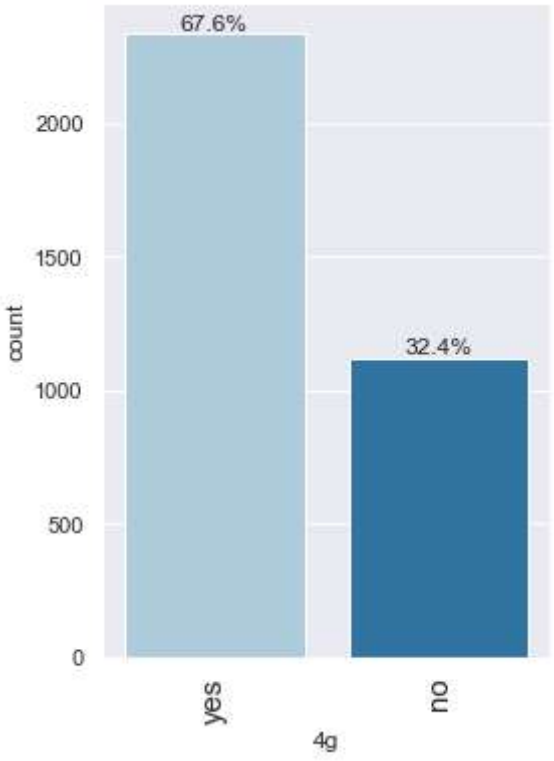
In [217…
```
labeled_barplot(df, "os", perc=True)
```



Android phones have a significantly high number as compared to all other operating systems.

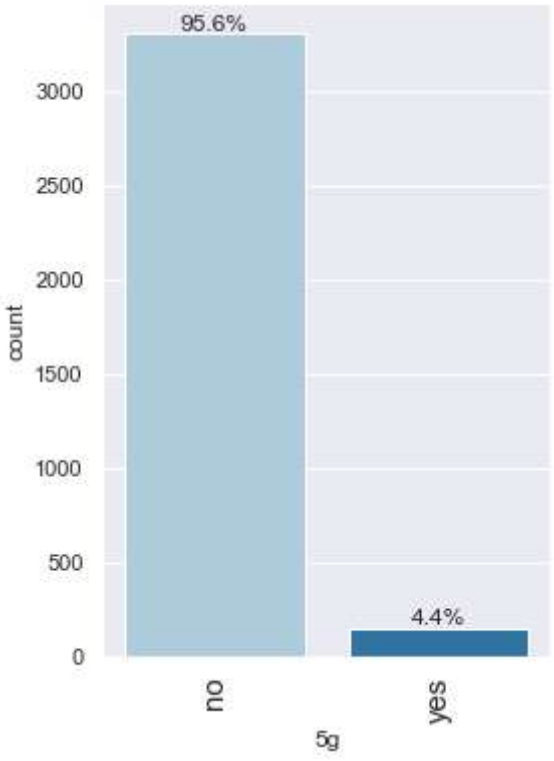
Androids dominate 93.1% over all other os in the data.

In [218…
```
labeled_barplot(df, "4g", perc=True)
```



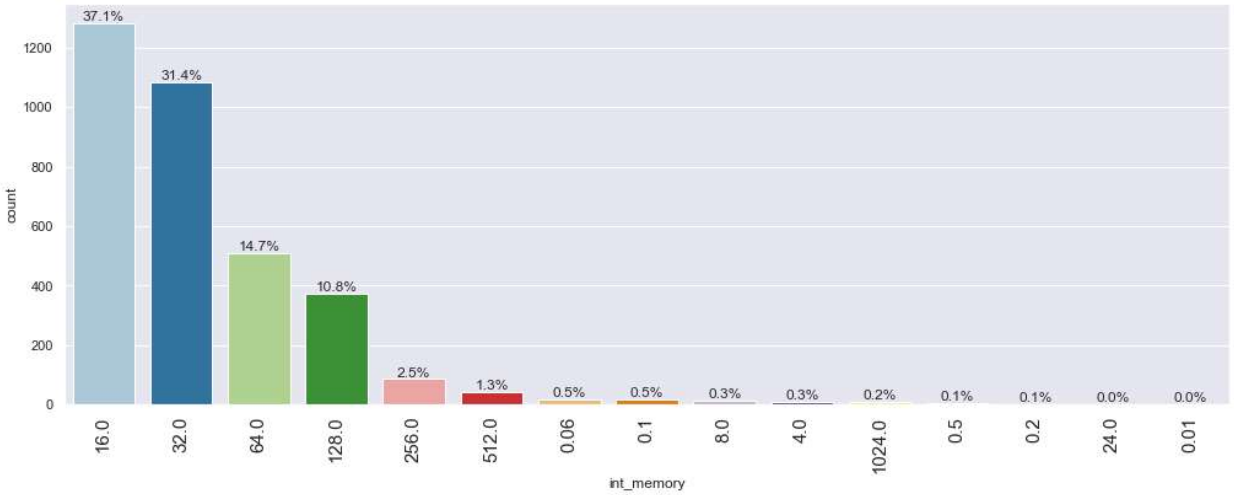
A significantly high number of phones support 4g

In [219…
```
labeled_barplot(df, "5g", perc=True)
```



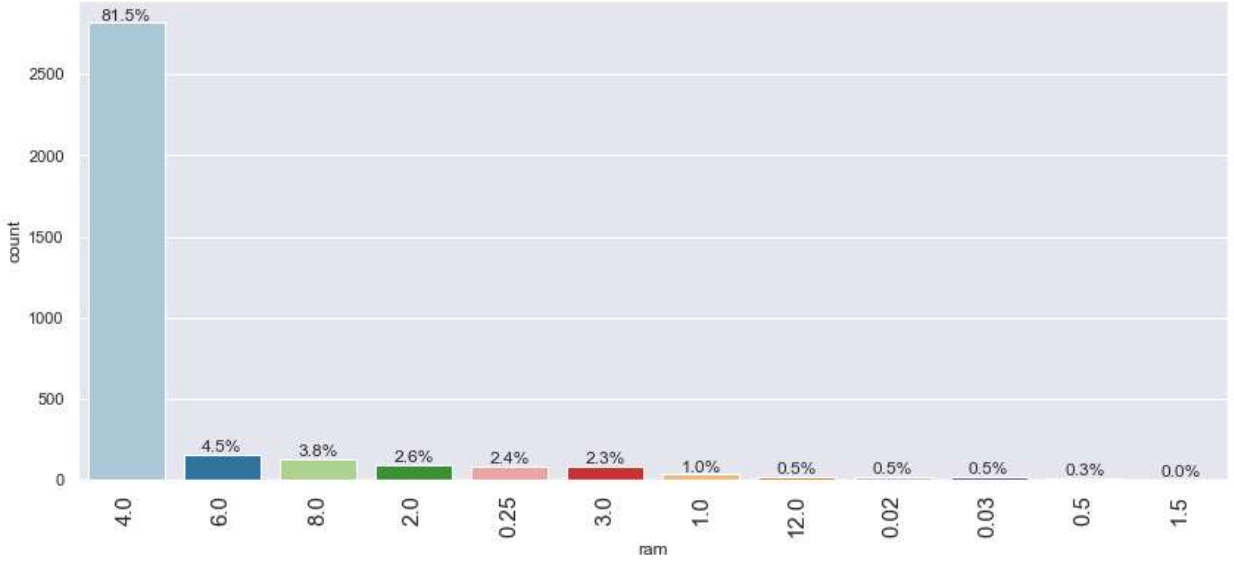
A significantly high number of phones do not support 5g

In [220…
```
labeled_barplot(df, "int_memory", perc=True)
```



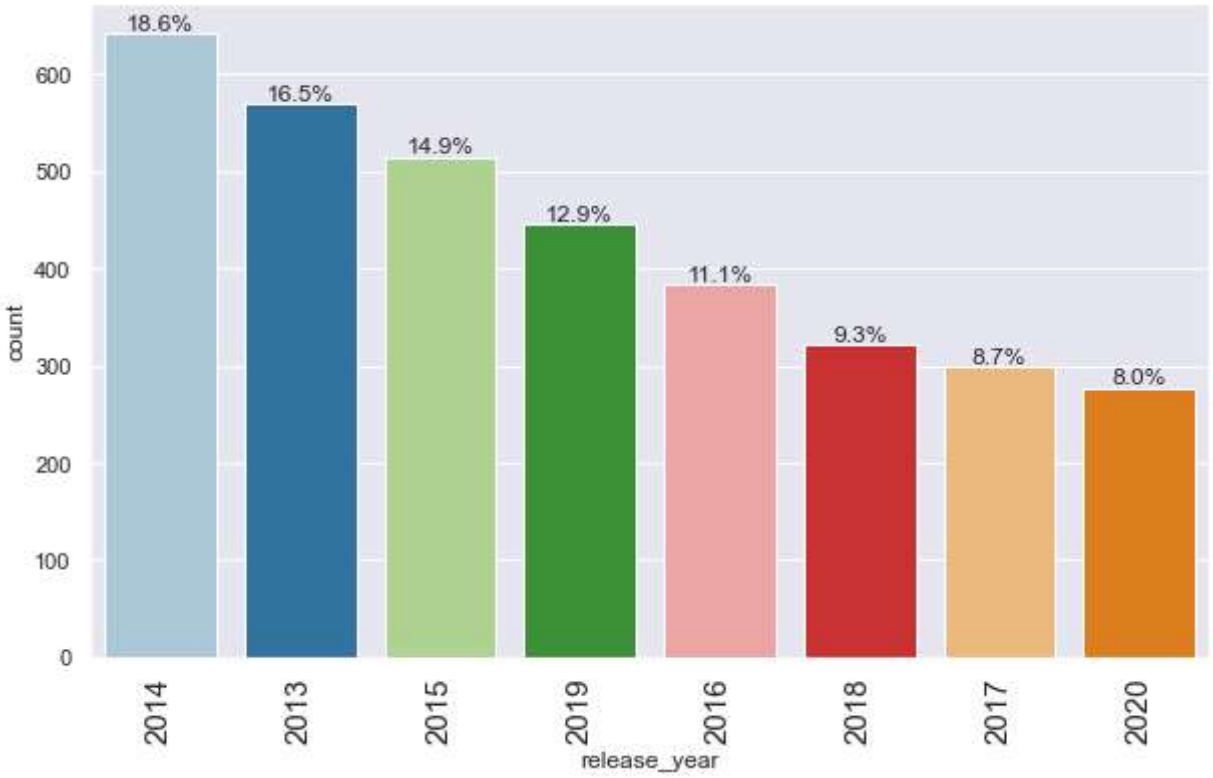
A very large number of phones have memory of 16GB followed by 32GB. There are very few phones with either a very high internal memory or a very low internal memory.

In [221…
```
labeled_barplot(df, "ram", perc=True)
```



A significantly high number of phones have 4GB ram.

In [222…
```
labeled_barplot(df, "release_year", perc=True)
```

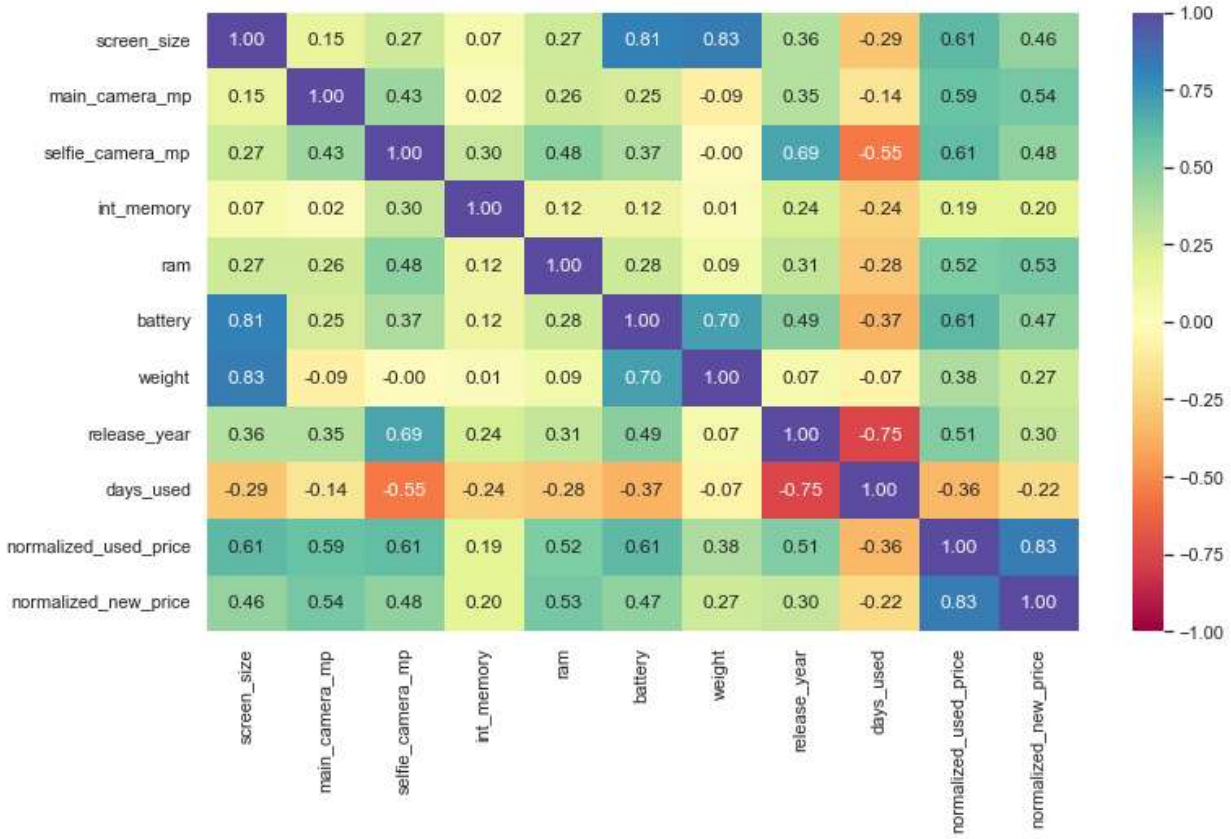


There are phones released between 2013 and 2020 with 2014 with highest representation and 2020 with least representation.

Bivariate Analysis

In [223…

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

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



Battery and weight have a high correlation. Bigger the battery, heavier the phone.

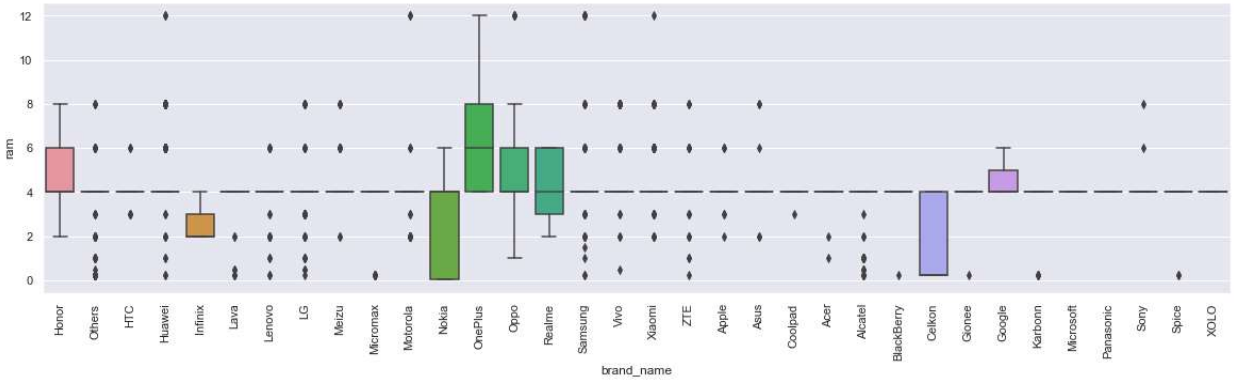
There's also a strong correlation of screen size with battery and weight. Larger the screen size, it needs a bigger battery contributing to higher weight.

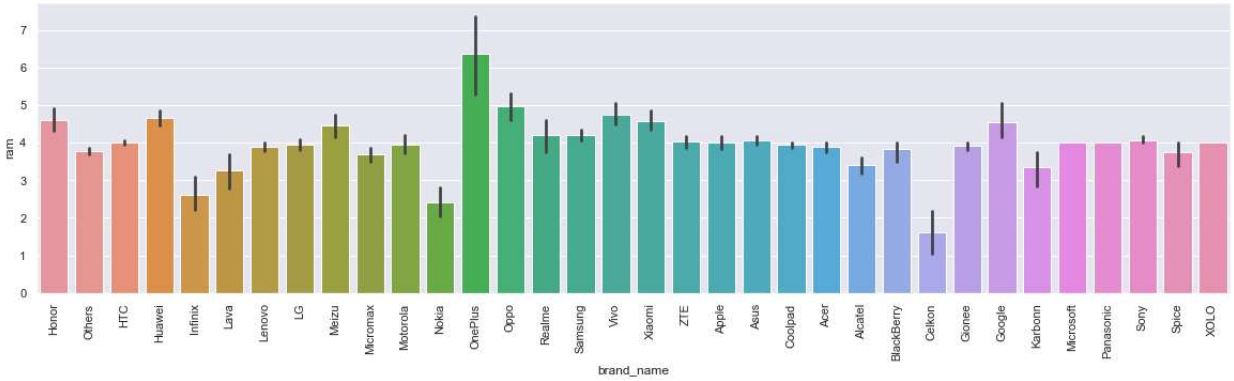
Normalized_used_price has a high correlation with screen_size, main_camera_mp, selfie_camera_mp, ram, battery and release_year. It has a lower correlation with int_memory and weight. It has a negative correlation with days_used.

Brands vs RAM

In [224…

```
plt.figure(figsize=(20, 5))

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

plt.figure(figsize=(20, 5))

sns.barplot(data=df, y="ram", x="brand_name")
plt.xticks(rotation=90)
plt.show()
```





There are outliers present on both the higher end and lower end of RAM vs Brand distribution.

Oneplus appears to provide a higher RAM for most of its models than other brands.

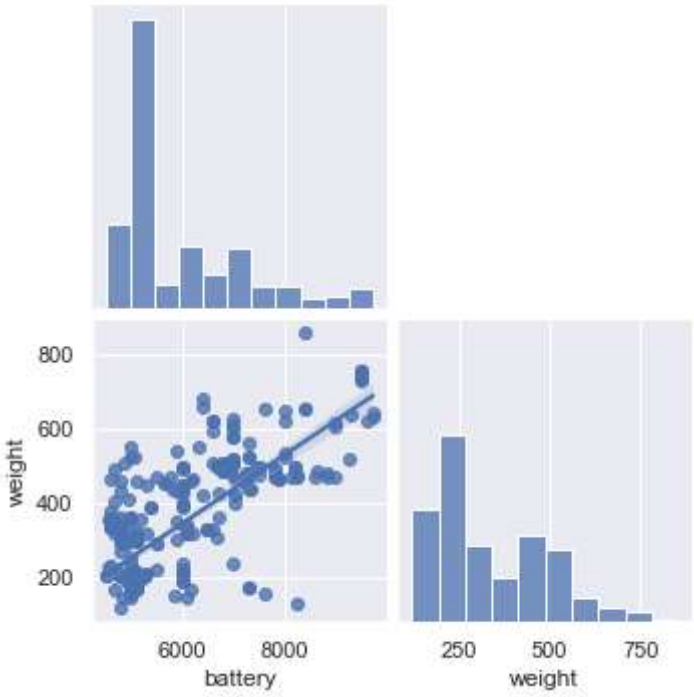
Brands like Celkon, Infinix and Nokia do not have a higher RAM in all their models as compared to other brands.

Overall, RAM distribution is pretty similar for most brands.

Weight vs Battery (>4500 mAh)

In [225…

```
df_large_battery = df[df["battery"] > 4500]
a = sns.pairplot(data=df_large_battery[["battery", "weight"]], corner=True, kind="reg"
plt.show()
```

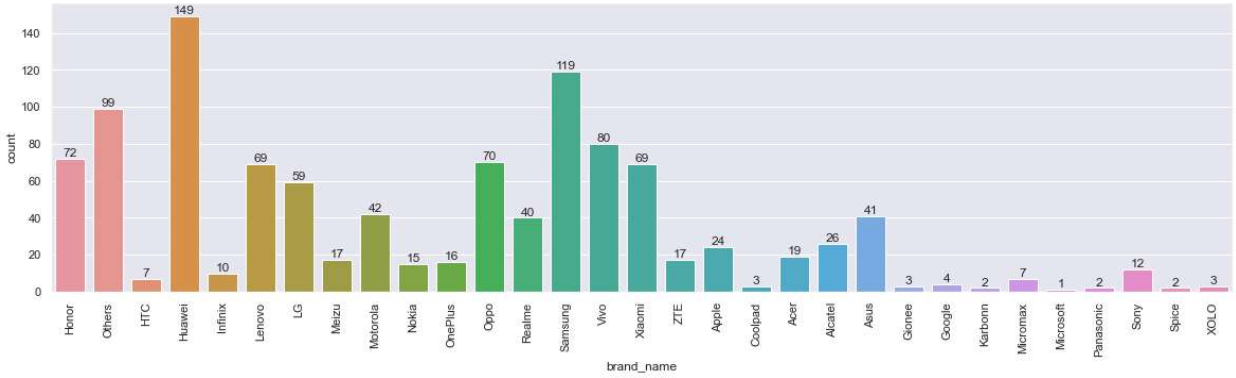


We see a moderately high correlation between battery(>4500 mAh) and weight. So it can be assumed that as the battery size increases, so does the weight.

Brands vs Large screen size >6 inches i.e. >15.24 cm

In [226…

```
df_large_screen = df[df["screen_size"] > 15.24]

plt.figure(figsize=(20, 5))

ax = sns.countplot(data=df_large_screen, x="brand_name")
plt.xticks(rotation=90)
for p in ax.patches:
    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="offset points",
    )
plt.show()
```

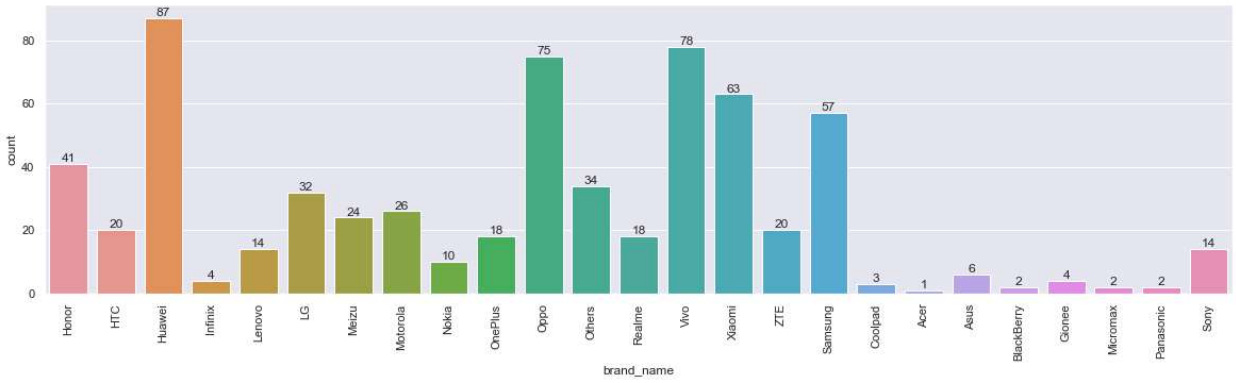


Huawei has most phones with large screen size followed by Samsung. Other unknows brands also have a fairly large screen size. Brands like Microsoft, Karbonn, Panasonic, Spice, Xolo, Micromax, Google, Coolpad have very few phones with larger screen size.

Brand vs Selfie Camera (>8MP)

In [227…

```
df_selfie = df[df["selfie_camera_mp"] > 8]

plt.figure(figsize=(20, 5))

ax = sns.countplot(data=df_selfie, x="brand_name")
plt.xticks(rotation=90)
for p in ax.patches:
    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="offset points",
    )
plt.show()
```



Huawei has a high number of models with good selfie camera followed by Vivo, Oppo, Xiaomi and Samsung. Brands like Acer, Coolpad, Infinix, Asus, Blackberry have a very low number of good selfie camera phones.

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)

In [228…
```
df.isnull().sum()
```

Out[228]:
```
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
```

There are missing values in the columns main_camera_mp, selfie_camera_mp, int_memory, RAM, battery and weight.

These can be replaced with MEDIAN values of each of their brands in respective columns.

In [229…
```
# we first create a copy of the data to avoid changes to it
df1 = df.copy()

df1["main_camera_mp"] = df1["main_camera_mp"].fillna(
    value=df1.groupby(["brand_name"])["main_camera_mp"].transform("median")
)

df1["selfie_camera_mp"] = df1["selfie_camera_mp"].fillna(
    value=df1.groupby(["brand_name"])["selfie_camera_mp"].transform("median")
)

df1["int_memory"] = df1["int_memory"].fillna(
    value=df1.groupby(["brand_name"])["int_memory"].transform("median")
)

df1["ram"] = df1["ram"].fillna(
    value=df1.groupby(["brand_name"])["ram"].transform("median")
)

df1["weight"] = df1["weight"].fillna(
    value=df1.groupby(["brand_name"])["weight"].transform("median")
```

```
)

df1["battery"] = df1["battery"].fillna(
    value=df1.groupby(["brand_name"])["battery"].transform("median")
)

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

Out[229]:
```
brand_name               0
os                       0
screen_size              0
4g                       0
5g                       0
main_camera_mp          10
selfie_camera_mp         0
int_memory               0
ram                      0
battery                  0
weight                   0
release_year             0
days_used                0
normalized_used_price    0
normalized_new_price     0
dtype: int64
```

There are still 10 rows where main_camera_mp is null. These would be for brands that have 1 entry in the data. For them we can populate the MEDIAN value across brands in that column

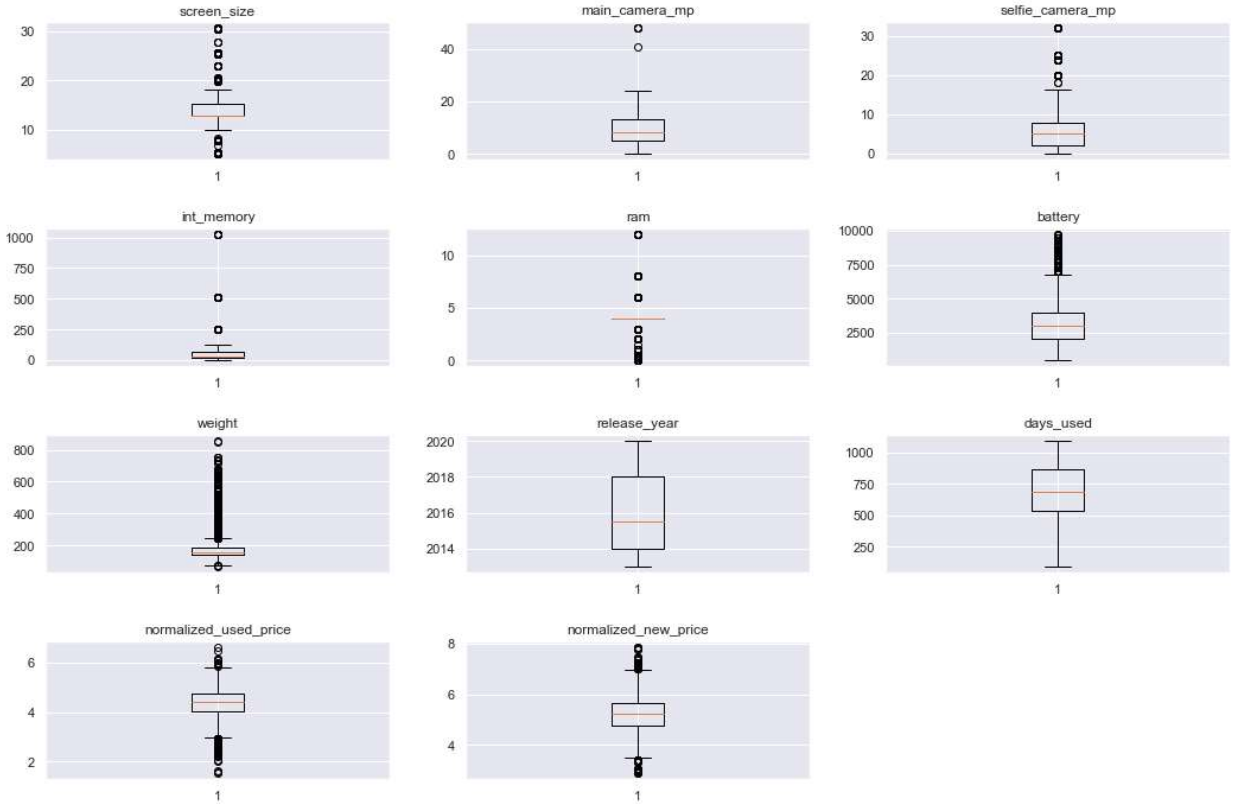
In [230…
```
df1[["main_camera_mp"]] = df1[["main_camera_mp"]].apply(
    lambda x: x.fillna(x.median()), axis=0
)

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

Out[230]:
```
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 have been treated.

In [231…
```
# outlier detection using boxplot
num_cols = df1.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(15, 10))
```

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

plt.show()
```



There are quite a few outliers in almost all columns except release_year and days_used.

But we will not treat them as these are actual values for the most part. Treating these outliers would lead to loss of data.

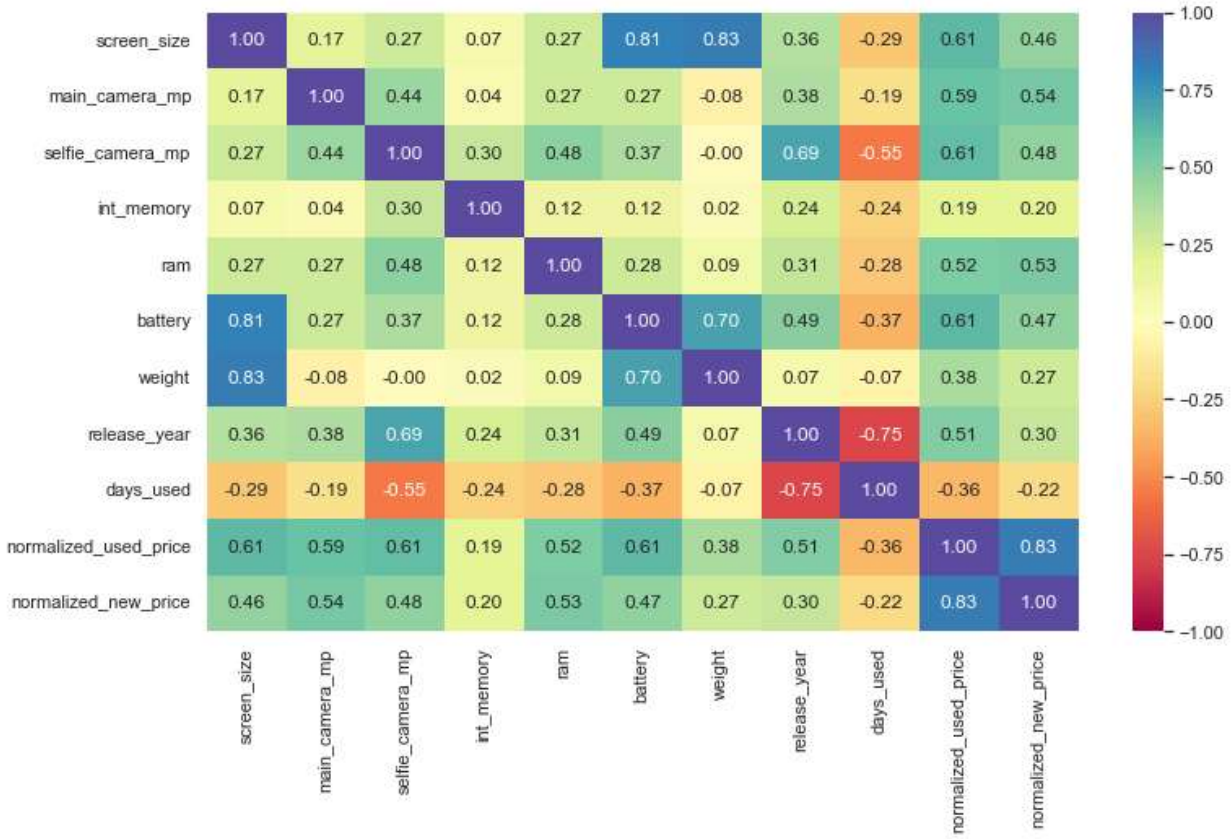
EDA

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

- Performing the bivariate analysis done earlier to check for any updates.

In [232…

```
num_cols = df1.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(12, 7))
sns.heatmap(
    df1[num_cols].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



Battery and weight have a high correlation. Bigger the battery, heavier the phone.

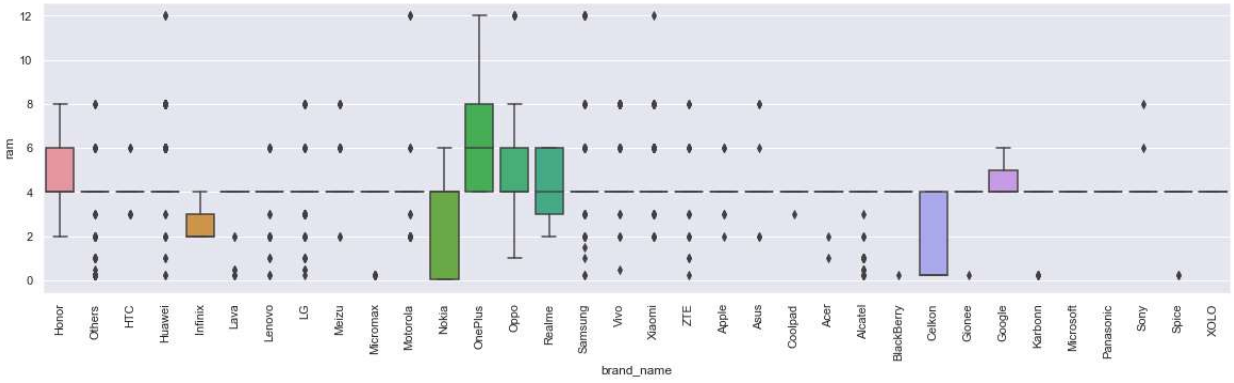
There's also a strong correlation of screen size with battery and weight. Larger the screen size, it needs a bigger battery contributing to higher weight.

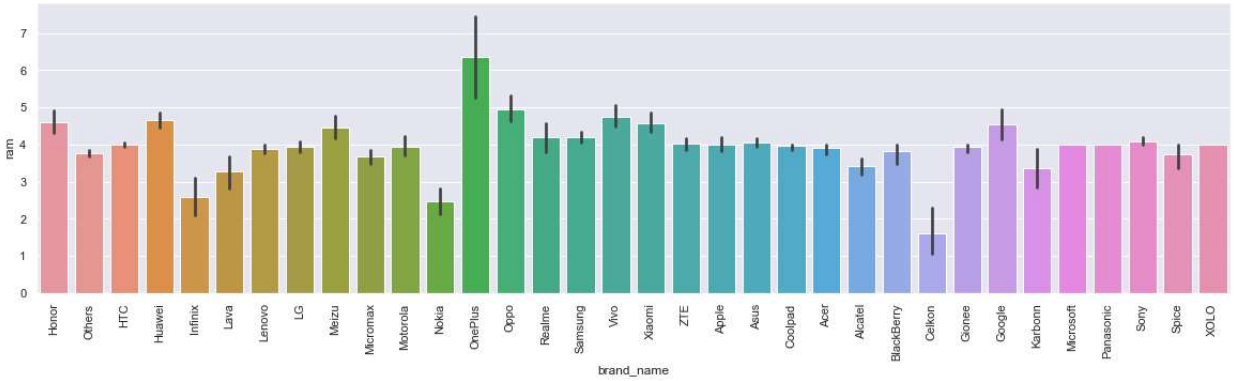
Normalized_used_price has a high correlation with screen_size, main_camera_mp, selfie_camera_mp, ram, battery and release_year. It has a lower correlation with int_memory and weight. It has a negative correlation with days_used.

Brands vs RAM

In [233…

```
plt.figure(figsize=(20, 5))

sns.boxplot(data=df1, x="brand_name", y="ram")
plt.xticks(rotation=90)
plt.show()

plt.figure(figsize=(20, 5))

sns.barplot(data=df1, y="ram", x="brand_name")
plt.xticks(rotation=90)
plt.show()
```





There are outliers present on both the higher end and lower end of RAM vs Brand distribution.

Oneplus appears to provide a higher RAM for most of its models than other brands.

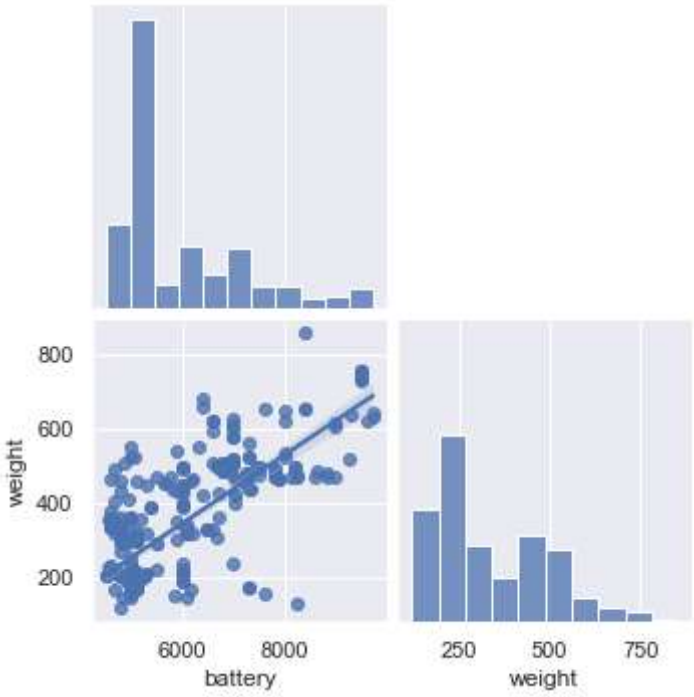
Brands like Celkon, Infinix and Nokia do not have a higher RAM in all their models as compared to other brands.

Overall, RAM distribution is pretty similar for most brands.

Weight vs Battery (>4500 mAh)

In [234…

```
df1_large_battery = df1[df1["battery"] > 4500]
b = sns.pairplot(data=df1_large_battery[["battery", "weight"]], corner=True, kind="reg
plt.show()
```

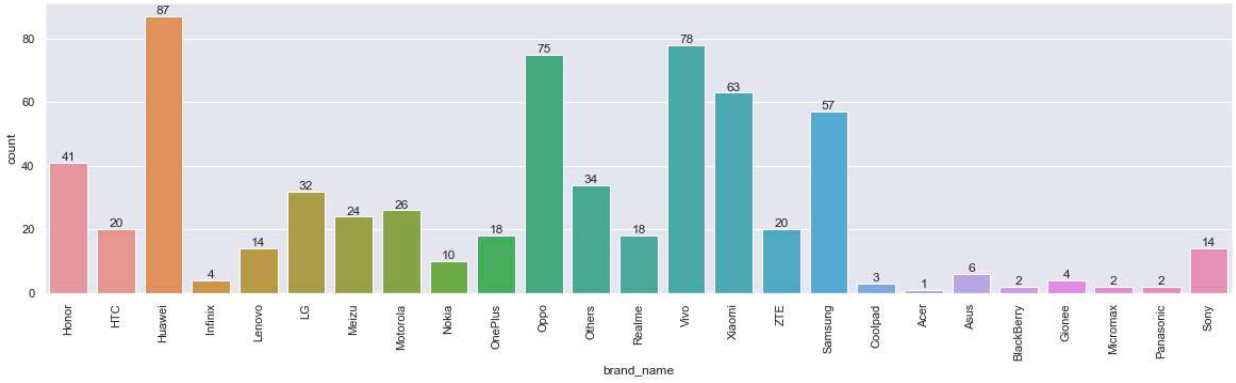


We see a moderately high correlation between battery(>4500 mAh) and weight.

Brand vs Selfie Camera (>8MP)

In [235…

```
df1_selfie = df1[df1["selfie_camera_mp"] > 8]

plt.figure(figsize=(20, 5))

ax = sns.countplot(data=df1_selfie, x="brand_name")
plt.xticks(rotation=90)
for p in ax.patches:
    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="offset points",
    )
plt.show()
```



Huawei has a high number of models with good selfie camera followed by Vivo, Oppo, Xiaomi and Samsung. Brands like Acer, Coolpad, Infinix, Asus, Blackberry have a very low number of good selfie camera phones.

There have been no significant changes after adding missing values.

Data Preparation for Modeling

```
We want to predict the used price of a device
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 [236…

```
# Create dummy variables for the categorical variables
df2 = pd.get_dummies(df1, columns=["brand_name", "os", "4g", "5g"], drop_first=True)

df2.head()
```

Out[236]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year
0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	2020
1	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	2020
2	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	2020
3	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	2020
4	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	2020

5 rows × 49 columns

Dummy variables for categorical columns have been created for model building

In [237…

```
# defining X and y variables
X = df2.drop(["normalized_used_price"], axis=1)
y = df2["normalized_used_price"]
```

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

```
   screen_size  main_camera_mp  selfie_camera_mp  int_memory  ram  battery  \
0        14.50            13.0               5.0        64.0  3.0   3020.0
1        17.30            13.0              16.0       128.0  8.0   4300.0
2        16.69            13.0               8.0       128.0  8.0   4200.0
3        25.50            13.0               8.0        64.0  6.0   7250.0
4        15.32            13.0               8.0        64.0  3.0   5000.0

   weight  release_year  days_used  normalized_new_price  ...  \
0   146.0          2020        127              4.715100  ...
1   213.0          2020        325              5.519018  ...
2   213.0          2020        162              5.884631  ...
3   480.0          2020        345              5.630961  ...
4   185.0          2020        293              4.947837  ...

   brand_name_Spice  brand_name_Vivo  brand_name_XOLO  brand_name_Xiaomi  \
0                 0                0                0                  0
1                 0                0                0                  0
2                 0                0                0                  0
3                 0                0                0                  0
4                 0                0                0                  0

   brand_name_ZTE  os_Others  os_Windows  os_iOS  4g_yes  5g_yes
0               0          0           0       0       1       0
1               0          0           0       0       1       1
2               0          0           0       0       1       1
3               0          0           0       0       1       1
4               0          0           0       0       1       0

[5 rows x 48 columns]
0    4.307572
1    5.162097
2    5.111084
3    5.135387
4    4.389995
Name: normalized_used_price, dtype: float64
```

In [238…

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

In [239…

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

Out[239]:

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

5 rows × 49 columns

In [240…

```
# 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=
```

In [241…

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

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

Model Building - Linear Regression

In [242…

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

OLS Regression Results

Dep. Variable:	normalized_used_price	R-squared:	0.845
Model:	OLS	Adj. R-squared:	0.842
Method:	Least Squares	F-statistic:	268.8
Date:	Sat, 12 Nov 2022	Prob (F-statistic):	0.00
Time:	07:53:06	Log-Likelihood:	124.15
No. Observations:	2417	AIC:	-150.3
Df Residuals:	2368	BIC:	133.4
Df Model:	48		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-46.4646	9.197	-5.052	0.000	-64.500	-28.430
screen_size	0.0244	0.003	7.156	0.000	0.018	0.031
main_camera_mp	0.0208	0.002	13.848	0.000	0.018	0.024
selfie_camera_mp	0.0135	0.001	11.996	0.000	0.011	0.016
int_memory	0.0001	6.97e-05	1.664	0.096	-2.07e-05	0.000
ram	0.0232	0.005	4.515	0.000	0.013	0.033
battery	-1.686e-05	7.27e-06	-2.318	0.021	-3.11e-05	-2.6e-06
weight	0.0010	0.000	7.488	0.000	0.001	0.001
release_year	0.0236	0.005	5.189	0.000	0.015	0.033
days_used	4.196e-05	3.09e-05	1.360	0.174	-1.85e-05	0.000
normalized_new_price	0.4309	0.012	35.134	0.000	0.407	0.455
brand_name_Alcatel	0.0154	0.048	0.324	0.746	-0.078	0.109
brand_name_Apple	-0.0032	0.147	-0.021	0.983	-0.292	0.285
brand_name_Asus	0.0150	0.048	0.313	0.754	-0.079	0.109
brand_name_BlackBerry	-0.0297	0.070	-0.423	0.672	-0.167	0.108
brand_name_Celkon	-0.0463	0.066	-0.699	0.484	-0.176	0.084
brand_name_Coolpad	0.0209	0.073	0.286	0.775	-0.122	0.164
brand_name_Gionee	0.0447	0.058	0.775	0.438	-0.068	0.158
brand_name_Google	-0.0327	0.085	-0.386	0.700	-0.199	0.133
brand_name_HTC	-0.0131	0.048	-0.271	0.786	-0.108	0.081
brand_name_Honor	0.0316	0.049	0.642	0.521	-0.065	0.128
brand_name_Huawei	-0.0022	0.044	-0.049	0.961	-0.089	

```
0.085
brand_name_Infinix        0.1634      0.093      1.753      0.080      -0.019
0.346
brand_name_Karbonn        0.0943      0.067      1.406      0.160      -0.037
0.226
brand_name_LG            -0.0132      0.045     -0.292      0.771      -0.102
0.076
brand_name_Lava           0.0332      0.062      0.533      0.594      -0.089
0.155
brand_name_Lenovo         0.0453      0.045      1.003      0.316      -0.043
0.134
brand_name_Meizu         -0.0130      0.056     -0.232      0.817      -0.123
0.097
brand_name_Micromax      -0.0337      0.048     -0.704      0.481      -0.128
0.060
brand_name_Microsoft      0.0947      0.088      1.072      0.284      -0.079
0.268
brand_name_Motorola      -0.0113      0.050     -0.228      0.820      -0.109
0.086
brand_name_Nokia          0.0705      0.052      1.362      0.173      -0.031
0.172
brand_name_OnePlus        0.0707      0.077      0.913      0.361      -0.081
0.222
brand_name_Oppo           0.0124      0.048      0.259      0.796      -0.081
0.106
brand_name_Others        -0.0080      0.042     -0.191      0.849      -0.091
0.074
brand_name_Panasonic      0.0562      0.056      1.006      0.314      -0.053
0.166
brand_name_Realme         0.0319      0.062      0.517      0.605      -0.089
0.153
brand_name_Samsung       -0.0314      0.043     -0.726      0.468      -0.116
0.053
brand_name_Sony          -0.0616      0.050     -1.221      0.222      -0.161
0.037
brand_name_Spice         -0.0148      0.063     -0.234      0.815      -0.139
0.109
brand_name_Vivo          -0.0155      0.048     -0.320      0.749      -0.111
0.080
brand_name_XOLO           0.0151      0.055      0.276      0.783      -0.092
0.123
brand_name_Xiaomi         0.0868      0.048      1.804      0.071      -0.008
0.181
brand_name_ZTE           -0.0058      0.047     -0.122      0.903      -0.099
0.087
os_Others                -0.0519      0.033     -1.585      0.113      -0.116
0.012
os_Windows               -0.0202      0.045     -0.448      0.654      -0.109
0.068
os_iOS                   -0.0669      0.146     -0.457      0.648      -0.354
0.220
4g_yes                    0.0530      0.016      3.341      0.001       0.022
0.084
5g_yes                   -0.0721      0.031     -2.292      0.022      -0.134      -
0.010
==============================================================================
Omnibus:                      223.220   Durbin-Watson:                   1.911
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              422.514
Skew:                          -0.618   Prob(JB):                     1.79e-92
Kurtosis:                       4.633   Cond. No.                     7.70e+06
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 7.7e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpreting the Regression Results:

1. **Adjusted. R-squared**: It reflects the fit of the model.
   - Adjusted R-squared values generally range from 0 to 1, where a higher value generally indicates a better fit, assuming certain conditions are met.
   - In our case, the value for adj. R-squared is **0.842**, which is good.

1. ***const* coefficient**: It is the Y-intercept.
   - It means that if all the predictor variable coefficients are zero, then the expected output (i.e., Y) would be equal to the *const* coefficient.
   - In our case, the value for `const` coefficient is **-46.4646**

1. **Coefficient of a predictor variable**: It represents the change in the output Y due to a change in the predictor variable (everything else held constant).
   - In our case, the coefficient of `screen_size` is **0.0244.**
   - In our case, the coefficient of `main_camera_mp` is **0.0208.**
   - In our case, the coefficient of `selfie_camera_mp` is **0.0135.**
   - In our case, the coefficient of `ram` is **0.0232.**

Model Performance Check

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

- We will be using metric functions defined in sklearn for RMSE, MAE, and R^2.
- We will define a function to calculate MAPE and adjusted R^2.
  - 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 [243…

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

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


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

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

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

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

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

    return df_perf
```

In [244…

```
# 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[244]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.229856	0.180302	0.844924	0.841713	4.326213

In [245…

```
# 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[245]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.238482	0.184868	0.842315	0.834487	4.505694

Observations

The training R2 is 0.844, so the model is not underfitting

The train and test RMSE and MAE are comparable, so the model is not overfitting either

MAE suggests that the model can predict used mobile prices within a mean error of 0.18 on the test data

MAPE of 4.5 on the test data means that we are able to predict within 4.5% of the mobile prices

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:

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

TEST FOR MULTICOLLINEARITY

- Multicollinearity occurs when predictor variables in a regression model are correlated. This correlation is a problem because predictor variables should be independent. If the correlation between variables is high, it can cause problems when we fit the model and interpret the results. When we have multicollinearity in the linear model, the coefficients that the model suggests are unreliable.
- There are different ways of detecting (or testing) multicollinearity. One such way is by using the Variance Inflation Factor, or VIF.
- **Variance Inflation Factor (VIF)**: Variance inflation factors measure the inflation in the variances of the regression parameter estimates due to collinearities that exist among the

predictors. It is a measure of how much the variance of the estimated regression coefficient β_k is "inflated" by the existence of correlation among the predictor variables in the model.

  - If VIF is 1, then there is no correlation among the kth predictor and the remaining predictor variables, and hence, the variance of β_k is not inflated at all.
- **General Rule of thumb**:
  - If VIF is between 1 and 5, then there is low multicollinearity.
  - If VIF is between 5 and 10, we say there is moderate multicollinearity.
  - If VIF is exceeding 10, it shows signs of high multicollinearity.

Let's define a function to check VIF.

In [246…
```
from statsmodels.stats.outliers_influence import variance_inflation_factor


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
```

In [247…
```
checking_vif(x_train)
```

Out[247]:

	feature	VIF
0	const	3791081.98
1	screen_size	7.68
2	main_camera_mp	2.28
3	selfie_camera_mp	2.81
4	int_memory	1.36
5	ram	2.26
6	battery	4.08
7	weight	6.40
8	release_year	4.90
9	days_used	2.66
10	normalized_new_price	3.12
11	brand_name_Alcatel	3.41
12	brand_name_Apple	13.05
13	brand_name_Asus	3.33
14	brand_name_BlackBerry	1.63
15	brand_name_Celkon	1.77
16	brand_name_Coolpad	1.47
17	brand_name_Gionee	1.95
18	brand_name_Google	1.32
19	brand_name_HTC	3.41
20	brand_name_Honor	3.34
21	brand_name_Huawei	5.98
22	brand_name_Infinix	1.28
23	brand_name_Karbonn	1.57
24	brand_name_LG	4.85
25	brand_name_Lava	1.71
26	brand_name_Lenovo	4.56
27	brand_name_Meizu	2.18
28	brand_name_Micromax	3.36
29	brand_name_Microsoft	1.87
30	brand_name_Motorola	3.27
31	brand_name_Nokia	3.47
32	brand_name_OnePlus	1.44
33	brand_name_Oppo	3.97

	feature	VIF
34	brand_name_Others	9.71
35	brand_name_Panasonic	2.11
36	brand_name_Realme	1.95
37	brand_name_Samsung	7.54
38	brand_name_Sony	2.94
39	brand_name_Spice	1.69
40	brand_name_Vivo	3.65
41	brand_name_XOLO	2.14
42	brand_name_Xiaomi	3.72
43	brand_name_ZTE	3.80
44	os_Others	1.85
45	os_Windows	1.60
46	os_iOS	11.78
47	4g_yes	2.47
48	5g_yes	1.81

- 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

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

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

In [248…

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

    predictors: independent variables
    target: dependent variable
```

```
    high_vif_columns: columns having high VIF
    """
    # empty lists to store adj. R-squared and RMSE values
    adj_r2 = []
    rmse = []

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

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

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

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

    return temp
```

In [249…

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

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

Out[249]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	brand_name_Apple	0.841847	0.232173
1	brand_name_Huawei	0.841847	0.232173
2	brand_name_Others	0.841844	0.232175
3	brand_name_LG	0.841841	0.232177
4	os_iOS	0.841833	0.232183
5	brand_name_Samsung	0.841812	0.232199
6	screen_size	0.838427	0.234670
7	weight	0.838102	0.234906

In [250…

```
col_to_drop = "brand_name_Apple"
x_train2 = x_train.loc[:, ~x_train.columns.str.startswith(col_to_drop)]
x_test2 = x_test.loc[:, ~x_test.columns.str.startswith(col_to_drop)]

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

```
VIF after dropping  brand_name_Apple
```

Out[250]:

	feature	VIF
0	const	3784495.42
1	screen_size	7.64
2	main_camera_mp	2.28
3	selfie_camera_mp	2.79
4	int_memory	1.36
5	ram	2.25
6	battery	4.08
7	weight	6.39
8	release_year	4.89
9	days_used	2.66
10	normalized_new_price	3.10
11	brand_name_Alcatel	3.23
12	brand_name_Asus	3.14
13	brand_name_BlackBerry	1.56
14	brand_name_Celkon	1.73
15	brand_name_Coolpad	1.44
16	brand_name_Gionee	1.89
17	brand_name_Google	1.29
18	brand_name_HTC	3.24
19	brand_name_Honor	3.16
20	brand_name_Huawei	5.58
21	brand_name_Infinix	1.27
22	brand_name_Karbonn	1.54
23	brand_name_LG	4.56
24	brand_name_Lava	1.67
25	brand_name_Lenovo	4.29
26	brand_name_Meizu	2.09
27	brand_name_Micromax	3.21
28	brand_name_Microsoft	1.84
29	brand_name_Motorola	3.11
30	brand_name_Nokia	3.26
31	brand_name_OnePlus	1.40
32	brand_name_Oppo	3.76
33	brand_name_Others	9.08

	feature	VIF
34	brand_name_Panasonic	2.03
35	brand_name_Realme	1.88
36	brand_name_Samsung	6.99
37	brand_name_Sony	2.80
38	brand_name_Spice	1.66
39	brand_name_Vivo	3.45
40	brand_name_XOLO	2.07
41	brand_name_Xiaomi	3.51
42	brand_name_ZTE	3.60
43	os_Others	1.73
44	os_Windows	1.59
45	os_iOS	1.91
46	4g_yes	2.47
47	5g_yes	1.80

In [251…

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

# Check VIF now
vif = checking_vif(x_train3)
print("VIF after dropping ", col_to_drop)
vif
```

```
VIF after dropping  brand_name_Huawei
```

Out[251]:

	feature	VIF
0	const	3784495.25
1	screen_size	7.63
2	main_camera_mp	2.28
3	selfie_camera_mp	2.78
4	int_memory	1.36
5	ram	2.25
6	battery	4.08
7	weight	6.39
8	release_year	4.89
9	days_used	2.66
10	normalized_new_price	3.10
11	brand_name_Alcatel	1.43
12	brand_name_Asus	1.37
13	brand_name_BlackBerry	1.16
14	brand_name_Celkon	1.26
15	brand_name_Coolpad	1.08
16	brand_name_Gionee	1.17
17	brand_name_Google	1.06
18	brand_name_HTC	1.40
19	brand_name_Honor	1.36
20	brand_name_Infinix	1.07
21	brand_name_Karbonn	1.14
22	brand_name_LG	1.61
23	brand_name_Lava	1.15
24	brand_name_Lenovo	1.56
25	brand_name_Meizu	1.18
26	brand_name_Micromax	1.48
27	brand_name_Microsoft	1.54
28	brand_name_Motorola	1.38
29	brand_name_Nokia	1.71
30	brand_name_OnePlus	1.08
31	brand_name_Oppo	1.46
32	brand_name_Others	2.44
33	brand_name_Panasonic	1.19

	feature	VIF
34	brand_name_Realme	1.20
35	brand_name_Samsung	2.00
36	brand_name_Sony	1.35
37	brand_name_Spice	1.16
38	brand_name_Vivo	1.40
39	brand_name_XOLO	1.24
40	brand_name_Xiaomi	1.41
41	brand_name_ZTE	1.45
42	os_Others	1.73
43	os_Windows	1.59
44	os_iOS	1.22
45	4g_yes	2.45
46	5g_yes	1.80

In [252…

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

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

```
VIF after dropping  brand_name_Others
```

Out[252]:

	feature	VIF
0	const	3784034.41
1	screen_size	7.52
2	main_camera_mp	2.28
3	selfie_camera_mp	2.77
4	int_memory	1.36
5	ram	2.24
6	battery	4.07
7	weight	6.32
8	release_year	4.89
9	days_used	2.66
10	normalized_new_price	3.10
11	brand_name_Alcatel	1.16
12	brand_name_Asus	1.13
13	brand_name_BlackBerry	1.11
14	brand_name_Celkon	1.17
15	brand_name_Coolpad	1.04
16	brand_name_Gionee	1.06
17	brand_name_Google	1.03
18	brand_name_HTC	1.15
19	brand_name_Honor	1.17
20	brand_name_Infinix	1.05
21	brand_name_Karbonn	1.06
22	brand_name_LG	1.22
23	brand_name_Lava	1.06
24	brand_name_Lenovo	1.19
25	brand_name_Meizu	1.08
26	brand_name_Micromax	1.18
27	brand_name_Microsoft	1.49
28	brand_name_Motorola	1.17
29	brand_name_Nokia	1.44
30	brand_name_OnePlus	1.06
31	brand_name_Oppo	1.23
32	brand_name_Panasonic	1.07
33	brand_name_Realme	1.13

	feature	VIF
34	brand_name_Samsung	1.35
35	brand_name_Sony	1.15
36	brand_name_Spice	1.07
37	brand_name_Vivo	1.20
38	brand_name_XOLO	1.10
39	brand_name_Xiaomi	1.19
40	brand_name_ZTE	1.17
41	os_Others	1.72
42	os_Windows	1.59
43	os_iOS	1.14
44	4g_yes	2.44
45	5g_yes	1.80

In [253…

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

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

```
VIF after dropping  screen_size
```

Out[253]:

	feature	VIF
0	const	3652032.17
1	main_camera_mp	2.28
2	selfie_camera_mp	2.77
3	int_memory	1.36
4	ram	2.24
5	battery	3.82
6	weight	2.99
7	release_year	4.71
8	days_used	2.65
9	normalized_new_price	3.05
10	brand_name_Alcatel	1.15
11	brand_name_Asus	1.13
12	brand_name_BlackBerry	1.11
13	brand_name_Celkon	1.16
14	brand_name_Coolpad	1.04
15	brand_name_Gionee	1.06
16	brand_name_Google	1.03
17	brand_name_HTC	1.15
18	brand_name_Honor	1.16
19	brand_name_Infinix	1.05
20	brand_name_Karbonn	1.06
21	brand_name_LG	1.22
22	brand_name_Lava	1.06
23	brand_name_Lenovo	1.19
24	brand_name_Meizu	1.08
25	brand_name_Micromax	1.18
26	brand_name_Microsoft	1.49
27	brand_name_Motorola	1.17
28	brand_name_Nokia	1.44
29	brand_name_OnePlus	1.06
30	brand_name_Oppo	1.23
31	brand_name_Panasonic	1.07
32	brand_name_Realme	1.13
33	brand_name_Samsung	1.35

	feature	VIF
34	brand_name_Sony	1.15
35	brand_name_Spice	1.07
36	brand_name_Vivo	1.19
37	brand_name_XOLO	1.09
38	brand_name_Xiaomi	1.19
39	brand_name_ZTE	1.17
40	os_Others	1.52
41	os_Windows	1.59
42	os_iOS	1.14
43	4g_yes	2.44
44	5g_yes	1.80

- We have dealt with multicollinearity in the data
- Let's rebuild the model using the updated set of predictors variables

In [254…

```
olsmod1 = sm.OLS(y_train, x_train5).fit()
print(olsmod1.summary())
```

OLS Regression Results

Dep. Variable:	normalized_used_price	R-squared:	0.841
Model:	OLS	Adj. R-squared:	0.839
Method:	Least Squares	F-statistic:	286.1
Date:	Sat, 12 Nov 2022	Prob (F-statistic):	0.00
Time:	07:55:44	Log-Likelihood:	97.430
No. Observations:	2417	AIC:	-104.9
Df Residuals:	2372	BIC:	155.7
Df Model:	44		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-58.9014	9.119	-6.459	0.000	-76.784	-41.019
main_camera_mp	0.0212	0.002	13.968	0.000	0.018	0.024
selfie_camera_mp	0.0138	0.001	12.217	0.000	0.012	0.016
int_memory	9.667e-05	7.04e-05	1.374	0.170	-4.13e-05	0.000
ram	0.0236	0.005	4.570	0.000	0.014	0.034
battery	-3.602e-06	7.11e-06	-0.507	0.612	-1.75e-05	1.03e-05
weight	0.0017	9.21e-05	18.414	0.000	0.002	0.002
release_year	0.0299	0.005	6.613	0.000	0.021	0.039
days_used	2.701e-05	3.11e-05	0.868	0.385	-3.4e-05	8.8e-05
normalized_new_price	0.4420	0.012	36.066	0.000	0.418	0.466
brand_name_Alcatel	0.0408	0.028	1.461	0.144	-0.014	0.095
brand_name_Asus	0.0229	0.028	0.815	0.415	-0.032	0.078
brand_name_BlackBerry	-0.0245	0.058	-0.419	0.675	-0.139	0.090
brand_name_Celkon	-0.0265	0.054	-0.488	0.626	-0.133	0.080
brand_name_Coolpad	0.0357	0.062	0.578	0.564	-0.086	0.157
brand_name_Gionee	0.0376	0.043	0.874	0.382	-0.047	0.122
brand_name_Google	-0.0386	0.076	-0.512	0.609	-0.187	0.109
brand_name_HTC	-0.0099	0.028	-0.350	0.727	-0.065	0.045
brand_name_Honor	0.0559	0.029	1.908	0.056	-0.002	0.113
brand_name_Infinix	0.1720	0.085	2.020	0.043	0.005	0.339
brand_name_Karbonn	0.1246	0.056	2.240	0.025	0.015	0.234
brand_name_LG	-0.0107	0.023	-0.468	0.640	-0.056	

```
0.034
brand_name_Lava           0.0517      0.049      1.046      0.296      -0.045
0.149
brand_name_Lenovo         0.0565      0.023      2.423      0.015       0.011
0.102
brand_name_Meizu         -0.0073      0.040     -0.182      0.855      -0.086
0.071
brand_name_Micromax      -0.0231      0.029     -0.806      0.420      -0.079
0.033
brand_name_Microsoft      0.1004      0.080      1.262      0.207      -0.056
0.256
brand_name_Motorola      -0.0115      0.030     -0.384      0.701      -0.070
0.047
brand_name_Nokia          0.0683      0.034      2.026      0.043       0.002
0.134
brand_name_OnePlus        0.0876      0.067      1.306      0.192      -0.044
0.219
brand_name_Oppo           0.0200      0.027      0.747      0.455      -0.033
0.073
brand_name_Panasonic      0.0707      0.040      1.757      0.079      -0.008
0.150
brand_name_Realme         0.0352      0.047      0.744      0.457      -0.058
0.128
brand_name_Samsung       -0.0246      0.019     -1.328      0.184      -0.061
0.012
brand_name_Sony          -0.0557      0.032     -1.747      0.081      -0.118
0.007
brand_name_Spice         -0.0163      0.051     -0.320      0.749      -0.116
0.083
brand_name_Vivo          -0.0001      0.028     -0.005      0.996      -0.055
0.055
brand_name_XOLO           0.0350      0.040      0.883      0.377      -0.043
0.113
brand_name_Xiaomi         0.0940      0.028      3.416      0.001       0.040
0.148
brand_name_ZTE            0.0002      0.027      0.009      0.993      -0.052
0.052
os_Others                -0.1309      0.030     -4.370      0.000      -0.190      -
0.072
os_Windows               -0.0182      0.046     -0.400      0.689      -0.108
0.071
os_iOS                   -0.0799      0.046     -1.736      0.083      -0.170
0.010
4g_yes                    0.0518      0.016      3.250      0.001       0.021
0.083
5g_yes                   -0.0839      0.032     -2.649      0.008      -0.146      -
0.022
==============================================================================
Omnibus:                      234.903   Durbin-Watson:                   1.908
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              441.342
Skew:                          -0.647   Prob(JB):                     1.46e-96
Kurtosis:                       4.645   Cond. No.                     7.55e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 7.55e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpreting the Regression Results:

1. **std err**: It reflects the level of accuracy of the coefficients.
   - The lower it is, the higher is the level of accuracy.

1. **P>|t|**: It is p-value.

   - For each independent feature, there is a null hypothesis and an alternate hypothesis. Here β_i is the coefficient of the ith independent variable.
     - H_o : Independent feature is not significant ($\beta_i = 0$)
     - H_a : Independent feature is that it is significant ($\beta_i \neq 0$)
   - (P>|t|) gives the p-value for each independent feature to check that null hypothesis. We are considering 0.05 (5%) as significance level.
     - A p-value of less than 0.05 is considered to be statistically significant.

1. **Confidence Interval**: It represents the range in which our coefficients are likely to fall (with a likelihood of 95%).

Observations

- We can see that adj. R-squared has dropped from 0.842 to 0.839, which shows that the dropped columns did not have much effect on the model
- As there is no multicollinearity, we can look at the p-values of predictor variables to check their significance

- Some of the dummy variables in the data have p-value > 0.05. So, they are not significant and we'll drop them
- 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

Note: 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 [255…

```
# initial list of columns
predictors = x_train5.copy()
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', 'release_year', 'nor
malized_new_price', 'brand_name_Karbonn', 'brand_name_Lenovo', 'brand_name_Xiaomi',
'os_Others', '4g_yes', '5g_yes']
```

In [256…

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

In [182…

```
olsmod2 = sm.OLS(y_train, x_train6).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:                     1043.
Date:                   Sat, 12 Nov 2022   Prob (F-statistic):               0.00
Time:                           00:54:57   Log-Likelihood:                 77.798
No. Observations:                   2417   AIC:                            -129.6
Df Residuals:                       2404   BIC:                            -54.32
Df Model:                             12
Covariance Type:               nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                -58.4926      6.834     -8.559      0.000     -71.893     -45.
092
main_camera_mp         0.0210      0.001     15.032      0.000       0.018       0.
024
selfie_camera_mp       0.0143      0.001     13.359      0.000       0.012       0.
016
ram                    0.0212      0.005      4.268      0.000       0.011       0.
031
weight                 0.0016   6.03e-05     27.148      0.000       0.002       0.
002
release_year           0.0297      0.003      8.766      0.000       0.023       0.
036
normalized_new_price   0.4345      0.011     39.985      0.000       0.413       0.
456
brand_name_Karbonn     0.1214      0.055      2.215      0.027       0.014       0.
229
brand_name_Lenovo      0.0524      0.022      2.417      0.016       0.010       0.
095
brand_name_Xiaomi      0.0883      0.026      3.434      0.001       0.038       0.
139
os_Others             -0.1300      0.027     -4.768      0.000      -0.183      -0.
077
4g_yes                 0.0458      0.015      3.041      0.002       0.016       0.
075
5g_yes                -0.0621      0.031     -2.032      0.042      -0.122      -0.
002
==============================================================================
Omnibus:                      245.082   Durbin-Watson:                   1.914
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              480.174
Skew:                          -0.657   Prob(JB):                    5.39e-105
Kurtosis:                       4.744   Cond. No.                     2.89e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 2.89e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [257…

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

```
olsmod2_train_perf = model_performance_regression(olsmod2, x_train6, y_train)
olsmod2_train_perf
```

```
Training Performance
```

Out[257]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.234306	0.183222	0.838861	0.837989	4.407247

In [258…

```
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmod2_test_perf = model_performance_regression(olsmod2, x_test6, y_test)
olsmod2_test_perf
```

```
Test Performance
```

Out[258]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.241711	0.187639	0.838016	0.835958	4.579522

Observations

- Now no feature has p-value greater than 0.05, so we'll consider the features in *x_train6* as the final set of predictor variables and *olsmod2* as the final model to move forward with
- Now adjusted R-squared is 0.83, i.e., our model is able to explain ~83% of the variance
- The adjusted R-squared in *olsmod1* (where we considered the variables without multicollinearity) was 0.84
  - This shows that the variables we dropped were not affecting the model
- 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 *olsmod2*.

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

TEST FOR LINEARITY AND INDEPENDENCE

Why the test?

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

make us incorrectly conclude a parameter to be statistically significant.

How to check linearity and independence?

- Make a plot of fitted values vs residuals.
- If they don't follow any pattern, then we say the model is linear and residuals are independent.
- Otherwise, the model is showing signs of non-linearity and residuals are not independent.

How to fix if this assumption is not followed?

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

In [260…

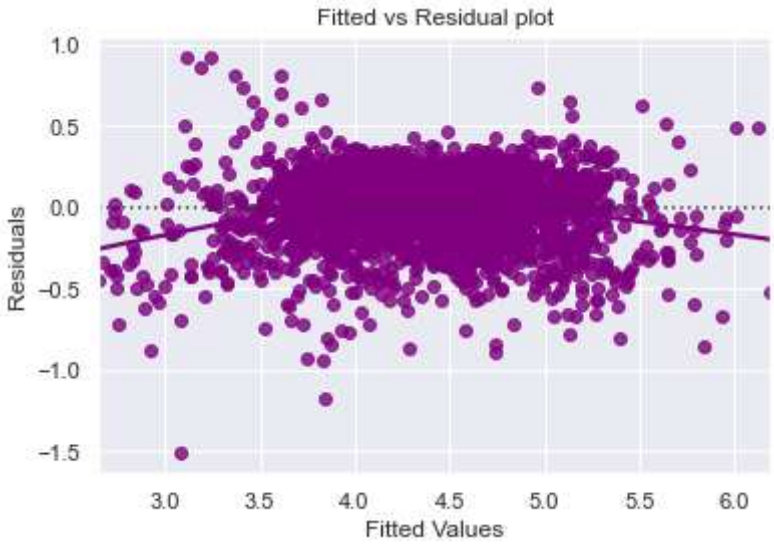
```
# let us create a dataframe with actual, fitted and residual values
df_pred = pd.DataFrame()

df_pred["Actual Values"] = y_train  # actual values
df_pred["Fitted Values"] = olsmod2.fittedvalues  # predicted values
df_pred["Residuals"] = olsmod2.resid  # residuals

df_pred.head()
```

Out[260]:

	Actual Values	Fitted Values	Residuals
3026	4.087488	3.861134	0.226354
1525	4.448399	4.640090	-0.191691
1128	4.315353	4.280133	0.035220
3003	4.282068	4.184221	0.097847
2907	4.456438	4.485982	-0.029544

In [261…

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

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



- The scatter plot shows the distribution of residuals (errors) vs fitted values (predicted values).
- If there exist any pattern in this plot, we consider it as signs of non-linearity in the data and a pattern means that the model doesn't capture non-linear effects.
- We see concentration of data towards the center but there does not appear a clear pattern in the plot above to completely justify the entire distribution. Hence, the assumptions of linearity and independence are satisfied.

TEST FOR NORMALITY

Why the test?

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

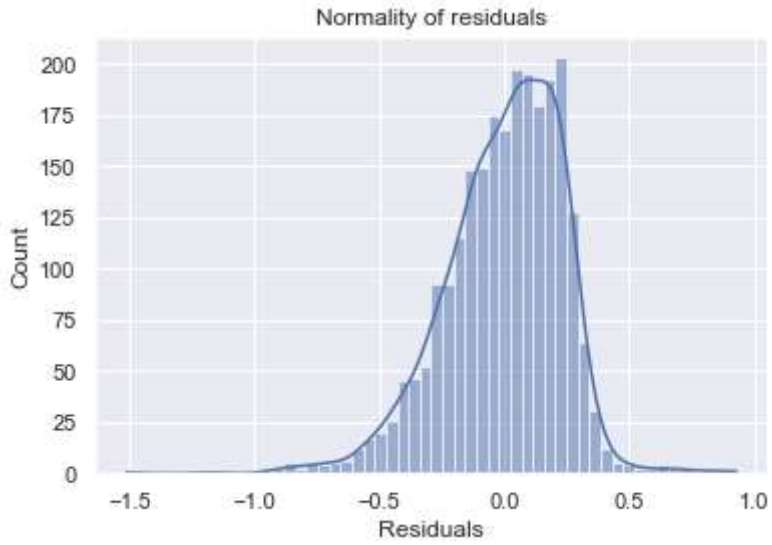
How to check normality?

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

How to fix if this assumption is not followed?

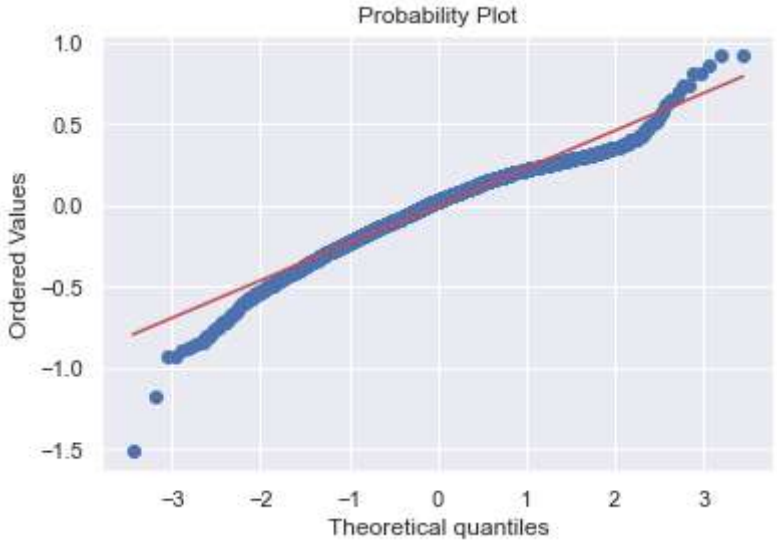
- We can apply transformations like log, exponential, arcsinh, etc. as per our data.

In [262…
```
sns.histplot(data=df_pred, x="Residuals", kde=True)
plt.title("Normality of residuals")
plt.show()
```



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

In [263…
```
import pylab
import scipy.stats as stats

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



- The residuals more or less follow a straight line except for the tails.
- Let's check the results of the Shapiro-Wilk test.

In [264…
```
stats.shapiro(df_pred["Residuals"])
```

Out[264]:
```
ShapiroResult(statistic=0.9679602384567261, pvalue=8.601209196001604e-23)
```

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

TEST FOR HOMOSCEDASTICITY

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

Why the test?

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

How to check for homoscedasticity?

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

How to fix if this assumption is not followed?

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

In [265…
```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

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

Out[265]:
```
[('F statistic', 1.0121600929061185), ('p-value', 0.4172554627513414)]
```

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

Predictions on test data

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

In [266…
```
# predictions on the test set
pred = olsmod2.predict(x_test6)

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

Out[266]:

	Actual	Predicted
1995	4.566741	4.375713
2341	3.696103	3.996781
1913	3.592093	3.639062
688	4.306495	4.091126
650	4.522115	5.181058
2291	4.259294	4.386158
40	4.997685	5.439343
1884	3.875359	4.046184
2538	4.206631	4.062773
45	5.380450	5.219544

- We can observe here that our model has returned pretty good prediction results, and the actual and predicted values are comparable

Final Model

Let's recreate the final model and print it's summary to gain insights.

In [268…
```
# Equation of linear regression
Equation = "Used Phone Price ="
print(Equation, end=" ")
for i in range(len(x_train6.columns)):
    if i == 0:
        print(np.round(olsmod2.params[i], 4), "+", end=" ")
    elif i != len(x_train6.columns) - 1:
        print(
            "(",
            np.round(olsmod2.params[i], 4),
```

```
            ")*(",
            x_train6.columns[i],
            ")",
            "+",
            end="  ",
        )
    else:
        print("(", np.round(olsmod2.params[i], 4), ")*(", x_train6.columns[i], ")")
```

```
Used Phone Price = -58.4926 + ( 0.021 )*( main_camera_mp ) +  ( 0.0143 )*( selfie_cam
era_mp ) +  ( 0.0212 )*( ram ) +  ( 0.0016 )*( weight ) +  ( 0.0297 )*( release_year
) +  ( 0.4345 )*( normalized_new_price ) +  ( 0.1214 )*( brand_name_Karbonn ) +  ( 0.
0524 )*( brand_name_Lenovo ) +  ( 0.0883 )*( brand_name_Xiaomi ) +  ( -0.13 )*( os_Ot
hers ) +  ( 0.0458 )*( 4g_yes ) +  ( -0.0621 )*( 5g_yes )
```

In [269…

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

In [270…

```
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:                     1043.
Date:                   Sat, 12 Nov 2022   Prob (F-statistic):               0.00
Time:                           08:00:16   Log-Likelihood:                 77.798
No. Observations:                   2417   AIC:                            -129.6
Df Residuals:                       2404   BIC:                            -54.32
Df Model:                             12
Covariance Type:               nonrobust
==============================================================================
===
                         coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const                  -58.4926      6.834     -8.559      0.000     -71.893     -45.
092
main_camera_mp           0.0210      0.001     15.032      0.000       0.018       0.
024
selfie_camera_mp         0.0143      0.001     13.359      0.000       0.012       0.
016
ram                      0.0212      0.005      4.268      0.000       0.011       0.
031
weight                   0.0016   6.03e-05     27.148      0.000       0.002       0.
002
release_year             0.0297      0.003      8.766      0.000       0.023       0.
036
normalized_new_price     0.4345      0.011     39.985      0.000       0.413       0.
456
brand_name_Karbonn       0.1214      0.055      2.215      0.027       0.014       0.
229
brand_name_Lenovo        0.0524      0.022      2.417      0.016       0.010       0.
095
brand_name_Xiaomi        0.0883      0.026      3.434      0.001       0.038       0.
139
os_Others               -0.1300      0.027     -4.768      0.000      -0.183      -0.
077
4g_yes                   0.0458      0.015      3.041      0.002       0.016       0.
075
5g_yes                  -0.0621      0.031     -2.032      0.042      -0.122      -0.
002
==============================================================================
Omnibus:                      245.082   Durbin-Watson:                   1.914
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              480.174
Skew:                          -0.657   Prob(JB):                    5.39e-105
Kurtosis:                       4.744   Cond. No.                     2.89e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 2.89e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [271…

```
# 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[271]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.234306	0.183222	0.838861	0.837989	4.407247

In [272…

```
# 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[272]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.241711	0.187639	0.838016	0.835958	4.579522

- 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 4.5% of the anime ratings
- Hence, we can conclude the model *olsmodel_final* is good for prediction as well as inference purposes

Actionable Insights and Recommendations

- Attributes main_camera_mp , selfie_camera_mp, ram are important parameters that decide the value of a used phone
- Attributes weight, release_year, brand_name_Karbonn, brand_name_Lenovo, brand_name_Xiaomi, 4g_yes and 5g_yes also contribute to the price of phone
- Phone models like Karbonn, Lenovo and Xiaomi are not popular and can be discontinued
- The data is android heavy, so doesn not justify popularity of other os.
- The data does not show which phone brands/models are popular among which age group of people
- Adding more 5g enabled phones could boost sales