ReCell Project

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 [6]:
```
# Installing the libraries with the specified version.
!pip install scikit-learn seaborn matplotlib numpy pandas -q --user
```

In [8]:
```
# 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

# Perform the Shapiro-Wilk test for normality
from scipy.stats import shapiro
```

Loading the dataset

In [10]:
```
file_path = '/Users/estarconsulting/Desktop/used_device_data.csv'  # File path to the dataset
data = pd.read_csv(file_path)  # Reading the dataset into a pandas DataFrame
```

Data Overview

Displaying the first few rows of the dataset

In [14]:
```
data.head()
```

Out[14]:

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

Checking the shape of the dataset

In [16]:
```
data.shape
```

Out[16]: (3454, 15)

Checking the data types of the columns for the dataset

In [18]:
```
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
```

Statistical summary of the dataset

In [20]:
```
data.describe()
```

Out[20]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	day
count	3454.000000	3275.000000	3452.000000	3450.000000	3450.000000	3448.000000	3447.000000	3454.000000	3454.
mean	13.713115	9.460208	6.554229	54.573099	4.036122	3133.402697	182.751871	2015.965258	674.
std	3.805280	4.815461	6.970372	84.972371	1.365105	1299.682844	88.413228	2.298455	248.
min	5.080000	0.080000	0.000000	0.010000	0.020000	500.000000	69.000000	2013.000000	91.
25%	12.700000	5.000000	2.000000	16.000000	4.000000	2100.000000	142.000000	2014.000000	533.
50%	12.830000	8.000000	5.000000	32.000000	4.000000	3000.000000	160.000000	2015.500000	690.
75%	15.340000	13.000000	8.000000	64.000000	4.000000	4000.000000	185.000000	2018.000000	868.
max	30.710000	48.000000	32.000000	1024.000000	12.000000	9720.000000	855.000000	2020.000000	1094.

Checking for duplicate values

In [22]:
```
data.duplicated().sum()
```

Out[22]:
```
0
```

Checking for missing values

In [24]:
```
data.isnull().sum()
```

Out[24]:
```
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 [25]:
```
# creating a copy of the data so that original data remains unchanged
df = data.copy()
```

Exploratory Data Analysis

Univariate Analysis

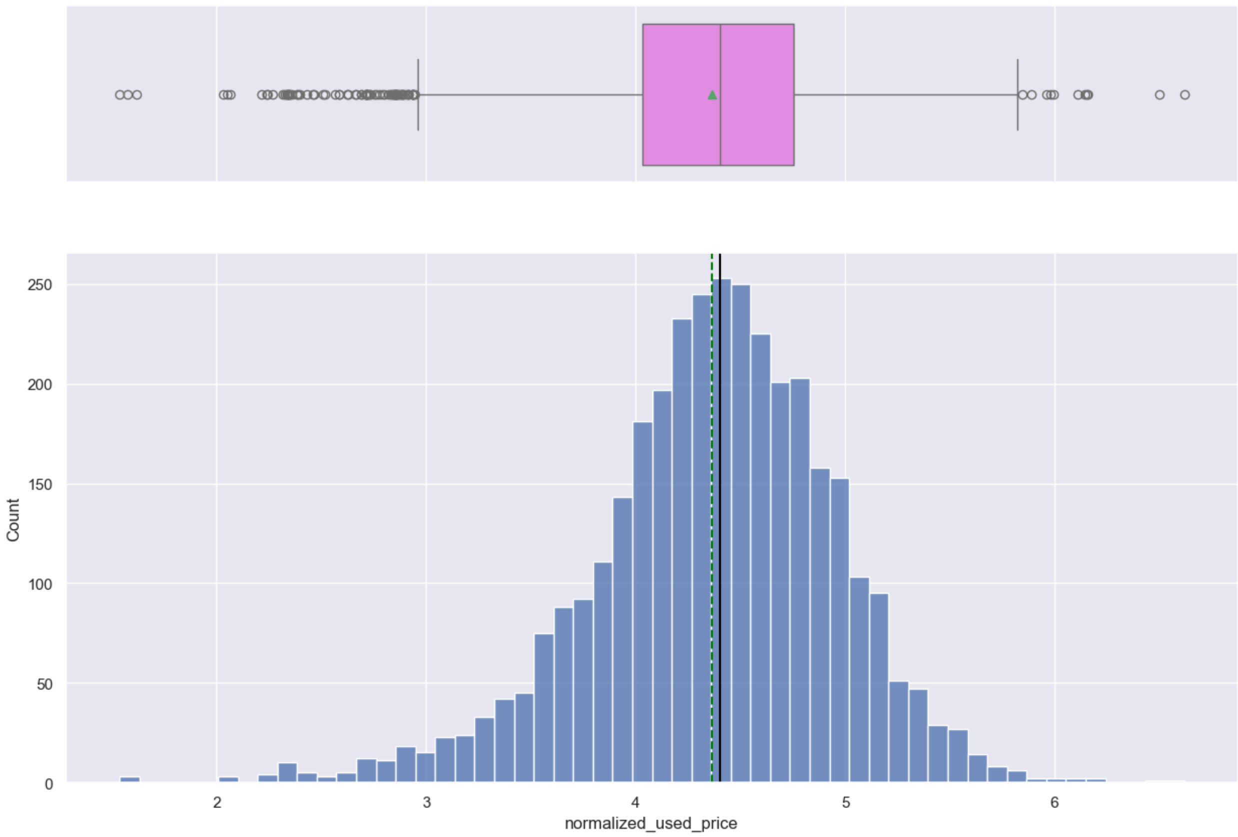
In [28]:
```
# function to plot a boxplot and a histogram along the same scale.


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 column
    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 [29]:
```
# 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))
```

```python
    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
```

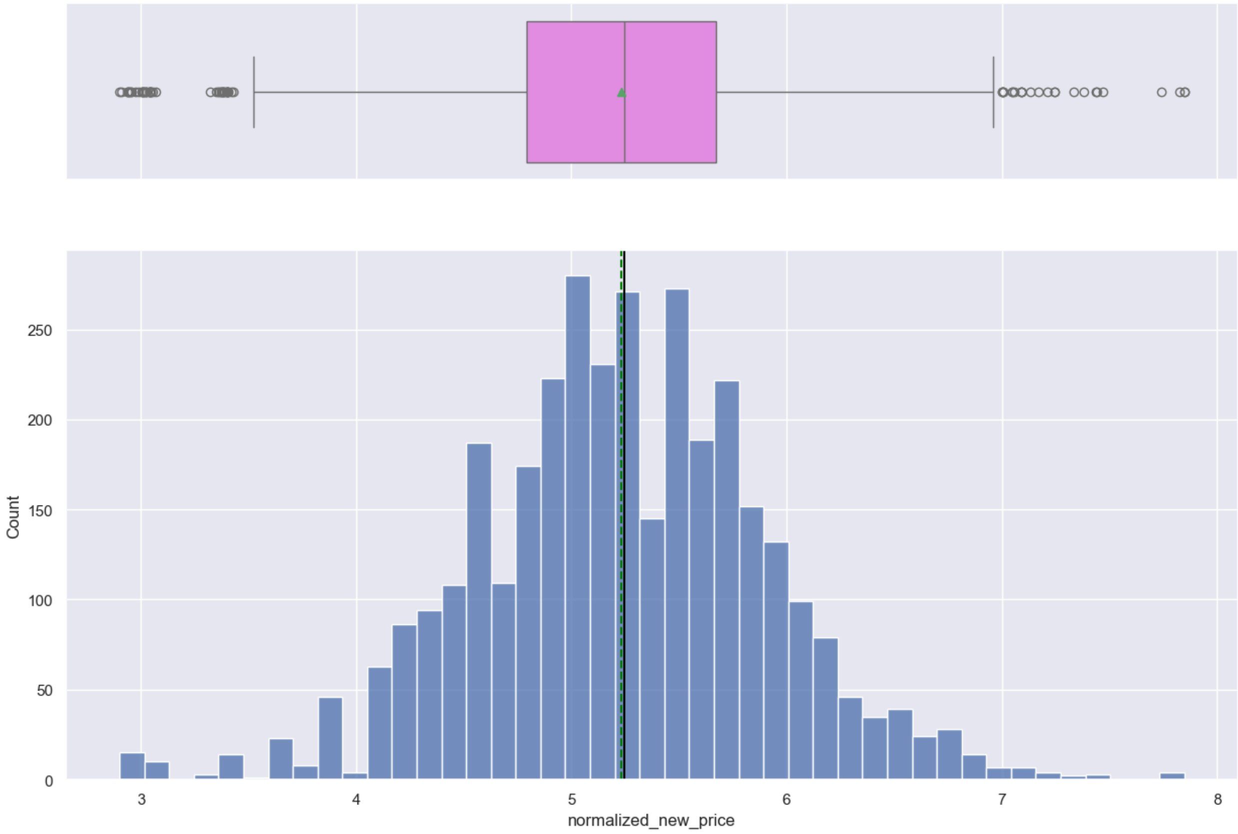
normalized_used_price

In [31]:
```python
histogram_boxplot(df, "normalized_used_price")
```



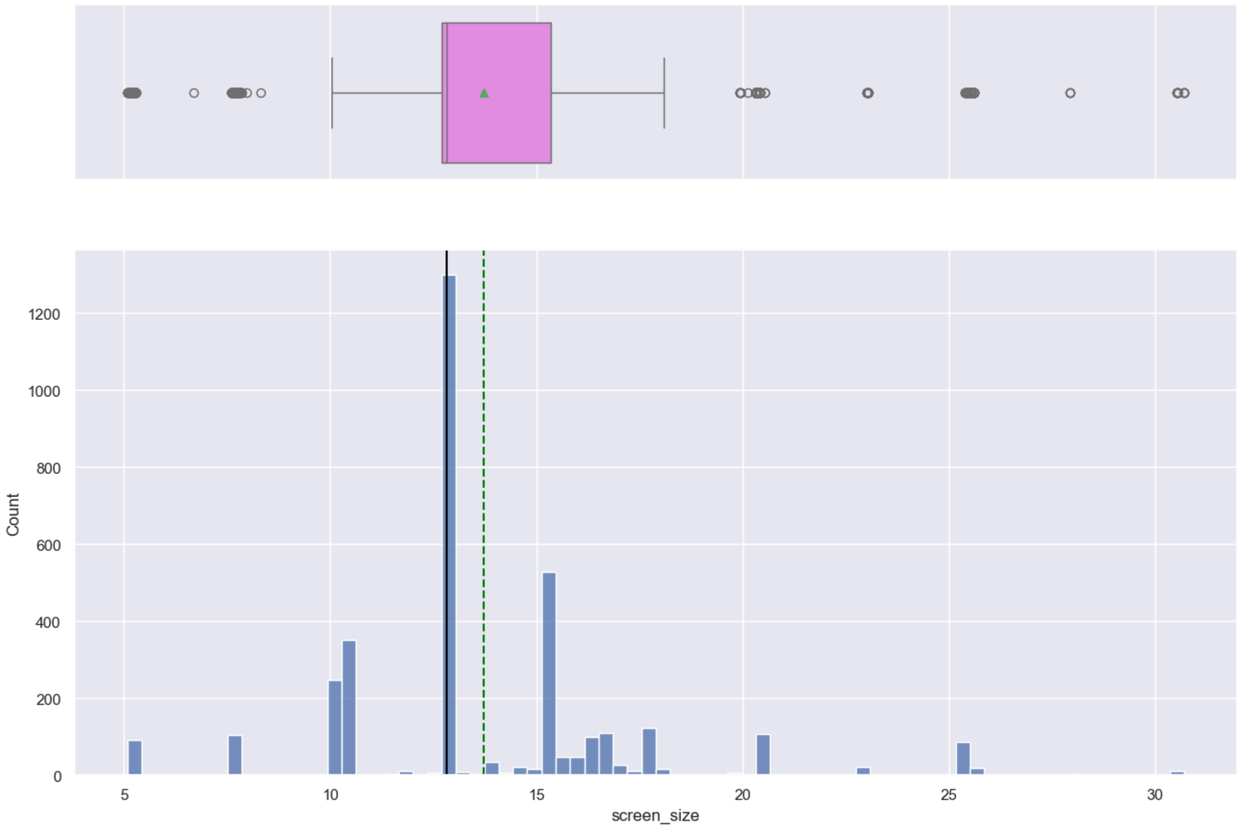
normalized_new_price

In [33]:
```python
histogram_boxplot(df, "normalized_new_price")
```



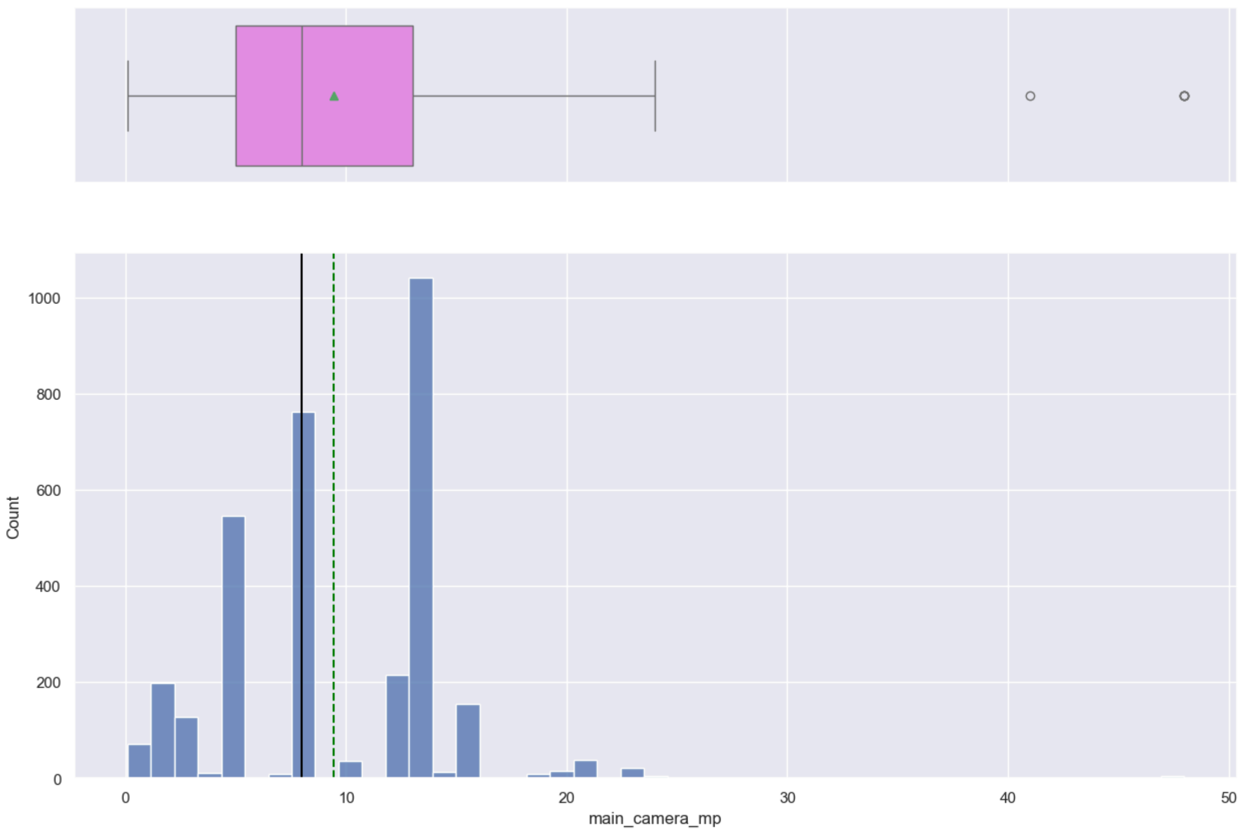
`screen_size`

In [35]:
```
histogram_boxplot(df, "screen_size")
```



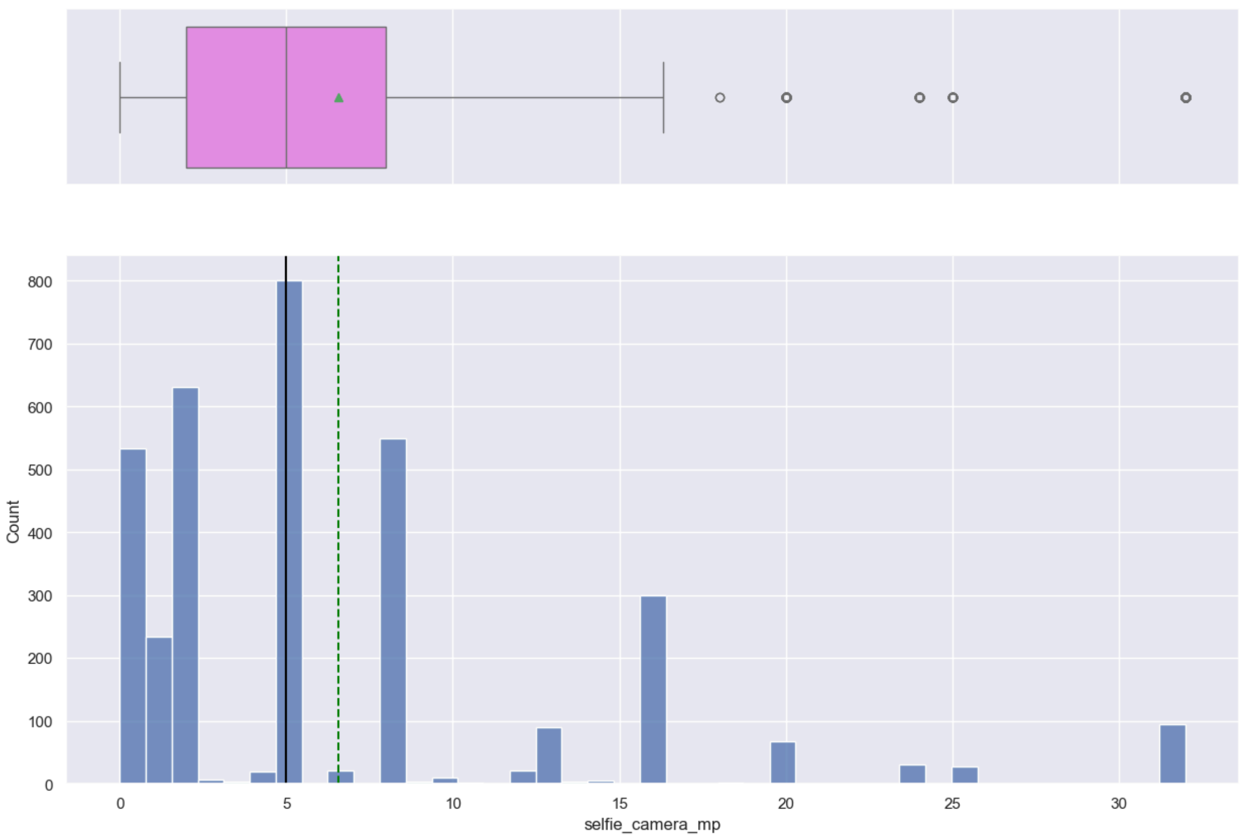
`main_camera_mp`

In [37]:
```
histogram_boxplot(df, "main_camera_mp")
```



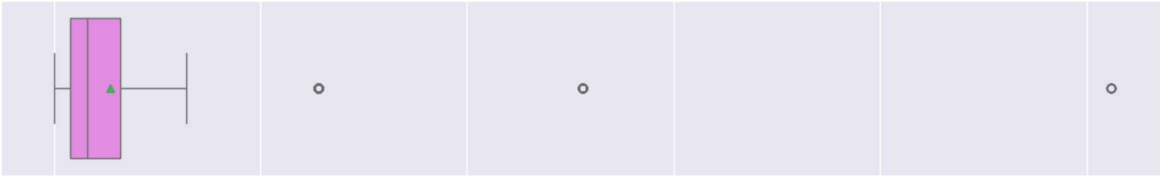
`selfie_camera_mp`

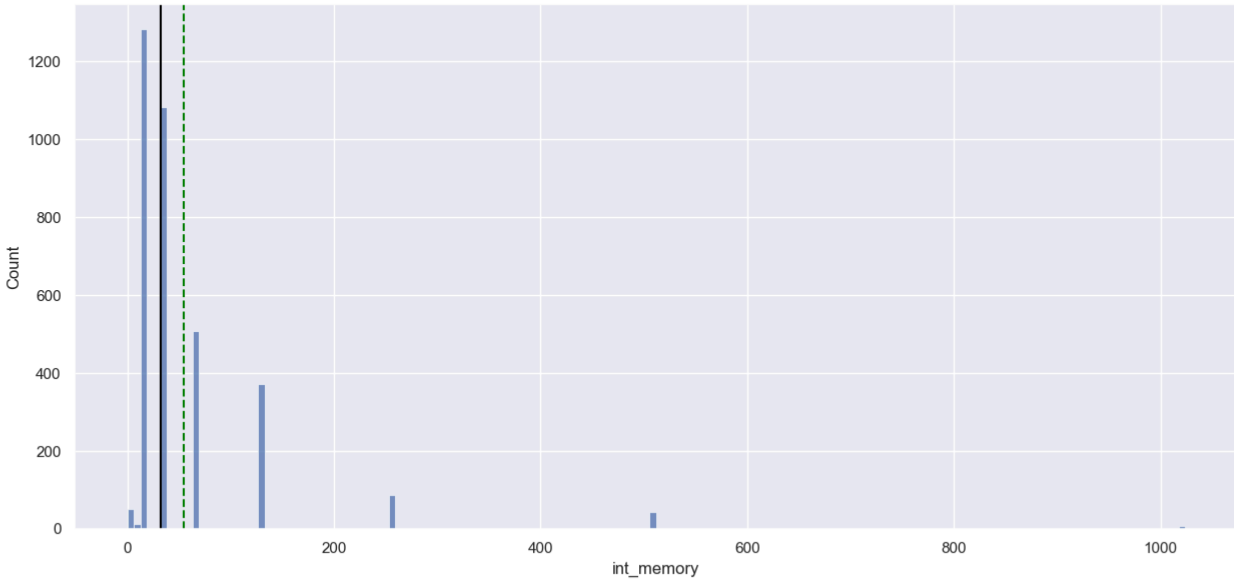
In [39]:
```
histogram_boxplot(df, "selfie_camera_mp")
```



`int_memory`

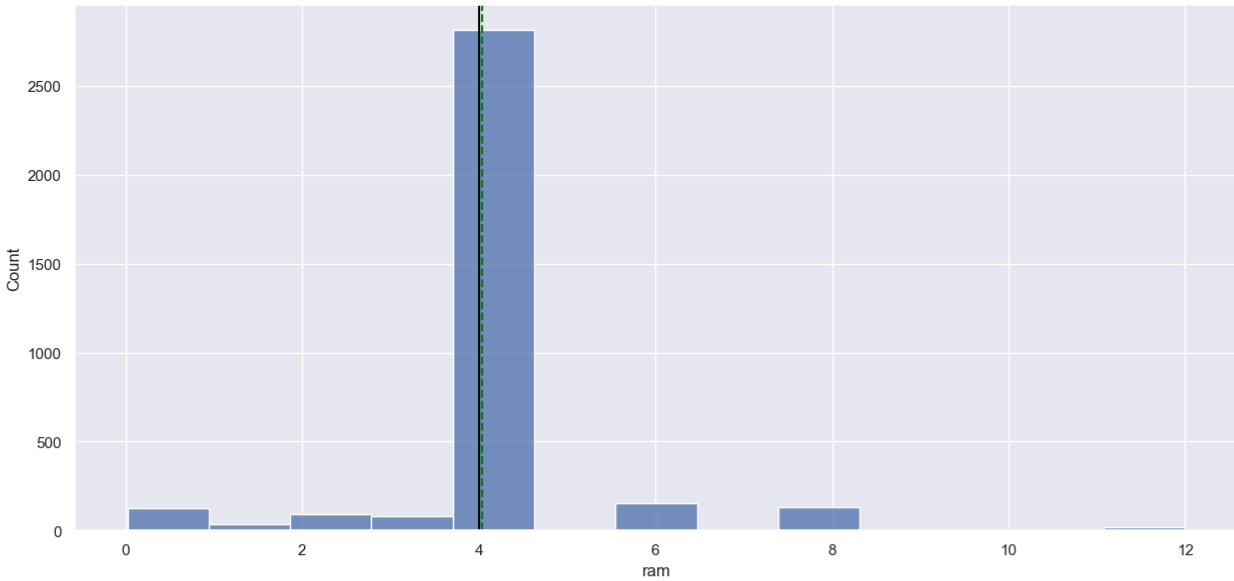
In [41]:
```
histogram_boxplot(df, "int_memory")
```





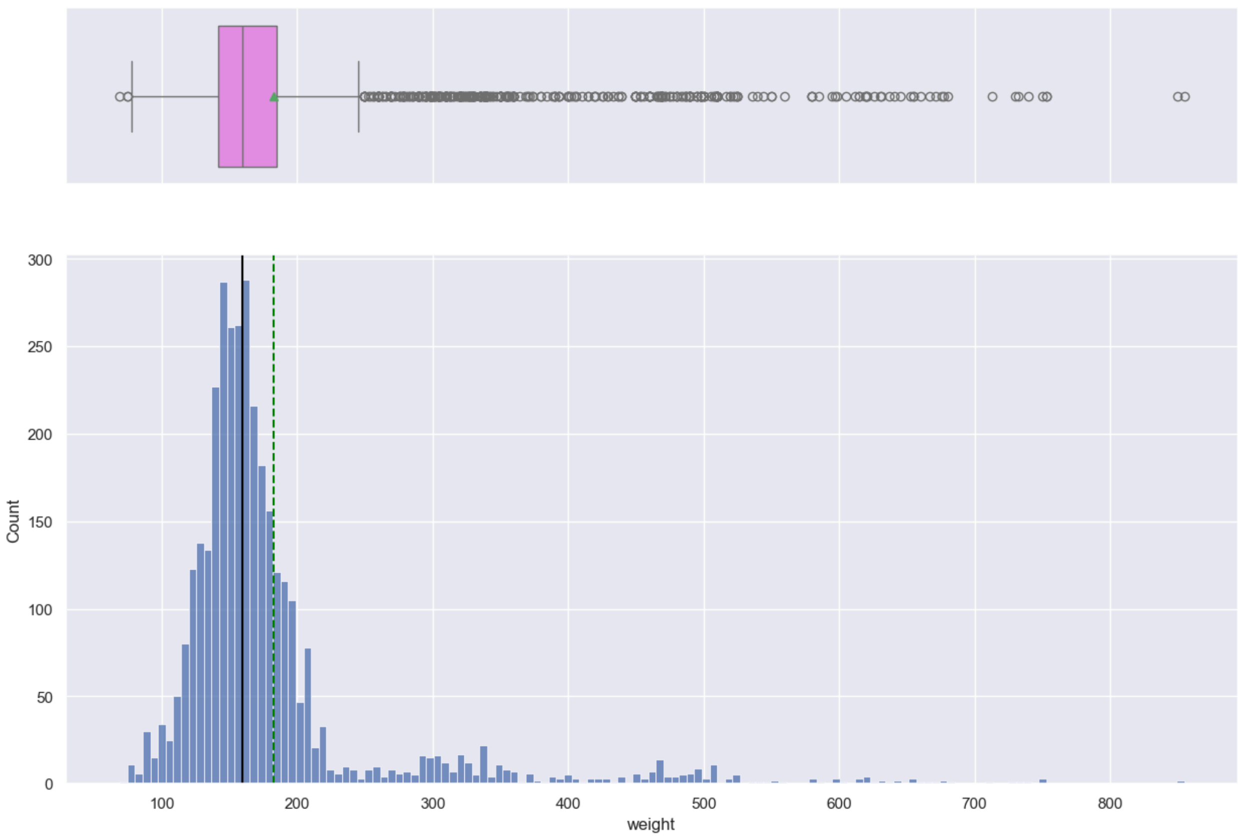
`ram`

In [43]:
```
histogram_boxplot(df, "ram")
```



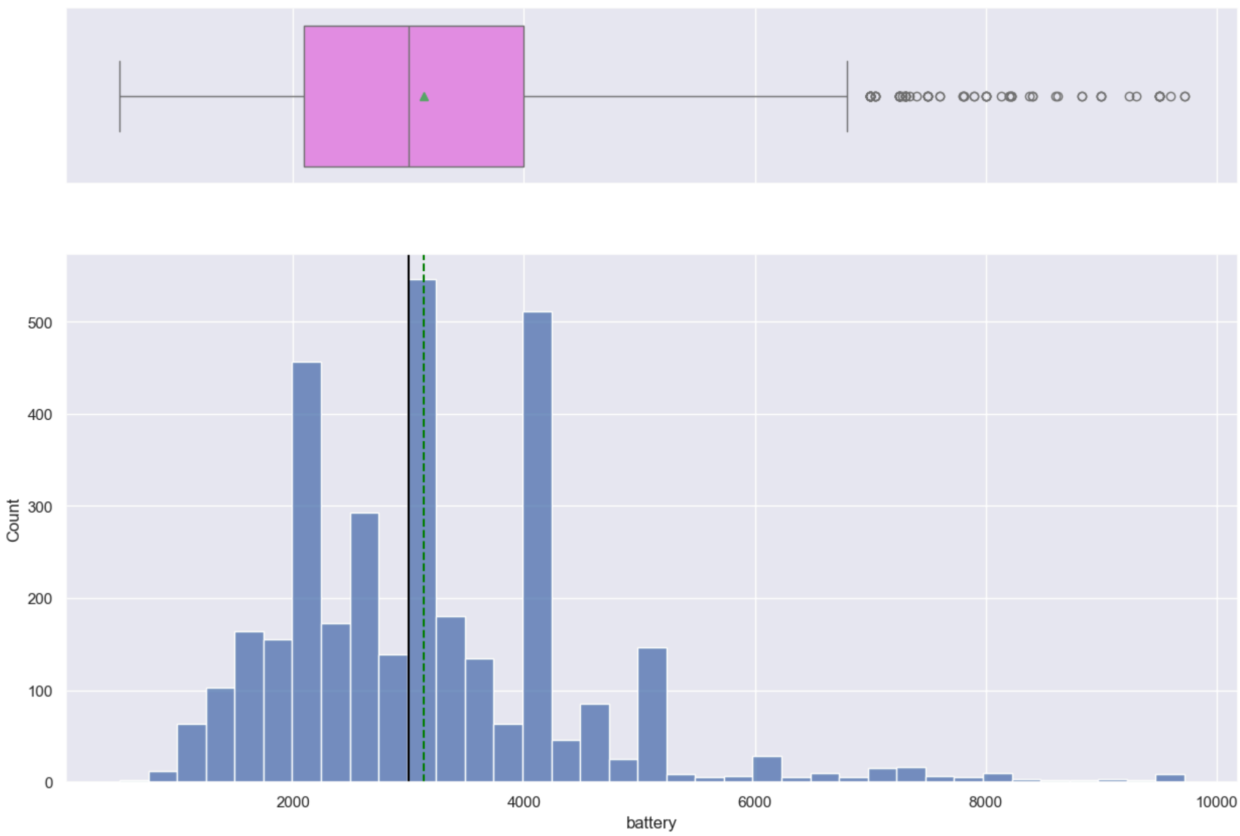
`weight`

In [45]:
```
histogram_boxplot(df, "weight")
```



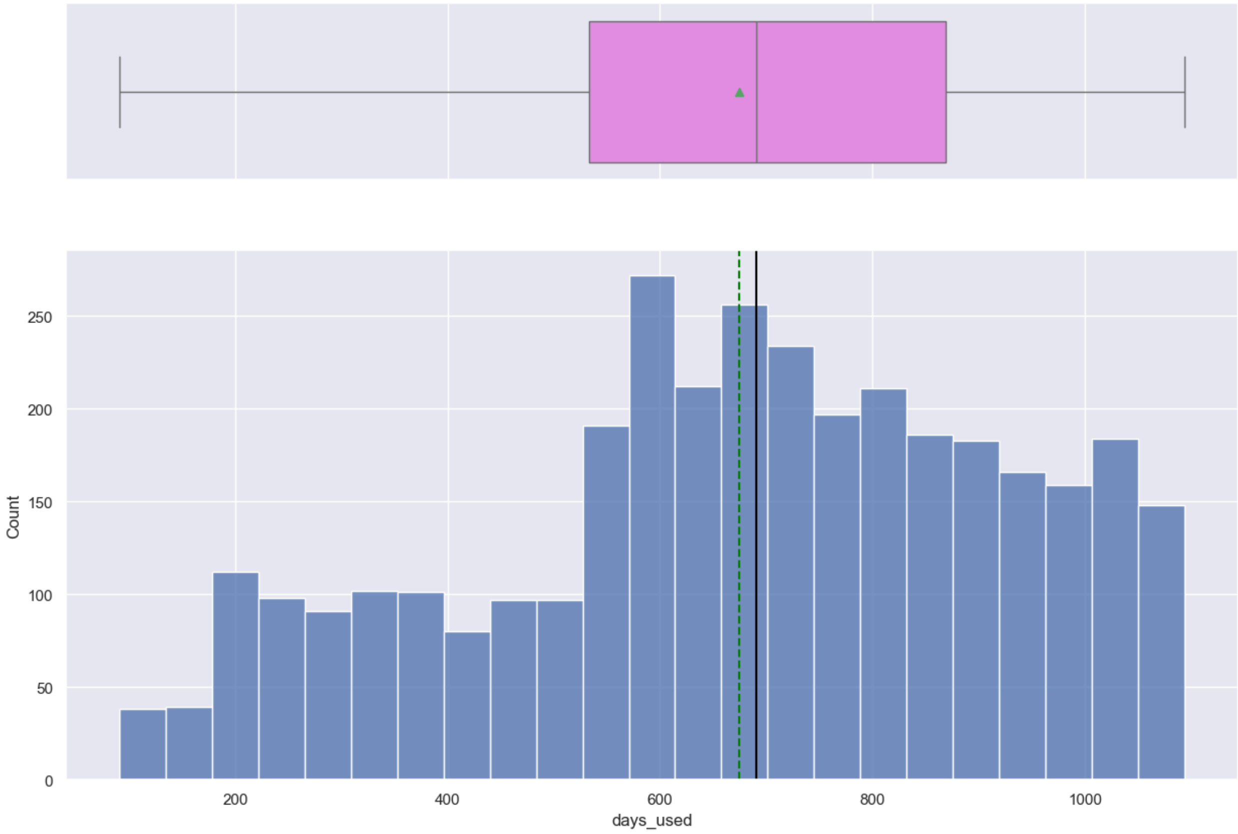
`battery`

```
In [47]: histogram_boxplot(df, "battery")
```



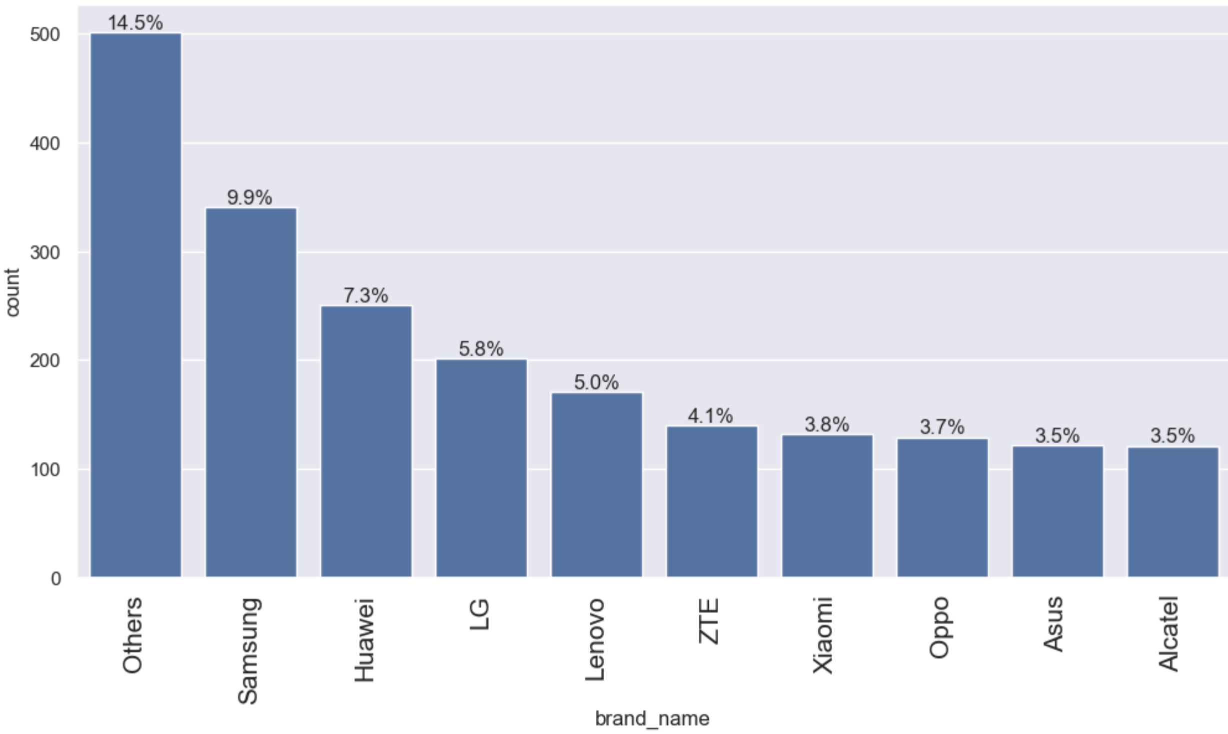
`days_used`

```
In [49]: histogram_boxplot(df, "days_used")
```



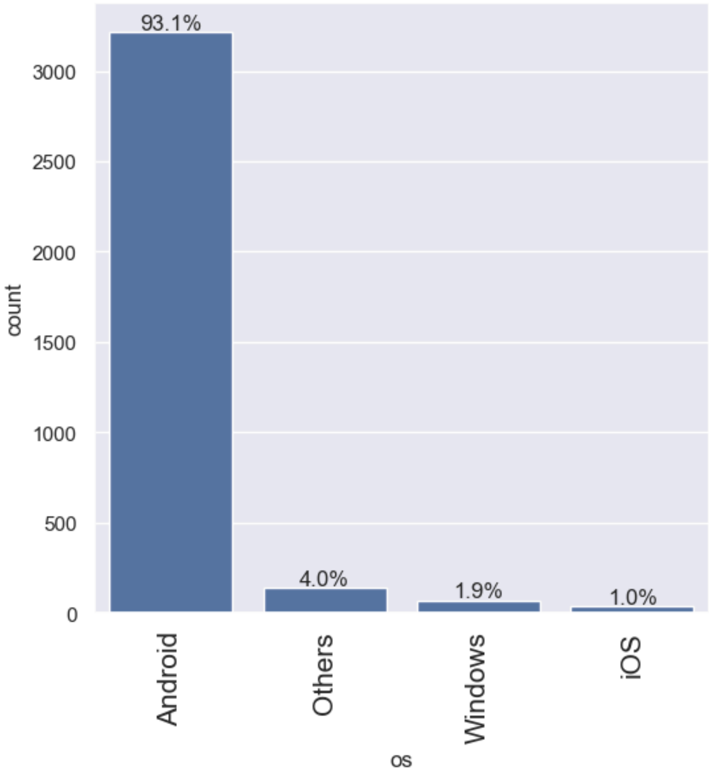
brand_name

```
In [51]: labeled_barplot(df, "brand_name", perc=True, n=10)
```



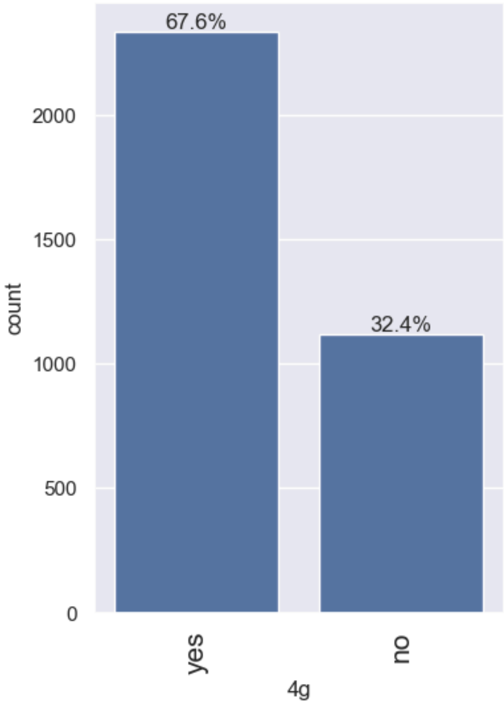
os

```
In [53]: labeled_barplot(df, "os", perc=True)
```



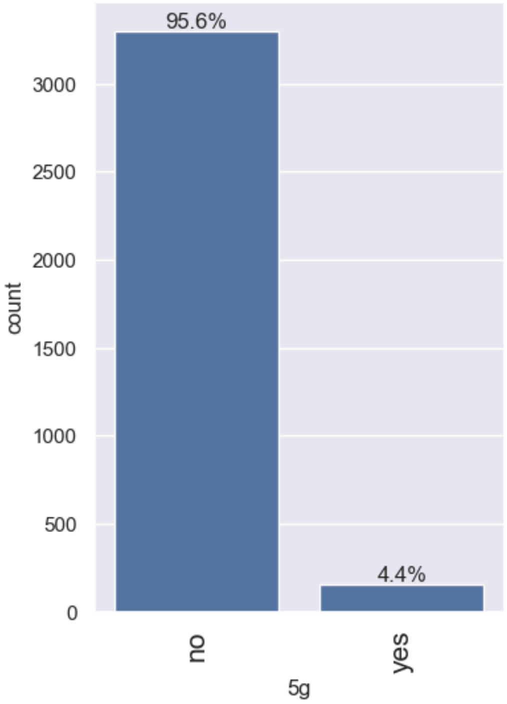
4g

In [55]:
```
labeled_barplot(df, "4g", perc=True)
```



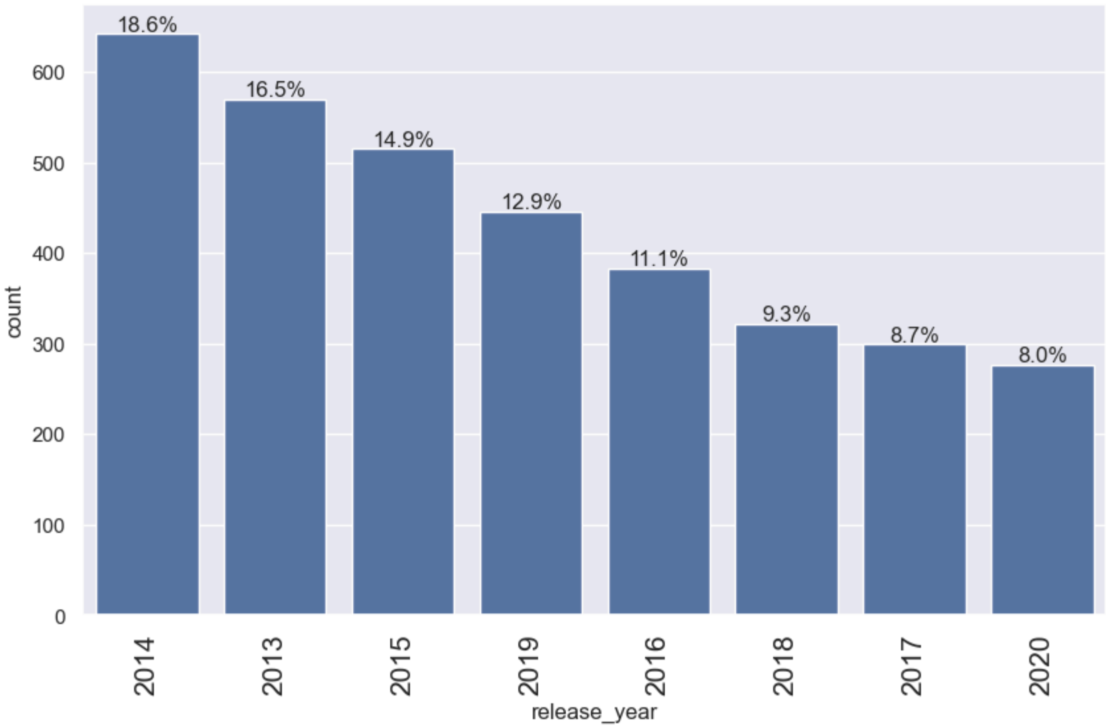
5g

In [57]:
```
labeled_barplot(df, "5g", perc=True)
```



`release_year`

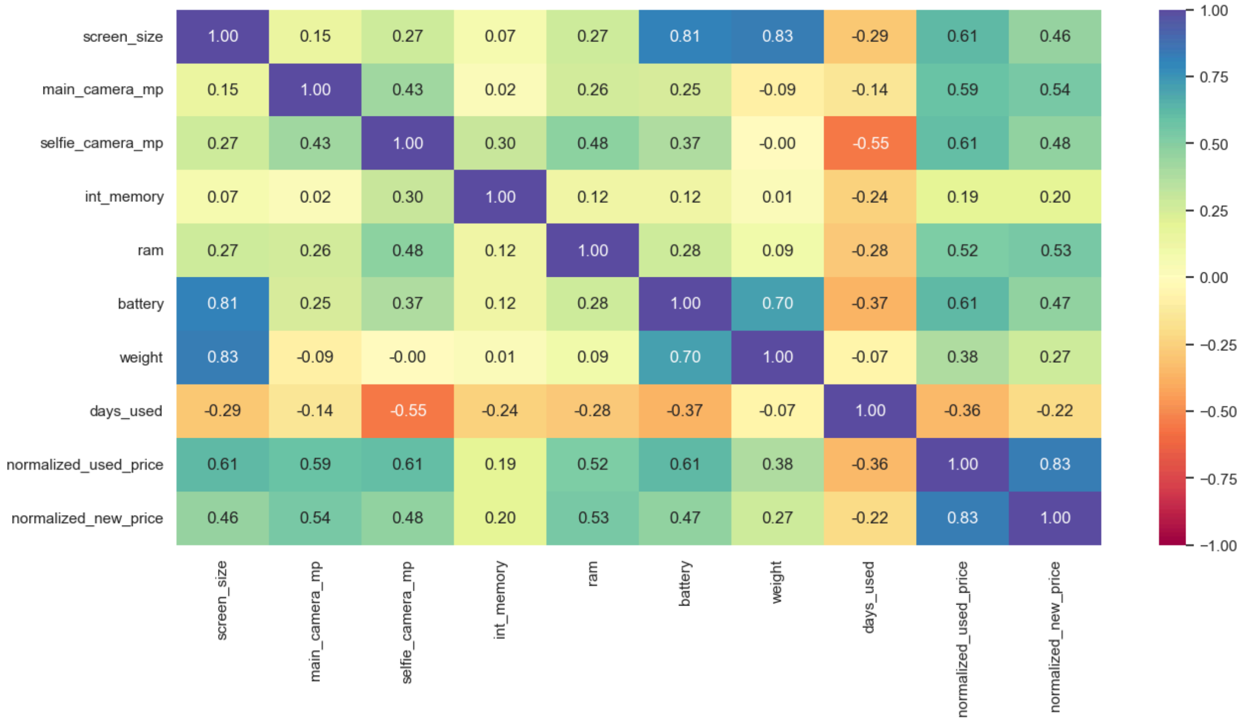
In [59]:
```
labeled_barplot(df, "release_year", perc=True)
```



Bivariate Analysis

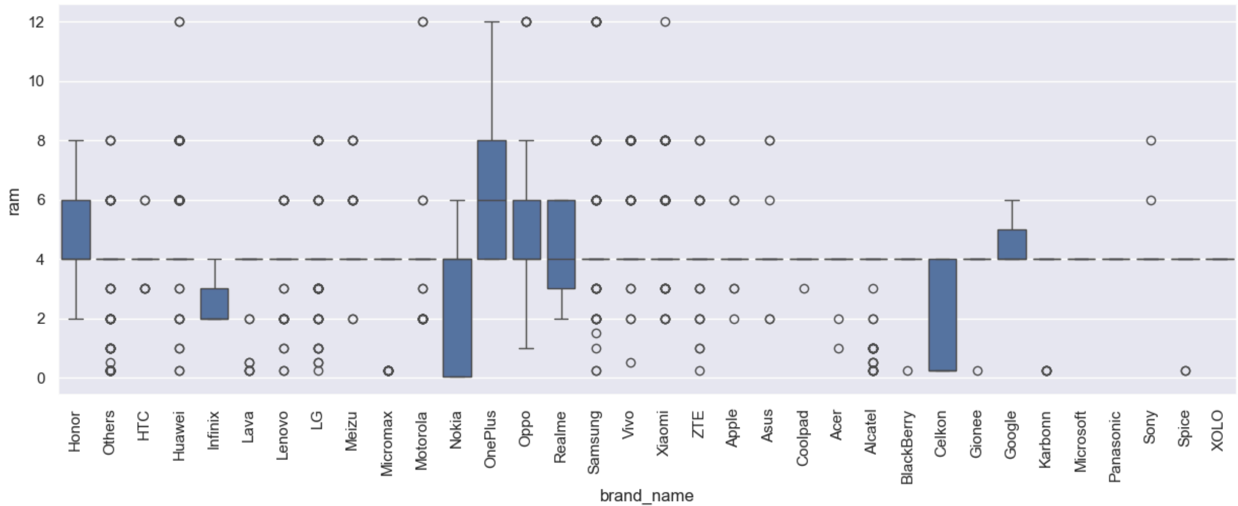
Correlation Check

In [62]:
```
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()
```



The amount of RAM is important for the smooth functioning of a device. Will see how the amount of RAM varies across brands.

In [64]:
```
plt.figure(figsize=(15, 5))
sns.boxplot(data=df, x="brand_name", y="ram")
plt.xticks(rotation=90)
plt.show()
```

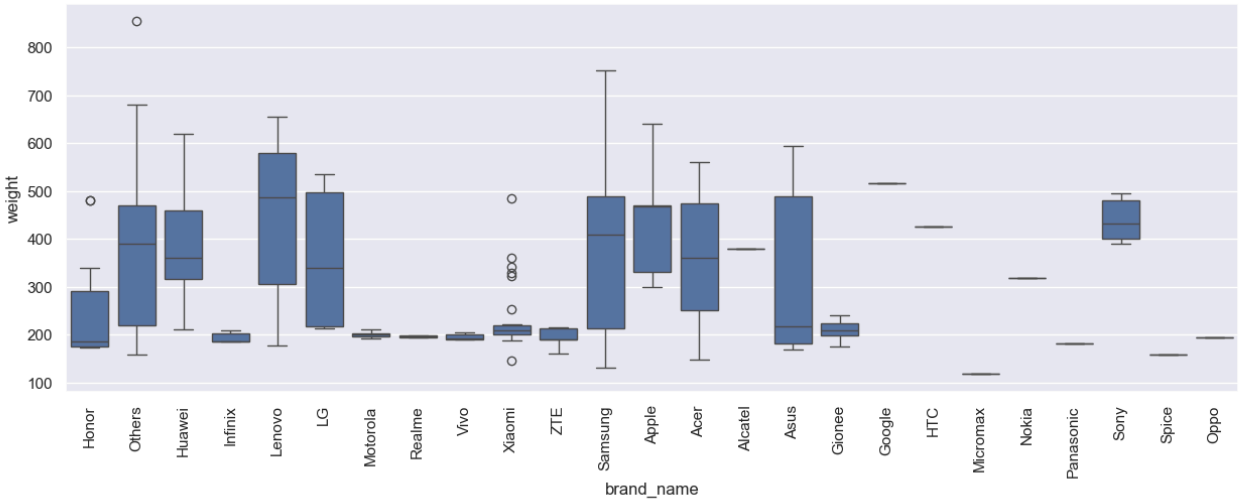


People who travel frequently require devices with large batteries to run through the day. But large battery often increases weight, making it feel uncomfortable in the hands. Will create a new dataframe of only those devices which offer a large battery and analyze.

In [66]:
```
df_large_battery = df[df.battery > 4500]
df_large_battery.shape
```

Out[66]: (341, 15)

In [67]:
```
plt.figure(figsize=(15, 5))
sns.boxplot(data=df_large_battery, x="brand_name", y="weight")
plt.xticks(rotation=90)
plt.show()
```

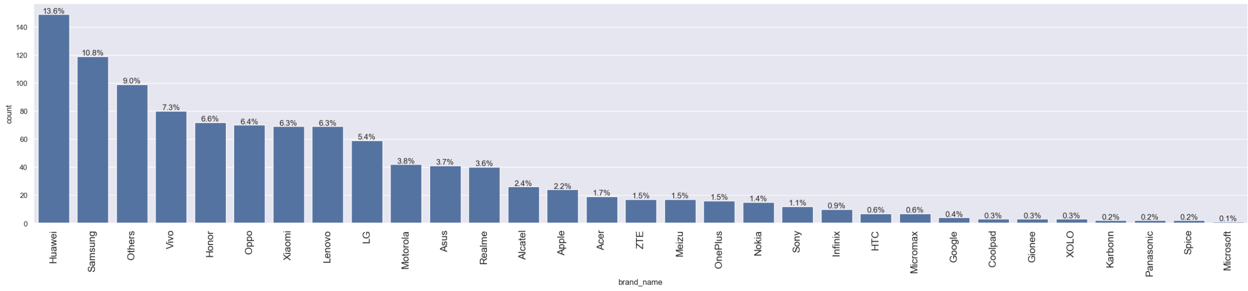


People who buy phones and tablets primarily for entertainment purposes prefer a large screen as they offer a better viewing experience. Will create a new dataframe of only those devices which are suitable for such people and analyze.

In [69]:
```
df_large_screen = df[df.screen_size > 6 * 2.54]
df_large_screen.shape
```

Out[69]: (1099, 15)

In [70]:
```
labeled_barplot(df_large_screen, "brand_name", perc=True)
```

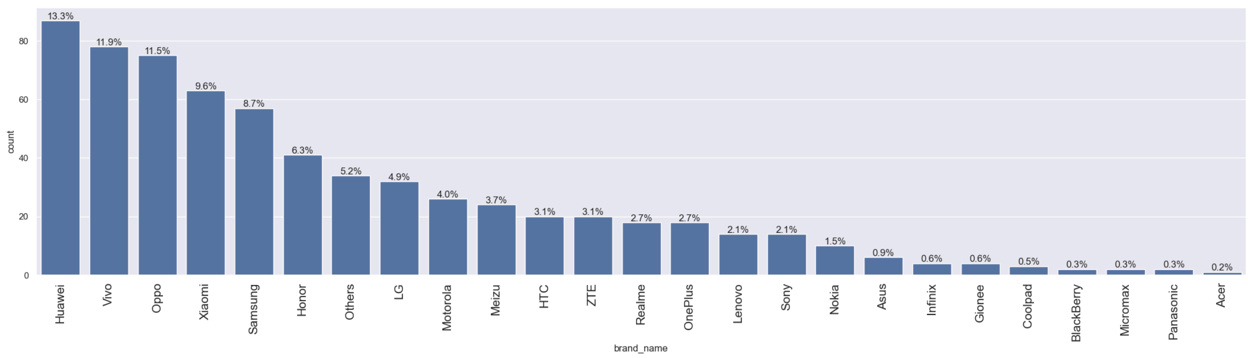


Everyone likes a good camera to capture their favorite moments with loved ones. Some customers specifically look for good front cameras to click cool selfies. Will create a new dataframe of only those devices which are suitable for this customer segment and analyze.

In [72]:
```
df_selfie_camera = df[df.selfie_camera_mp > 8]
df_selfie_camera.shape
```

Out[72]: (655, 15)

In [73]:
```
labeled_barplot(df_selfie_camera, "brand_name", perc=True)
```



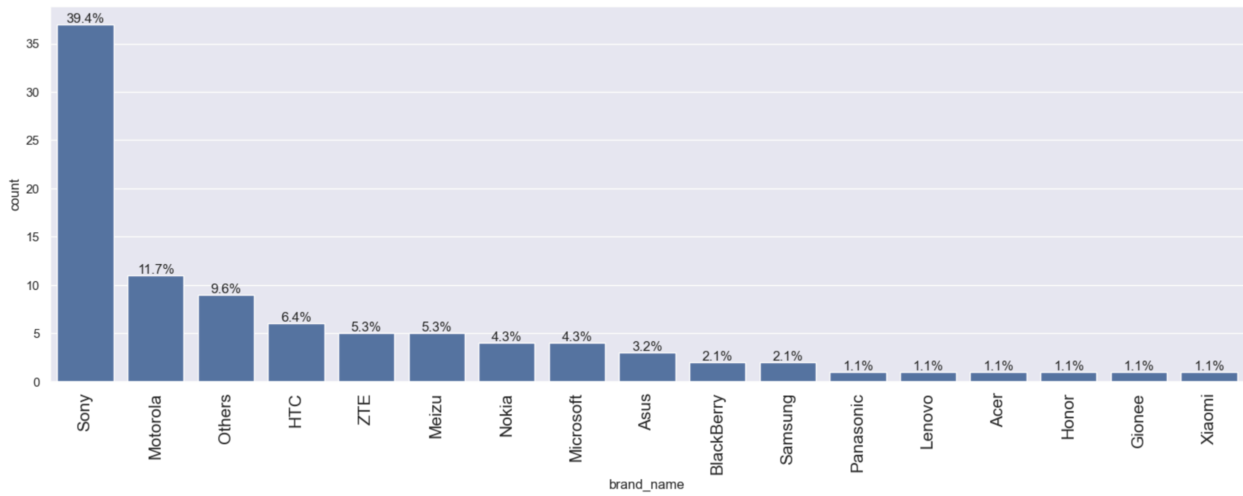
Will do a similar analysis for rear cameras.

- Rear cameras generally have a better resolution than front cameras, so I set the threshold higher for them at 16MP.

In [75]:
```
df_main_camera = df[df.main_camera_mp > 16]
df_main_camera.shape
```

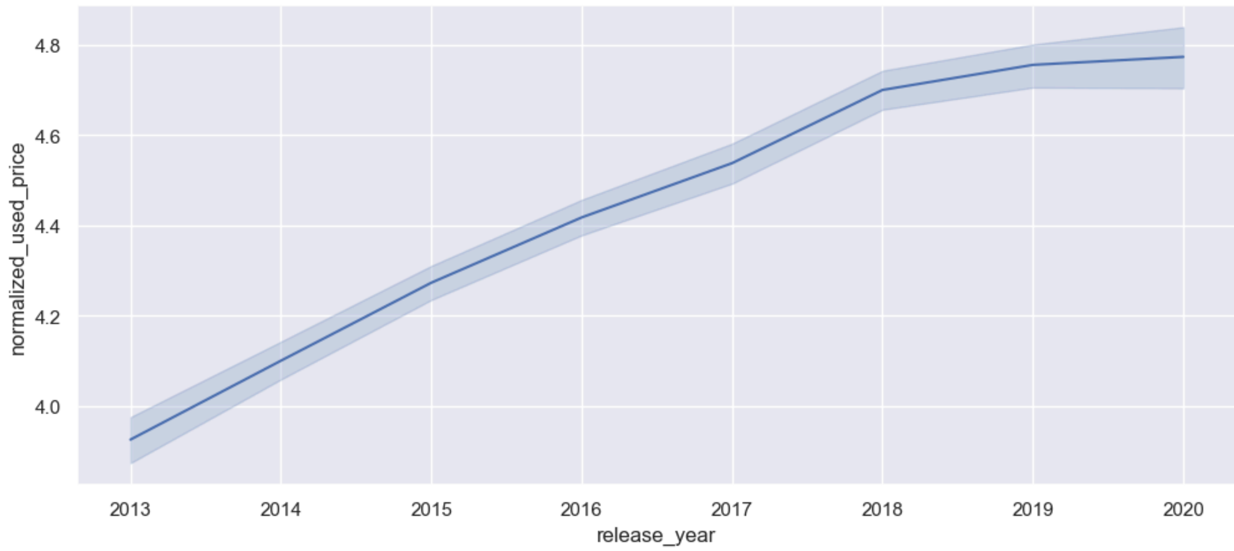
Out[75]: (94, 15)

In [76]:
```
labeled_barplot(df_main_camera, "brand_name", perc=True)
```



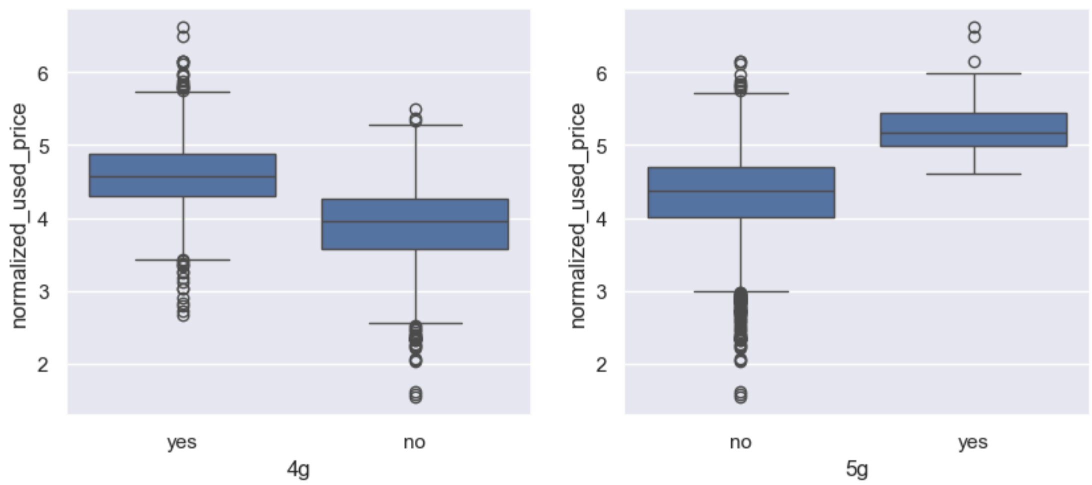
Will see how the price of used devices varies across the years.

In [78]:
```
plt.figure(figsize=(12, 5))
sns.lineplot(data=df, x="release_year", y="normalized_used_price")
plt.show()
```



Will check how the prices vary for used phones and tablets offering 4G and 5G networks.

In [80]:
```
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()
```



Data Preprocessing

Missing Value Imputation

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

In [83]:
```
# create a copy of the data
df1 = df.copy()
```

In [84]:
```
# checking for missing values
df1.isnull().sum()
```

Out[84]:
```
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 [85]:
```
cols_impute = [
    "main_camera_mp",
    "selfie_camera_mp",
    "int_memory",
    "ram",
    "battery",
    "weight",
]

# Impute missing values by grouping on release_year and brand_name
for col in cols_impute:
    df1[col] = df1[col].fillna(
        value=df1.groupby(["release_year", "brand_name"])[col].transform("median")
    )

# Check missing values after imputation
df1.isnull().sum()
```

Out[85]:
```
brand_name                 0
os                         0
screen_size                0
4g                         0
5g                         0
main_camera_mp           179
selfie_camera_mp           2
int_memory                 0
ram                        0
battery                    6
weight                     7
release_year               0
days_used                  0
normalized_used_price      0
normalized_new_price       0
dtype: int64
```

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

In [87]:
```
cols_impute = [
    "main_camera_mp",
    "selfie_camera_mp",
    "battery",
    "weight",
]

# Impute missing values by grouping on brand_name
for col in cols_impute:
    df1[col] = df1[col].fillna(
        value=df1.groupby(["brand_name"])[col].transform("median")
    )

# Check missing values after this imputation
df1.isnull().sum()
```

```
Out[87]: 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
```

- I will fill the remaining missing values in the `main_camera_mp` column by the column median.

```
In [89]: # Impute remaining missing values in main_camera_mp using the column median
df1["main_camera_mp"] = df1["main_camera_mp"].fillna(df1["main_camera_mp"].median())

# Check missing values after final imputation
df1.isnull().sum()  # Fill the blank to confirm all missing values are handled
```

```
Out[89]: 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
```

Feature Engineering

- Create a new column `years_since_release` from the `release_year` column.
- Consider the year of data collection, 2021, as the baseline.
- Will drop the `release_year` column.

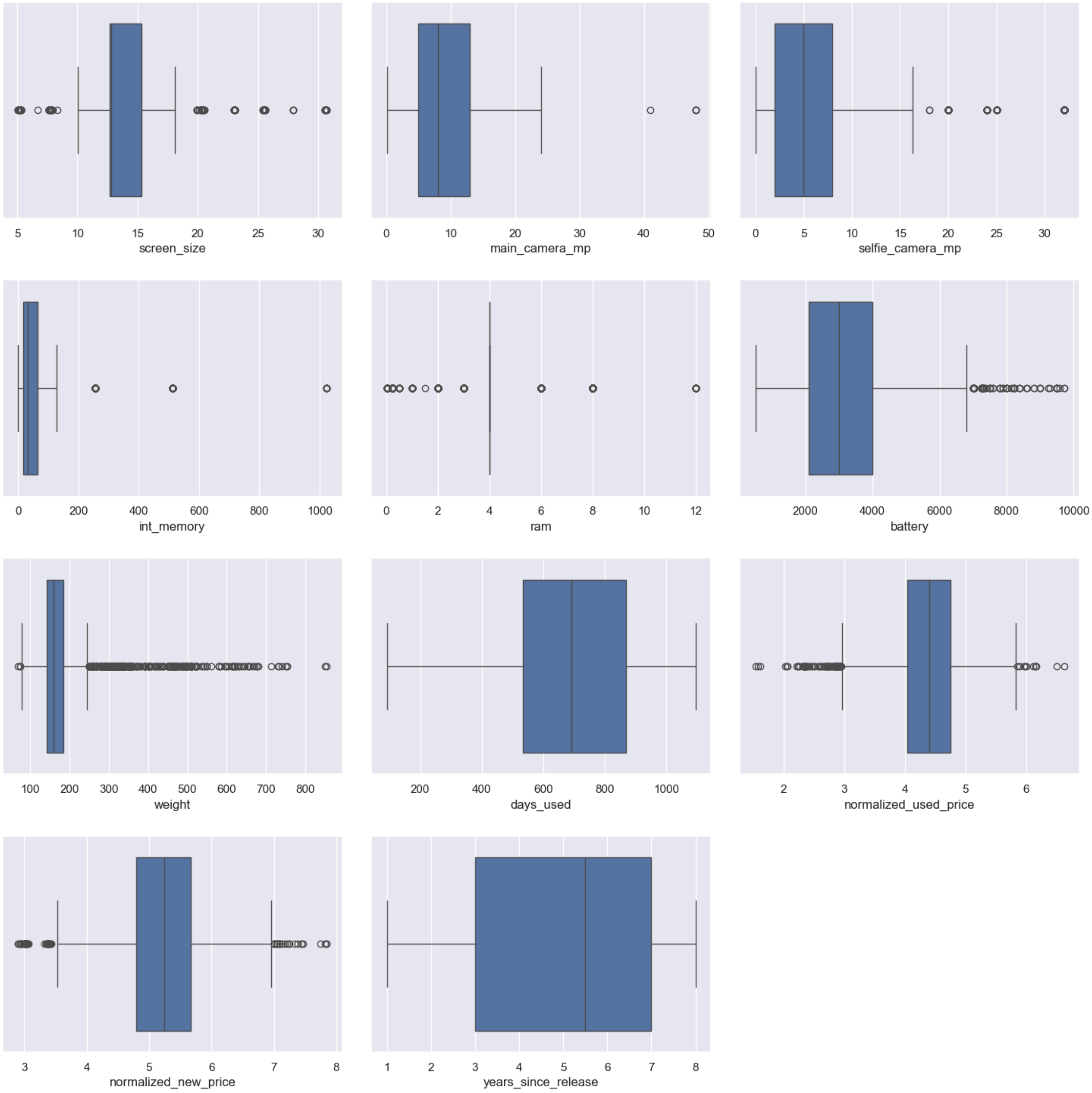
```
In [92]: df1["years_since_release"] = 2021 - df1["release_year"]
df1.drop("release_year", axis=1, inplace=True)
df1["years_since_release"].describe()
```

```
Out[92]: count    3454.000000
mean        5.034742
std         2.298455
min         1.000000
25%         3.000000
50%         5.500000
75%         7.000000
max         8.000000
Name: years_since_release, dtype: float64
```

Outlier Check

- Will check for outliers in the data.

```
In [95]: # 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()
```



Data Preparation for modeling

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

In [98]:
```
# Define dependent (y) and independent (X) variables
X = df1.drop(columns=["normalized_used_price"])  # Drop the target variable
y = df1["normalized_used_price"]  # Target variable

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

In [99]:
```
# Add an intercept to the data
X = sm.add_constant(X)
```

In [100…
```
# Create dummy variables for categorical features
X = pd.get_dummies(
    X,
    columns=X.select_dtypes(include=["object", "category"]).columns.tolist(),
    drop_first=True,
)

X.head()  # Check the transformed dataset
```

Out[100…

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	days_used	normalized_new_price	...
0	1.0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	127	4.715100	...
1	1.0	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	325	5.519018	...
2	1.0	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	162	5.884631	...
3	1.0	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	345	5.630961	...
4	1.0	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	293	4.947837	...

5 rows × 49 columns

In [101…
```
# 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=42)
```

In [102…
```
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 [104…
```
# Check the data types of x_train and y_train
print(x_train.dtypes)
print(y_train.dtypes)
```

```
const                      float64
screen_size                float64
main_camera_mp             float64
selfie_camera_mp           float64
int_memory                 float64
ram                        float64
battery                    float64
weight                     float64
days_used                    int64
normalized_new_price       float64
years_since_release          int64
brand_name_Alcatel            bool
brand_name_Apple              bool
brand_name_Asus               bool
brand_name_BlackBerry         bool
brand_name_Celkon             bool
brand_name_Coolpad            bool
brand_name_Gionee             bool
brand_name_Google             bool
brand_name_HTC                bool
brand_name_Honor              bool
brand_name_Huawei             bool
brand_name_Infinix            bool
brand_name_Karbonn            bool
brand_name_LG                 bool
brand_name_Lava               bool
brand_name_Lenovo             bool
brand_name_Meizu              bool
brand_name_Micromax           bool
brand_name_Microsoft          bool
brand_name_Motorola           bool
brand_name_Nokia              bool
brand_name_OnePlus            bool
brand_name_Oppo               bool
brand_name_Others             bool
brand_name_Panasonic          bool
brand_name_Realme             bool
brand_name_Samsung            bool
brand_name_Sony               bool
brand_name_Spice              bool
brand_name_Vivo               bool
brand_name_XOLO               bool
brand_name_Xiaomi             bool
brand_name_ZTE                bool
os_Others                     bool
os_Windows                    bool
os_iOS                        bool
4g_yes                        bool
5g_yes                        bool
dtype: object
float64
```

In [105…

```python
# Convert all boolean columns to int
x_train = x_train.astype({col: 'int' for col in x_train.select_dtypes(include=['bool']).columns})
x_test = x_test.astype({col: 'int' for col in x_test.select_dtypes(include=['bool']).columns})
```

In [106…

```python
# Build the OLS linear regression model
olsmodel1 = sm.OLS(y_train, x_train).fit()

# Print the summary of the model
print(olsmodel1.summary())
```

```
                            OLS Regression Results
===============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.849
Model:                               OLS   Adj. R-squared:                  0.846
Method:                    Least Squares   F-statistic:                     277.1
Date:                   Thu, 23 Jan 2025   Prob (F-statistic):               0.00
Time:                           16:50:45   Log-Likelihood:                 125.15
No. Observations:                   2417   AIC:                            -152.3
Df Residuals:                       2368   BIC:                             131.4
Df Model:                             48
Covariance Type:               nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                     1.4172      0.072     19.677      0.000       1.276       1.558
screen_size               0.0295      0.004      8.352      0.000       0.023       0.036
main_camera_mp            0.0232      0.002     14.892      0.000       0.020       0.026
selfie_camera_mp          0.0116      0.001      9.924      0.000       0.009       0.014
int_memory                0.0002   6.76e-05      2.779      0.005    5.53e-05       0.000
ram                       0.0305      0.005      5.797      0.000       0.020       0.041
battery               -1.665e-05   7.35e-06     -2.266      0.024   -3.11e-05   -2.24e-06
weight                    0.0008      0.000      5.946      0.000       0.001       0.001
days_used              3.376e-05   3.07e-05      1.101      0.271   -2.64e-05    9.39e-05
normalized_new_price      0.4104      0.012     33.494      0.000       0.386       0.434
years_since_release      -0.0255      0.005     -5.589      0.000      -0.034      -0.017
brand_name_Alcatel       -0.0804      0.050     -1.618      0.106      -0.178       0.017
brand_name_Apple         -0.0438      0.148     -0.297      0.767      -0.333       0.246
brand_name_Asus           0.0068      0.049      0.138      0.890      -0.090       0.103
brand_name_BlackBerry     0.0312      0.072      0.434      0.664      -0.110       0.172
brand_name_Celkon        -0.2372      0.068     -3.485      0.001      -0.371      -0.104
brand_name_Coolpad       -0.0308      0.071     -0.434      0.664      -0.170       0.108
brand_name_Gionee        -0.0130      0.059     -0.221      0.825      -0.128       0.102
brand_name_Google        -0.1192      0.083     -1.442      0.149      -0.281       0.043
brand_name_HTC           -0.0403      0.050     -0.805      0.421      -0.138       0.058
brand_name_Honor         -0.0478      0.051     -0.941      0.347      -0.147       0.052
brand_name_Huawei        -0.0599      0.046     -1.299      0.194      -0.150       0.030
brand_name_Infinix        0.1338      0.113      1.179      0.238      -0.089       0.356
brand_name_Karbonn       -0.0592      0.068     -0.867      0.386      -0.193       0.075
brand_name_LG            -0.0608      0.047     -1.299      0.194      -0.152       0.031
brand_name_Lava          -0.0230      0.063     -0.364      0.716      -0.147       0.101
brand_name_Lenovo        -0.0364      0.047     -0.771      0.441      -0.129       0.056
brand_name_Meizu         -0.0860      0.056     -1.530      0.126      -0.196       0.024
brand_name_Micromax      -0.0645      0.049     -1.315      0.189      -0.161       0.032
brand_name_Microsoft      0.0749      0.082      0.916      0.360      -0.085       0.235
brand_name_Motorola      -0.0686      0.051     -1.349      0.177      -0.168       0.031
brand_name_Nokia          0.0380      0.052      0.724      0.469      -0.065       0.141
brand_name_OnePlus       -0.0384      0.073     -0.523      0.601      -0.182       0.105
brand_name_Oppo          -0.0294      0.049     -0.599      0.549      -0.126       0.067
brand_name_Others        -0.0679      0.044     -1.556      0.120      -0.154       0.018
brand_name_Panasonic     -0.0427      0.062     -0.690      0.490      -0.164       0.078
brand_name_Realme        -0.0359      0.063     -0.568      0.570      -0.160       0.088
brand_name_Samsung       -0.0617      0.045     -1.376      0.169      -0.150       0.026
brand_name_Sony          -0.0756      0.053     -1.428      0.153      -0.179       0.028
brand_name_Spice         -0.0356      0.068     -0.520      0.603      -0.170       0.099
brand_name_Vivo          -0.0644      0.050     -1.277      0.202      -0.163       0.034
brand_name_XOLO          -0.0781      0.057     -1.362      0.173      -0.191       0.034
brand_name_Xiaomi         0.0325      0.050      0.655      0.513      -0.065       0.130
brand_name_ZTE           -0.0460      0.048     -0.952      0.341      -0.141       0.049
os_Others                -0.0604      0.033     -1.856      0.064      -0.124       0.003
os_Windows               -0.0374      0.043     -0.861      0.389      -0.122       0.048
os_iOS                   -0.0141      0.148     -0.095      0.924      -0.304       0.276
4g_yes                    0.0406      0.016      2.514      0.012       0.009       0.072
5g_yes                   -0.0916      0.032     -2.846      0.004      -0.155      -0.029
==============================================================================
Omnibus:                      234.465   Durbin-Watson:                   1.994
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              630.705
Skew:                          -0.536   Prob(JB):                    1.11e-137
Kurtosis:                       5.262   Cond. No.                     1.85e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.85e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Model Performance Check

Will check the performance of the model using different metrics.

- I will be using metric functions defined in sklearn for RMSE, MAE, and R^2.
- Also will define a function to calculate MAPE and adjusted R^2.
- And will create a function which will print out all the above metrics in one go.

In [109…

```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))
```

```python
# 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 [110…

```python
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel1_train_perf = model_performance_regression(olsmodel1, x_train, y_train)
olsmodel1_train_perf
```

Training Performance

Out[110…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.229761	0.178533	0.848887	0.845758	4.293664

In [111…

```python
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmodel1_test_perf = model_performance_regression(olsmodel1, x_test, y_test)
olsmodel1_test_perf
```

Test Performance

Out[111…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.239062	0.188692	0.832176	0.823844	4.513288

Checking Linear Regression Assumptions

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

- I will test for multicollinearity using VIF.
- **General Rule of thumb**:
  - If VIF is 1 then there is no correlation between the kth predictor and the remaining predictor variables.
  - If VIF exceeds 5 or is close to exceeding 5, I say there is moderate multicollinearity.

- If VIF is 10 or exceeding 10, it shows signs of high multicollinearity.

Will define a function to check VIF.

In [117…

```
#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 [118…

```
checking_vif(x_train)
```

Out[118…

	feature	VIF
0	const	232.676933
1	screen_size	8.262147
2	main_camera_mp	2.418167
3	selfie_camera_mp	2.872720
4	int_memory	1.363390
5	ram	2.283507
6	battery	4.066126
7	weight	6.417982
8	days_used	2.580338
9	normalized_new_price	3.218722
10	years_since_release	4.878548
11	brand_name_Alcatel	3.458576
12	brand_name_Apple	11.195090
13	brand_name_Asus	3.652764
14	brand_name_BlackBerry	1.623330
15	brand_name_Celkon	1.873667
16	brand_name_Coolpad	1.575445
17	brand_name_Gionee	2.076683
18	brand_name_Google	1.388001
19	brand_name_HTC	3.460299
20	brand_name_Honor	3.559332
21	brand_name_Huawei	6.395779
22	brand_name_Infinix	1.191800
23	brand_name_Karbonn	1.628707
24	brand_name_LG	5.354573
25	brand_name_Lava	1.826002
26	brand_name_Lenovo	4.705199
27	brand_name_Meizu	2.417090
28	brand_name_Micromax	3.779514
29	brand_name_Microsoft	2.092692
30	brand_name_Motorola	3.487485
31	brand_name_Nokia	3.755089
32	brand_name_OnePlus	1.586224
33	brand_name_Oppo	4.285316
34	brand_name_Others	10.833481
35	brand_name_Panasonic	1.890592
36	brand_name_Realme	1.978470
37	brand_name_Samsung	8.013571
38	brand_name_Sony	2.891092
39	brand_name_Spice	1.638484
40	brand_name_Vivo	3.734375
41	brand_name_XOLO	2.163880
42	brand_name_Xiaomi	4.082217
43	brand_name_ZTE	4.343056
44	os_Others	1.885820
45	os_Windows	1.742679
46	os_iOS	10.037221
47	4g_yes	2.543525
48	5g_yes	1.808205

Removing Multicollinearity (if needed)

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.

Will define a function that will help do this.

In [121…
```
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 [122…
```
# Define the columns with high VIF
col_list = [
    "brand_name_Apple",
    "os_iOS",
    "brand_name_Others",
    "screen_size",
    "brand_name_Samsung",
    "weight",
    "brand_name_Huawei",
    "brand_name_LG",
]

# Evaluate the effect of dropping each column
res = treating_multicollinearity(x_train, y_train, col_list)
print(res)
```

```
                 col  Adj. R-squared after_dropping col  \
0             os_iOS                           0.845888   
1   brand_name_Apple                           0.845883   
2  brand_name_Huawei                           0.845779   
3      brand_name_LG                           0.845779   
4 brand_name_Samsung                           0.845765   
5  brand_name_Others                           0.845731   
6             weight                           0.843587   
7        screen_size                           0.841348   

  RMSE after dropping col  
0                0.232077  
1                0.232081  
2                0.232159  
3                0.232159  
4                0.232169  
5                0.232195  
6                0.233803  
7                0.235470  
```

In [123…
```
# Drop os_iOS
col_to_drop = "os_iOS"
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)]

# Recheck VIF after dropping os_iOS
```

```
vif = checking_vif(x_train2)
print(f"VIF after dropping {col_to_drop}:\n", vif)
```

```
VIF after dropping os_iOS:
                   feature         VIF
0                    const  231.265323
1              screen_size    8.175755
2           main_camera_mp    2.415490
3         selfie_camera_mp    2.860115
4               int_memory    1.363372
5                      ram    2.267806
6                  battery    4.061755
7                   weight    6.392276
8                days_used    2.579961
9     normalized_new_price    3.218565
10     years_since_release    4.876385
11      brand_name_Alcatel    3.458531
12        brand_name_Apple    2.002678
13         brand_name_Asus    3.652735
14   brand_name_BlackBerry    1.621180
15       brand_name_Celkon    1.873654
16      brand_name_Coolpad    1.575340
17       brand_name_Gionee    2.076508
18       brand_name_Google    1.387901
19          brand_name_HTC    3.459827
20        brand_name_Honor    3.559082
21       brand_name_Huawei    6.395463
22      brand_name_Infinix    1.191800
23      brand_name_Karbonn    1.628284
24           brand_name_LG    5.354530
25         brand_name_Lava    1.825517
26       brand_name_Lenovo    4.705179
27        brand_name_Meizu    2.417090
28     brand_name_Micromax    3.779217
29    brand_name_Microsoft    2.092465
30     brand_name_Motorola    3.487469
31        brand_name_Nokia    3.752726
32      brand_name_OnePlus    1.586107
33         brand_name_Oppo    4.285110
34       brand_name_Others   10.833310
35    brand_name_Panasonic    1.890542
36       brand_name_Realme    1.978364
37      brand_name_Samsung    8.013146
38         brand_name_Sony    2.891087
39        brand_name_Spice    1.637627
40         brand_name_Vivo    3.734253
41         brand_name_XOLO    2.163619
42       brand_name_Xiaomi    4.082163
43          brand_name_ZTE    4.342740
44               os_Others    1.770743
45              os_Windows    1.740883
46                  4g_yes    2.541759
47                  5g_yes    1.800836
```

In [124…

```
# Drop brand_name_Others
col_to_drop = "brand_name_Others"
x_train2 = x_train2.loc[:, ~x_train2.columns.str.startswith(col_to_drop)]
x_test2 = x_test2.loc[:, ~x_test2.columns.str.startswith(col_to_drop)]

# Recheck VIF after dropping brand_name_Others
vif = checking_vif(x_train2)
print(f"VIF after dropping {col_to_drop}:\n", vif)
```

```
VIF after dropping brand_name_Others:
                   feature         VIF
0                    const  149.761466
1              screen_size    8.162450
2           main_camera_mp    2.414127
3         selfie_camera_mp    2.859951
4               int_memory    1.363116
5                      ram    2.267805
6                  battery    4.058535
7                   weight    6.392276
8                days_used    2.579905
9       normalized_new_price  3.217954
10      years_since_release   4.872942
11       brand_name_Alcatel   1.195476
12         brand_name_Apple   1.184456
13          brand_name_Asus   1.210719
14    brand_name_BlackBerry   1.107081
15        brand_name_Celkon   1.211839
16       brand_name_Coolpad   1.058128
17        brand_name_Gionee   1.090095
18        brand_name_Google   1.054360
19           brand_name_HTC   1.210991
20         brand_name_Honor   1.281330
21        brand_name_Huawei   1.489810
22       brand_name_Infinix   1.037168
23       brand_name_Karbonn   1.068685
24            brand_name_LG   1.363852
25          brand_name_Lava   1.079987
26        brand_name_Lenovo   1.280555
27         brand_name_Meizu   1.148019
28      brand_name_Micromax   1.231572
29     brand_name_Microsoft   1.588226
30      brand_name_Motorola   1.248283
31         brand_name_Nokia   1.510422
32       brand_name_OnePlus   1.101703
33          brand_name_Oppo   1.371989
34     brand_name_Panasonic   1.077234
35        brand_name_Realme   1.149137
36       brand_name_Samsung   1.564781
37          brand_name_Sony   1.198731
38         brand_name_Spice   1.073204
39          brand_name_Vivo   1.316598
40          brand_name_XOLO   1.105375
41        brand_name_Xiaomi   1.338350
42           brand_name_ZTE   1.280910
43                os_Others   1.770628
44               os_Windows   1.740865
45                   4g_yes   2.541549
46                   5g_yes   1.800819
```

In [125…

```
# Drop screen_size
col_to_drop = "screen_size"
x_train2 = x_train2.loc[:, ~x_train2.columns.str.startswith(col_to_drop)]
x_test2 = x_test2.loc[:, ~x_test2.columns.str.startswith(col_to_drop)]

# Recheck VIF after dropping screen_size
vif = checking_vif(x_train2)
print(f"VIF after dropping {col_to_drop}:\n", vif)
```

```
VIF after dropping screen_size:
                   feature         VIF
0                    const  130.436836
1           main_camera_mp    2.413743
2         selfie_camera_mp    2.854342
3               int_memory    1.357339
4                      ram    2.267477
5                  battery    3.758659
6                   weight    2.857329
7                days_used    2.566869
8     normalized_new_price    3.167788
9      years_since_release    4.730224
10       brand_name_Alcatel    1.182609
11         brand_name_Apple    1.184452
12          brand_name_Asus    1.210356
13    brand_name_BlackBerry    1.107037
14        brand_name_Celkon    1.211740
15       brand_name_Coolpad    1.058083
16        brand_name_Gionee    1.088254
17        brand_name_Google    1.053976
18           brand_name_HTC    1.210825
19         brand_name_Honor    1.269136
20        brand_name_Huawei    1.480962
21       brand_name_Infinix    1.037158
22       brand_name_Karbonn    1.064654
23            brand_name_LG    1.363816
24          brand_name_Lava    1.078696
25        brand_name_Lenovo    1.279337
26         brand_name_Meizu    1.147774
27      brand_name_Micromax    1.229207
28     brand_name_Microsoft    1.588222
29      brand_name_Motorola    1.247552
30         brand_name_Nokia    1.509297
31       brand_name_OnePlus    1.100250
32          brand_name_Oppo    1.371241
33     brand_name_Panasonic    1.075914
34        brand_name_Realme    1.148831
35       brand_name_Samsung    1.564129
36          brand_name_Sony    1.198683
37         brand_name_Spice    1.071737
38          brand_name_Vivo    1.308470
39          brand_name_XOLO    1.103130
40        brand_name_Xiaomi    1.336752
41           brand_name_ZTE    1.280645
42                os_Others    1.511743
43               os_Windows    1.740703
44                   4g_yes    2.540811
45                   5g_yes    1.797220
```

Dropping high p-value variables

- 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, I'll not drop all variables at once.
- Instead, I 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 [127…

```python
# initial list of columns
predictors = x_train2.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
```

```python
selected_features = cols
print(selected_features)
```

```
['const', 'main_camera_mp', 'selfie_camera_mp', 'int_memory', 'ram', 'weight', 'normalized_new_price', 'years_since_release', 'brand_name_Asus', 'brand_name_Celkon', 'brand_name_Nokia', 'brand_name_Xiaomi', 'os_Others', '4g_yes', '5g_yes']
```

In [128…
```python
# Update train and test datasets with selected features
x_train3 = x_train2[selected_features]
x_test3 = x_test2[selected_features]
```

In [129…
```python
# Fit the final OLS model
olsmodel2 = sm.OLS(y_train, x_train3).fit()

# Print the summary of the model
print(olsmodel2.summary())
```

```
                              OLS Regression Results
==================================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.843
Model:                               OLS   Adj. R-squared:                  0.842
Method:                    Least Squares   F-statistic:                     920.4
Date:                   Thu, 23 Jan 2025   Prob (F-statistic):               0.00
Time:                           16:50:51   Log-Likelihood:                 78.083
No. Observations:                   2417   AIC:                            -126.2
Df Residuals:                       2402   BIC:                            -39.31
Df Model:                             14
Covariance Type:               nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
const                    1.5574      0.047     32.935      0.000       1.465       1.650
main_camera_mp           0.0233      0.001     15.995      0.000       0.020       0.026
selfie_camera_mp         0.0126      0.001     11.178      0.000       0.010       0.015
int_memory               0.0002   6.74e-05      2.340      0.019    2.56e-05       0.000
ram                      0.0302      0.005      5.780      0.000       0.020       0.040
weight                   0.0016   5.98e-05     27.293      0.000       0.002       0.002
normalized_new_price     0.4199      0.011     37.634      0.000       0.398       0.442
years_since_release     -0.0302      0.003     -8.735      0.000      -0.037      -0.023
brand_name_Asus          0.0618      0.026      2.351      0.019       0.010       0.113
brand_name_Celkon       -0.1903      0.054     -3.520      0.000      -0.296      -0.084
brand_name_Nokia         0.0700      0.030      2.340      0.019       0.011       0.129
brand_name_Xiaomi        0.0866      0.025      3.403      0.001       0.037       0.136
os_Others               -0.1598      0.028     -5.718      0.000      -0.215      -0.105
4g_yes                   0.0433      0.015      2.830      0.005       0.013       0.073
5g_yes                  -0.0966      0.032     -3.031      0.002      -0.159      -0.034
==============================================================================
Omnibus:                      241.022   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              628.246
Skew:                          -0.560   Prob(JB):                    3.78e-137
Kurtosis:                       5.233   Cond. No.                     2.46e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.46e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [130…
```python
# checking model performance on train set (seen 70% data)
print("Training Performance\n")
olsmodel2_train_perf = model_performance_regression(olsmodel2, x_train3, y_train)
olsmodel2_train_perf
```

Training Performance

Out[130…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.234279	0.182035	0.842885	0.841904	4.390436

In [131…
```python
# checking model performance on test set (seen 30% data)
print("Test Performance\n")
olsmodel2_test_perf = model_performance_regression(olsmodel2, x_test3, y_test)
olsmodel2_test_perf
```

Test Performance

Out[131…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.24076	0.189677	0.829783	0.827283	4.547638

Now I'll check the rest of the assumptions on *olsmod2*.

2. **Linearity of variables**

3. **Independence of error terms**

4. **Normality of error terms**

5. **No Heteroscedasticity**

TEST FOR LINEARITY AND INDEPENDENCE

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

In [135…

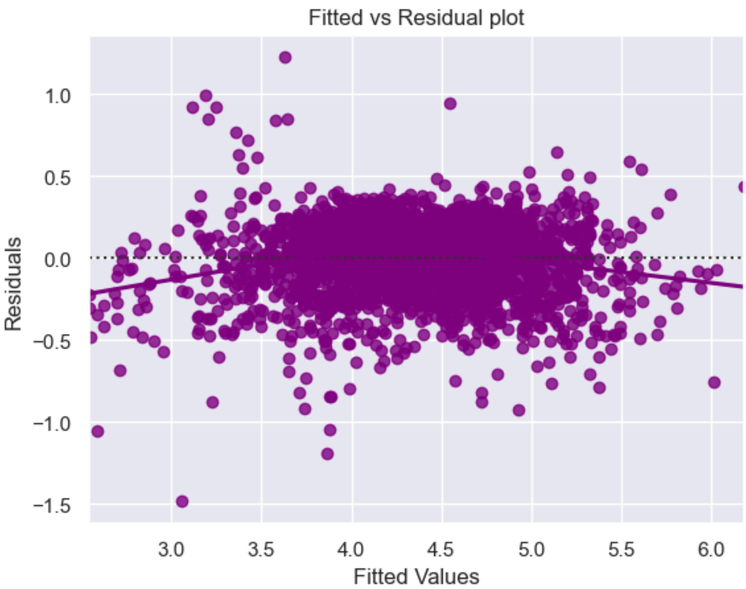
```
#create a dataframe with actual, fitted and residual values
df_pred = pd.DataFrame()

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()
```

Out[135…

	Actual Values	Fitted Values	Residuals
1744	4.261975	4.332388	-0.070412
3141	4.175156	3.913583	0.261573
1233	4.117410	4.426561	-0.309151
3046	3.782597	3.878337	-0.095740
2649	3.981922	3.888687	0.093235

In [136…

```
#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()
```

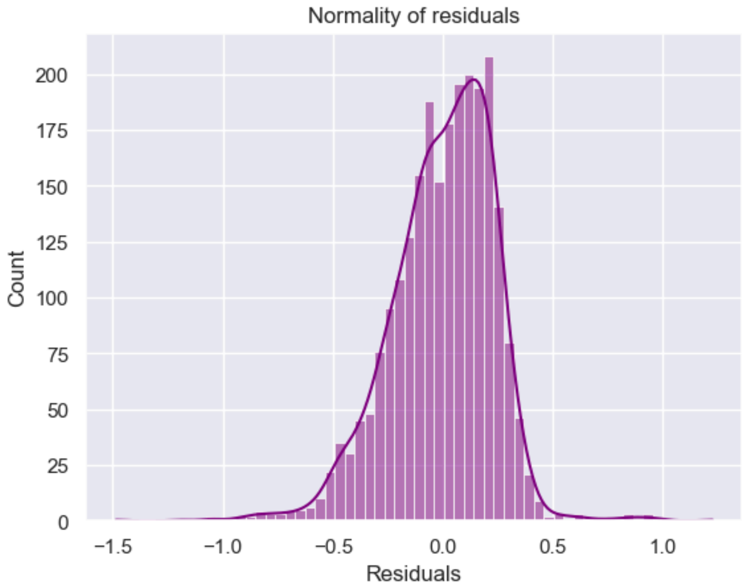


TEST FOR NORMALITY

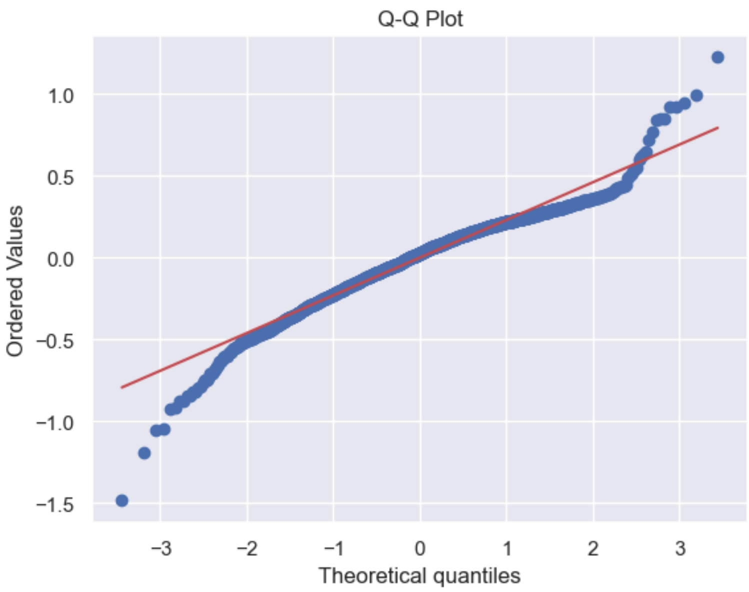
- Will test for normality by checking the distribution of residuals, by checking the Q-Q plot of residuals, and by using the Shapiro-Wilk test.
- If the residuals follow a normal distribution, they will make a straight line plot, otherwise not.
- If the p-value of the Shapiro-Wilk test is greater than 0.05, I can say the residuals are normally distributed.

In [138…

```
sns.histplot(data=df_pred, x="Residuals", kde=True, color="purple")
plt.title("Normality of residuals")
plt.show()
```



In [139…
```
import pylab
import scipy.stats as stats

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



In [140…
```
shapiro_test = shapiro(df_pred["Residuals"])
print(f"Shapiro-Wilk Test: W = {shapiro_test.statistic}, p-value = {shapiro_test.pvalue}")
```

```
Shapiro-Wilk Test: W = 0.9652157434919437, p-value = 1.0565242501108041e-23
```

TEST FOR HOMOSCEDASTICITY

- Will test for homoscedasticity by using the goldfeldquandt test.
- If I get a p-value greater than 0.05, I can say that the residuals are homoscedastic. Otherwise, they are heteroscedastic.

In [143…
```
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[143…
```
[('F statistic', 0.9499552250892246), ('p-value', 0.8123487842190585)]
```

Final Model Summary

In [145…
```
olsmodel_final = sm.OLS(y_train, x_train3).fit()
print(olsmodel_final.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.843
Model:                               OLS   Adj. R-squared:                  0.842
Method:                    Least Squares   F-statistic:                     920.4
Date:                   Thu, 23 Jan 2025   Prob (F-statistic):               0.00
Time:                           16:50:53   Log-Likelihood:                 78.083
No. Observations:                   2417   AIC:                            -126.2
Df Residuals:                       2402   BIC:                            -39.31
Df Model:                             14
Covariance Type:               nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                     1.5574      0.047     32.935      0.000       1.465       1.650
main_camera_mp            0.0233      0.001     15.995      0.000       0.020       0.026
selfie_camera_mp          0.0126      0.001     11.178      0.000       0.010       0.015
int_memory                0.0002   6.74e-05      2.340      0.019    2.56e-05       0.000
ram                       0.0302      0.005      5.780      0.000       0.020       0.040
weight                    0.0016   5.98e-05     27.293      0.000       0.002       0.002
normalized_new_price      0.4199      0.011     37.634      0.000       0.398       0.442
years_since_release      -0.0302      0.003     -8.735      0.000      -0.037      -0.023
brand_name_Asus           0.0618      0.026      2.351      0.019       0.010       0.113
brand_name_Celkon        -0.1903      0.054     -3.520      0.000      -0.296      -0.084
brand_name_Nokia          0.0700      0.030      2.340      0.019       0.011       0.129
brand_name_Xiaomi         0.0866      0.025      3.403      0.001       0.037       0.136
os_Others                -0.1598      0.028     -5.718      0.000      -0.215      -0.105
4g_yes                    0.0433      0.015      2.830      0.005       0.013       0.073
5g_yes                   -0.0966      0.032     -3.031      0.002      -0.159      -0.034
==============================================================================
Omnibus:                      241.022   Durbin-Watson:                   1.997
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              628.246
Skew:                          -0.560   Prob(JB):                    3.78e-137
Kurtosis:                       5.233   Cond. No.                     2.46e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.46e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [146…

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

Training Performance

Out[146…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.234279	0.182035	0.842885	0.841904	4.390436

In [147…

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

Test Performance

Out[147…

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.24076	0.189677	0.829783	0.827283	4.547638

Actionable Insights and Recommendations

- Turned Into Slides
