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 [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import MaxNLocator
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Loading the dataset

In [2]:
```
df=pd.read_csv("used_device_data.csv")
df.head()
```

Out[2]:

	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.(
1	Honor	Android	17.30	yes	yes	13.0	16.0	128.0	8.(
2	Honor	Android	16.69	yes	yes	13.0	8.0	128.0	8.(
3	Honor	Android	25.50	yes	yes	13.0	8.0	64.0	6.(
4	Honor	Android	15.32	yes	no	13.0	8.0	64.0	3.(

Data Overview

- Observations
- Sanity checks

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   brand_name        3454 non-null   object 
 1   os                3454 non-null   object 
 2   screen_size       3454 non-null   float64
 3   4g                3454 non-null   object 
 4   5g                3454 non-null   object 
 5   main_camera_mp    3275 non-null   float64
 6   selfie_camera_mp  3452 non-null   float64
 7   int_memory        3450 non-null   float64
```

```
 8   ram                    3450 non-null   float64
 9   battery                3448 non-null   float64
 10  weight                 3447 non-null   float64
 11  release_year           3454 non-null   int64
 12  days_used              3454 non-null   int64
 13  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
```

- There are a total of 3454 entries in the dataset for 15 columns
- main_camera_mp, selfie_camera_mp, int_memory, ram, battery, weight columns have some null entries
- brand_name and os have categorical data
- normalized_used_price is our response variable and rest are predictor variables

In [4]:
```
df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	
screen_size	3454.0	13.713115	3.805280	5.080000	12.700000	12.830000	15
main_camera_mp	3275.0	9.460208	4.815461	0.080000	5.000000	8.000000	13
selfie_camera_mp	3452.0	6.554229	6.970372	0.000000	2.000000	5.000000	8
int_memory	3450.0	54.573099	84.972371	0.010000	16.000000	32.000000	64
ram	3450.0	4.036122	1.365105	0.020000	4.000000	4.000000	4
battery	3448.0	3133.402697	1299.682844	500.000000	2100.000000	3000.000000	4000
weight	3447.0	182.751871	88.413228	69.000000	142.000000	160.000000	185
release_year	3454.0	2015.965258	2.298455	2013.000000	2014.000000	2015.500000	2018
days_used	3454.0	674.869716	248.580166	91.000000	533.500000	690.500000	868
normalized_used_price	3454.0	4.364712	0.588914	1.536867	4.033931	4.405133	4
normalized_new_price	3454.0	5.233107	0.683637	2.901422	4.790342	5.245892	5

- There is huge variation in absolute values of selfie_camera_mp and battery ; Perhaps, some scaling needs to be performed
- Some columns appear to have outliers

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 [5]:
```
plt.figure(figsize=(20, 8))
sns.set_style("dark")
ax=sns.histplot(data=df, x='normalized_used_price', kde=True,bins=20,color='chocolat
plt.axvline(df['normalized_used_price'].mean(), color='red', linewidth=2)
plt.axvline(df['normalized_used_price'].median(), color='blue', linewidth=2)
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.title("Distribution of Normalized Used Device Prices")
plt.show()
```

- The normalized_used_price column appears to follow a left-skewed normal distribution
- There is suspicion of outliers so let's look for them

In [6]:
```
plt.figure(figsize=(8, 8))
sns.set_style("darkgrid")
sns.boxplot(data=df,y=df['normalized_used_price'], flierprops={"markerfacecolor": "w
plt.ylabel(' Normalized Used Price')
plt.title('Boxplot for Normalized Used Price')
plt.show()
```

We will deal with outliers in Data Preprocessing section

In [7]:
```
print('Percentage of the used device market that is dominated by Android devices :',
```

```
Percentage of the used device market that is dominated by Android devices : 93.05 %
```

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

- We need to define our own criteria for RAM needed for smooth functioning of device
- We can look at days_used column and average out the RAM of phones which last beyond 700 days (roughly 2 years)

In [8]:
```
avg_ram = df[df['days_used'] > 700]['ram'].mean()
print('The amount of RAM is important for the smooth functioning of a device : ',rou
```

```
The amount of RAM is important for the smooth functioning of a device :  4.0 GB
```

In [9]:
```
# Create a box plot to visualize RAM distribution by brand
plt.figure(figsize=(30, 6))
```

```
sns.boxplot(x='brand_name', y='ram', data=df)
plt.title('RAM Distribution by Brand')
plt.xlabel('Brand Name')  # Change x-axis label to 'Brand Name'
plt.ylabel('RAM (GB)')    # Change y-axis label to 'RAM (GB)'
plt.xticks(rotation=90)   # Rotate x-axis labels for better readability
plt.show()
```

- Boxplot for RAM of all brands does not show anything conclusive
- We can plot mean RAM of devices by all brands
- We can plot violinplot for top 6 brands (by number of devices in dataset)

In [10]:
```
plt.figure(figsize=(20, 8))
sns.barplot(x='brand_name', y='ram', data=df, estimator=lambda x: sum(x) / len(x))
plt.title('Mean RAM by Brand')
plt.xlabel('Brand Name')
plt.ylabel('Mean RAM (GB)')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

- Each bar has an error bar. These error bars represent the confidence interval around the mean
- OnePlus seems to offer devices with higher RAM

In [11]:
```
brand_counts = df['brand_name'].value_counts()

# Select the top 6 brands with the most devices
top_brands = brand_counts.head(6).index.tolist()

# Create df2 containing only the data for the top 6 brands
df2 = df[df['brand_name'].isin(top_brands)].copy()

# Create a box plot to visualize RAM distribution by top 6 brands
plt.figure(figsize=(20, 6))
sns.violinplot(x='brand_name', y='ram', data=df2)  # Keep the original column names
plt.title('RAM Distribution by Top 6 Brands')
plt.xlabel('Brand Name')  # Change x-axis label to 'Brand Name'
plt.ylabel('RAM (GB)')    # Change y-axis label to 'RAM (GB)'
plt.xticks(rotation=90)   # Rotate x-axis labels for better readability
plt.show()
```

- Most devices appear to have RAM around 4 GB
- Brands like Huawei and Samsung offer devices with RAM as high as 12 GB

1. A large battery often increases a device's weight, making it feel uncomfortable in the hands. How does the weight vary for phones and tablets offering large batteries (more than 4500 mAh)?

In [12]:
```
plt.figure(figsize=(20, 6))
sns.histplot(data=df[df['battery'] > 4500], x='weight', bins=10, kde=True)
plt.title('Weight Distribution for Devices with >4500 mAh Battery')
plt.xlabel('Weight')
plt.ylabel('Frequency')
plt.show()
```

```
plt.figure(figsize=(20, 6))
#sns.scatterplot(data=df[df['battery'] > 4500], x='battery', y='weight', alpha=0.5)
sns.regplot(data=df[df['battery'] > 4500], x='battery', y='weight', scatter_kws={"al

plt.title('Scatterplot of Weight vs. Battery Capacity (more than 4500 mAh)')
plt.xlabel('Battery Capacity (mAh)')
plt.ylabel('Weight')
plt.show()
```

- A weak positive correlation can be seen between weight and battery capacity in the scatterplot

1. Bigger screens are desirable for entertainment purposes as they offer a better viewing experience. How many phones and tablets are available across different brands with a screen size larger than 6 inches?

In [13]:
```
# Filter phones with screen size larger than 6 inches
large_screen_phones = df[(df['screen_size'] > 6*2.54)] # Convert inch to cm since sc

# Count the number of phones by brand
phone_counts = large_screen_phones['brand_name'].value_counts()

# Display the counts
print(phone_counts)
print('Number of phones and tablets are available across different brands with a scr
```

```
Huawei        149
Samsung       119
Others         99
Vivo           80
Honor          72
Oppo           70
Lenovo         69
Xiaomi         69
LG             59
Motorola       42
Asus           41
Realme         40
Alcatel        26
Apple          24
Acer           19
Meizu          17
ZTE            17
OnePlus        16
Nokia          15
Sony           12
Infinix        10
Micromax        7
HTC             7
Google          4
XOLO            3
Gionee          3
Coolpad         3
Panasonic       2
Karbonn         2
Spice           2
Microsoft       1
Name: brand_name, dtype: int64
Number of phones and tablets are available across different brands with a screen size
larger than 6 inches : 1099
```

In [14]:
```
plt.figure(figsize=(20, 6))
sns.barplot(x=phone_counts.index, y=phone_counts.values)
plt.title('Number of Phones with Screen Size > 6 Inches by Brand')
plt.xlabel('Brand Name')
plt.ylabel('Count')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

- Huawei and Samsung appear to offer most devices with large screen sizes

1. 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?

In [15]:
```
plt.figure(figsize=(20,6))
sns.barplot(x=df[df['selfie_camera_mp'] > 8]['brand_name'].value_counts().index, y=d
plt.title('Number of Devices with Selfie Cameras > 8MP by Brand')
plt.xlabel('Brand Name')
plt.ylabel('Count')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

- Oppo, Vivo are can be seen offering high specifications for camera parameters

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

In [16]:
```
plt.figure(figsize=(15, 10))
sns.heatmap(df.corr(), annot=True, cmap='BuGn', linewidths=0.5)
sns.set_style("darkgrid")
#plt.title('Countplot for Different Number of Payment Installments')
plt.show()
```

- normalized_used_price appears highly correlated with normalized_new_price
- screen_size appears highly correlated with battery and weight. It makes sense that larger screens weigh more and can accomodate larger batteries
- days_used shows strong negative correlation with release year. The earlier a device is released, the more time it has to show how long it can be used
- screen_size, main_camera_mp, selfie_camera_mp, ram, battery, release_year appear weakly correlated with normalized_used_price

In [17]:
```
sns.pairplot(df)
plt.show()
```

- The insights drawn from the correlation plot are reflected in the scatterplots

In [18]:
```
plt.figure(figsize=(15, 8))
sns.boxplot(x='brand_name', y='days_used', data=df)
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.xlabel('Brand Name')
```

```
plt.ylabel('Days Used')
plt.title('Boxplot of Days Used for Each Brand')

# Show the plot
plt.tight_layout()  # Ensure the plot is well-arranged
plt.show()
```

- Brands like Infinix, Reliance stand out in being used for low number of days
- Most companies have similar maximum values for days used indicating that there are durable devices offered by all brands
- Let's plot a scatter plot of days used and new_price

In [19]:
```
plt.figure(figsize=(12, 10))  # Adjust the figure size as needed
sns.scatterplot(x='normalized_new_price', y='days_used', hue='brand_name', data=df,

# Customize labels and title
plt.xlabel('Normalized New Price')
plt.ylabel('Days Used')
plt.title('Scatterplot of Normalized New Price vs. Days Used (Colored by Brand)')

# Show a legend
plt.legend(title='Brand', loc='upper right', bbox_to_anchor=(1.15, 1))

# Show the plot
plt.tight_layout()  # Ensure the plot is well-arranged
plt.show()
```

- Nothing conclusive can be drawn from the scatter plot

In [20]:
```
df['os'].unique()
```

Out[20]:
```
array(['Android', 'Others', 'iOS', 'Windows'], dtype=object)
```

In [21]:
```
# Count the number of devices for each OS category
os_counts = df['os'].value_counts()

# Create a bar plot using Seaborn
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
sns.barplot(x=os_counts.index, y=os_counts.values, palette='viridis')

# Customize labels and title
plt.xlabel('Operating System')
plt.ylabel('Number of Devices')
plt.title('Number of Devices for Each Operating System')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()  # Ensure the plot is well-arranged
plt.show()
```

- The data is dominated by Android devices
- The data for os other than Android may be slightly insufficient to train the model

In [22]:
```
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed

# Create the scatterplot
sns.scatterplot(x='main_camera_mp', y='selfie_camera_mp', data=df, alpha=0.7)

# Customize labels and title
plt.xlabel('Main Camera Megapixels')
plt.ylabel('Selfie Camera Megapixels')
plt.title('Scatterplot of Main Camera Megapixels vs. Selfie Camera Megapixels')

# Show the plot
plt.tight_layout()  # Ensure the plot is well-arranged
plt.show()
```

In [23]:
```
# Let's check if there is any trend of screen size with release year
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed

# Group the data by 'release_year' and calculate the mean 'screen_size' for each yea
mean_screen_size_by_year = df.groupby('release_year')['screen_size'].mean().reset_in

# Create a line plot using Seaborn
sns.lineplot(x='release_year', y='screen_size', data=mean_screen_size_by_year, marke

# Customize labels and title
plt.xlabel('Release Year')
plt.ylabel('Mean Screen Size (cm)')
plt.title('Mean Screen Size by Release Year')
plt.grid(True)

# Show the plot
plt.tight_layout()  # Ensure the plot is well-arranged
plt.show()
```

- With every passing year, the average screen size follows an increasing trend

In [24]:
```
# Let's see if RAM follows any trend
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed

# Group the data by 'release_year' and calculate the mean 'ram' for each year
mean_ram_by_year = df.groupby('release_year')['ram'].mean().reset_index()

# Create a line plot using Seaborn
sns.lineplot(x='release_year', y='ram', data=mean_ram_by_year, marker='o', linestyle

# Customize labels and title
plt.xlabel('Release Year')
plt.ylabel('Mean RAM (GB)')
plt.title('Mean RAM by Release Year')
plt.grid(True)

# Show the plot
plt.tight_layout()  # Ensure the plot is well-arranged
plt.show()
```

- Some technical breakthrough around 2018-19 seems to have taken place

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 [25]:
```
# Let's look at df.info again for missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   brand_name             3454 non-null   object 
 1   os                     3454 non-null   object 
 2   screen_size            3454 non-null   float64
 3   4g                     3454 non-null   object 
 4   5g                     3454 non-null   object 
 5   main_camera_mp         3275 non-null   float64
 6   selfie_camera_mp       3452 non-null   float64
 7   int_memory             3450 non-null   float64
 8   ram                    3450 non-null   float64
 9   battery                3448 non-null   float64
 10  weight                 3447 non-null   float64
 11  release_year           3454 non-null   int64  
 12  days_used              3454 non-null   int64  
 13  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 [26]:
```
# Let's calculate the percentage of null values in each column

max_column_length = max(len(column) for column in df.columns)

for i in df.columns:
    null_percentage = (df[i].isnull().sum() / len(df)) * 100
    print(f"{i:{max_column_length}s}: {null_percentage:.2f}% null values")
```

```
brand_name           : 0.00% null values
os                   : 0.00% null values
screen_size          : 0.00% null values
4g                   : 0.00% null values
5g                   : 0.00% null values
main_camera_mp       : 5.18% null values
selfie_camera_mp     : 0.06% null values
int_memory           : 0.12% null values
ram                  : 0.12% null values
battery              : 0.17% null values
weight               : 0.20% null values
release_year         : 0.00% null values
days_used            : 0.00% null values
normalized_used_price: 0.00% null values
normalized_new_price : 0.00% null values
```

- main_camera_mp has significantly higher missing values compared to rest. We can impute values for main_camera_mp
- For rest we can simply remove entries having null values

In [27]:
```
print("Dataset size before dropping :",len(df))
columns_to_check = ['selfie_camera_mp', 'int_memory', 'ram', 'battery', 'weight']
df = df.dropna(subset=columns_to_check, how='any').reset_index(drop=True)
print("Dataset size after dropping :",len(df))
```

```
Dataset size before dropping : 3454
Dataset size after dropping : 3432
```

In [28]:
```
df.isna().any()
```

Out[28]:
```
brand_name               False
os                       False
screen_size              False
4g                       False
5g                       False
main_camera_mp            True
selfie_camera_mp         False
int_memory               False
ram                      False
battery                  False
weight                   False
release_year             False
days_used                False
normalized_used_price    False
normalized_new_price     False
dtype: bool
```

In [29]:
```
# Let's check the entries where main_camera_mp has missing values
df_check=df[df['main_camera_mp'].isnull()]
df_check
```

Out[29]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory
59	Infinix	Android	17.32	yes	no	NaN	8.0	32.0
60	Infinix	Android	15.39	yes	no	NaN	8.0	64.0
61	Infinix	Android	15.39	yes	no	NaN	8.0	32.0
62	Infinix	Android	15.39	yes	no	NaN	16.0	32.0
63	Infinix	Android	15.29	yes	no	NaN	16.0	32.0
...	...	...	...	...	...	...	...	...
3389	Realme	Android	15.34	yes	no	NaN	16.0	64.0
3390	Realme	Android	15.32	yes	no	NaN	16.0	64.0
3391	Realme	Android	15.32	yes	no	NaN	25.0	64.0
3426	Asus	Android	16.74	yes	no	NaN	24.0	128.0
3427	Asus	Android	15.34	yes	no	NaN	8.0	64.0

179 rows × 15 columns

In [30]:
```
df_check['brand_name'].unique()
```

Out[30]:
```
array(['Infinix', 'Lava', 'Meizu', 'Motorola', 'OnePlus', 'Oppo',
       'Realme', 'Vivo', 'Xiaomi', 'ZTE', 'Coolpad', 'Asus', 'BlackBerry',
       'Panasonic', 'Sony'], dtype=object)
```

- No pattern detected for missing values
- Let's normalize the columns first

In [31]:
```
# Initialize the StandardScaler
scaler = StandardScaler()

# Define the columns to exclude from standardization
columns_to_exclude = ['brand_name','normalized_used_price','os','4g','5g']

# Select the columns to standardize (exclude the ones to exclude)
columns_to_standardize = [col for col in df.columns if col not in columns_to_exclude

# Fit and transform the selected columns using StandardScaler
df[columns_to_standardize] = scaler.fit_transform(df[columns_to_standardize])
```

In [32]:
```
df.head()
```

Out[32]:

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	
0	Honor	Android	0.202287	yes	no	0.733869	-0.226736	0.108732	-0.7
1	Honor	Android	0.941416	yes	yes	0.733869	1.349615	0.860447	2.9
2	Honor	Android	0.780392	yes	yes	0.733869	0.203178	0.860447	2.9
3	Honor	Android	3.106008	yes	yes	0.733869	0.203178	0.108732	1.4
4	Honor	Android	0.418747	yes	no	0.733869	0.203178	0.108732	-0.7

In [33]:
```
# We also need to perform one-hot encoding for categorical variables
cat_variables = df[['brand_name', 'os','4g','5g']]
cat_dummies = pd.get_dummies(cat_variables, drop_first=True)
cat_dummies.head()
```

Out[33]:

	brand_name_Alcatel	brand_name_Apple	brand_name_Asus	brand_name_BlackBerry	brand_name_C
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	

5 rows × 38 columns

In [34]:
```
cat_dummies.columns
```

Out[34]:
```
Index(['brand_name_Alcatel', 'brand_name_Apple', 'brand_name_Asus',
       'brand_name_BlackBerry', 'brand_name_Celkon', 'brand_name_Coolpad',
       'brand_name_Gionee', 'brand_name_Google', 'brand_name_HTC',
       'brand_name_Honor', 'brand_name_Huawei', 'brand_name_Infinix',
       'brand_name_Karbonn', 'brand_name_LG', 'brand_name_Lava',
       'brand_name_Lenovo', 'brand_name_Meizu', 'brand_name_Micromax',
       'brand_name_Microsoft', 'brand_name_Motorola', 'brand_name_Nokia',
       'brand_name_OnePlus', 'brand_name_Oppo', 'brand_name_Others',
       'brand_name_Panasonic', 'brand_name_Realme', 'brand_name_Samsung',
       'brand_name_Sony', 'brand_name_Spice', 'brand_name_Vivo',
       'brand_name_XOLO', 'brand_name_Xiaomi', 'brand_name_ZTE', 'os_Others',
       'os_Windows', 'os_iOS', '4g_yes', '5g_yes'],
      dtype='object')
```

In [35]:
```
df = df.drop(['brand_name', 'os','4g','5g'], axis=1)
df = pd.concat([df, cat_dummies], axis=1)
df.head()
```

Out[35]:

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	rele
0	0.202287	0.733869	-0.226736	0.108732	-0.766332	-0.091659	-0.418656	
1	0.941416	0.733869	1.349615	0.860447	2.910510	0.893941	0.342115	
2	0.780392	0.733869	0.203178	0.860447	2.910510	0.816941	0.342115	
3	3.106008	0.733869	0.203178	0.108732	1.439773	3.165443	3.373845	
4	0.418747	0.733869	0.203178	0.108732	-0.766332	1.432942	0.024181	

5 rows × 49 columns

In [36]:
```
# Let's impute missing values in main_camera_mp now using KNN imputer
imputer = KNNImputer(n_neighbors=5)
df = pd.DataFrame(imputer.fit_transform(df),columns = df.columns)
```

In [37]:
```
df.isna().any() # Final check for null values
```

Out[37]:
```
screen_size              False
main_camera_mp           False
selfie_camera_mp         False
int_memory               False
ram                      False
battery                  False
weight                   False
release_year             False
days_used                False
normalized_used_price    False
normalized_new_price     False
brand_name_Alcatel       False
brand_name_Apple         False
brand_name_Asus          False
brand_name_BlackBerry    False
brand_name_Celkon        False
brand_name_Coolpad       False
brand_name_Gionee        False
brand_name_Google        False
brand_name_HTC           False
brand_name_Honor         False
brand_name_Huawei        False
brand_name_Infinix       False
brand_name_Karbonn       False
brand_name_LG            False
brand_name_Lava          False
brand_name_Lenovo        False
brand_name_Meizu         False
brand_name_Micromax      False
brand_name_Microsoft     False
brand_name_Motorola      False
brand_name_Nokia         False
brand_name_OnePlus       False
brand_name_Oppo          False
brand_name_Others        False
brand_name_Panasonic     False
brand_name_Realme        False
brand_name_Samsung       False
brand_name_Sony          False
brand_name_Spice         False
brand_name_Vivo          False
```

```
brand_name_XOLO          False
brand_name_Xiaomi        False
brand_name_ZTE           False
os_Others                False
os_Windows               False
os_iOS                   False
4g_yes                   False
5g_yes                   False
dtype: bool
```

- We do not have any null values anymore
- Let's deal with outliers now

In [38]:
```python
columns_to_plot = ['screen_size', 'main_camera_mp', 'selfie_camera_mp', 'int_memory'
                   'ram', 'battery', 'weight', 'release_year', 'days_used',
                   'normalized_used_price', 'normalized_new_price']

# Create a subset of the DataFrame with the selected columns
subset_df = df[columns_to_plot]

# Set the figure size
plt.figure(figsize=(20,8))

# Create boxplots for the selected columns
sns.boxplot(data=subset_df)

# Set the title and labels
plt.title('Boxplots for Selected Columns in df')
plt.xlabel('Columns')
plt.ylabel('Values')

# Rotate x-axis labels for better readability (optional)
plt.xticks(rotation=45)

# Display the plot
plt.show()

# Create an empty dictionary to store outlier counts
outlier_counts = {}

# Define the IQR threshold for detecting outliers (you can adjust this value)
iqr_threshold = 1.5

# Calculate the IQR and identify outliers for each column
for column in columns_to_check:
    # Calculate the IQR for the current column
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    # Identify outliers using the IQR threshold
    lower_bound = Q1 - iqr_threshold * IQR
    upper_bound = Q3 + iqr_threshold * IQR

    # Count the number of outliers
    num_outliers = ((df[column] < lower_bound) | (df[column] > upper_bound)).sum()

    # Store the outlier count in the dictionary
    outlier_counts[column] = num_outliers

# Display the outlier counts for each column
for column, count in outlier_counts.items():
    print(f"{column}: {count} outliers")
```

```
selfie_camera_mp: 221 outliers
int_memory: 138 outliers
ram: 630 outliers
battery: 77 outliers
weight: 367 outliers
```

- The outliers observed here have information. They're not wrong readings. Some phones realistically have bigger screen sizes. Some phones realistically have greater storage (such as 1 TB) and RAM. These values are crucial to determining the price of a phone. So in my opinion, outliers should be left as they are.

EDA

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

In [39]:
```python
# Filter out binary (one-hot encoded) variables (exclude columns with values of 0 an
non_binary_columns = df.columns[(df.min() != 0) | (df.max() != 1)]

# Select only the numeric columns (excluding one-hot encoded variables)
numeric_columns = df[non_binary_columns].select_dtypes(include=['number'])

# Set the figure size
plt.figure(figsize=(15, 10))

# Create a heatmap of correlations
sns.heatmap(numeric_columns.corr(), annot=True, cmap='BuGn', linewidths=0.5)
sns.set_style("darkgrid")

# Set the plot title (optional)
plt.title('Correlation Heatmap for Numeric Variables (Excluding Binary Variables)')

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

- Similar outcomes observed for correlation values when compared to observations before data-preprocessing

In [40]:
```python
df.describe().T
```

Out[40]:

	count	mean	std	min	25%	50%	75%
screen_size	3432.0	-5.387267e-16	1.000146	-2.284354	-0.272867	-0.238550	0.431945
main_camera_mp	3432.0	1.566106e-02	0.981458	-1.956978	-0.932290	-0.307480	0.733869
selfie_camera_mp	3432.0	-2.627123e-15	1.000146	-0.943259	-0.656650	-0.226736	0.203178
int_memory	3432.0	-4.470168e-16	1.000146	-0.642865	-0.455053	-0.267125	0.108732
ram	3432.0	-1.399554e-15	1.000146	-2.957731	-0.030964	-0.030964	-0.030964
battery	3432.0	-5.580229e-16	1.000146	-2.032060	-0.800059	-0.107059	0.662941

	count	mean	std	min	25%	50%	75%
weight	3432.0	1.634845e-16	1.000146	-1.292975	-0.464076	-0.259689	0.024181
release_year	3432.0	-3.137156e-14	1.000146	-1.290424	-0.855424	0.014576	0.884576
days_used	3432.0	-2.476651e-16	1.000146	-2.349456	-0.563358	0.062884	0.776719
normalized_used_price	3432.0	4.368437e+00	0.584702	1.536867	4.037201	4.406536	4.757934
normalized_new_price	3432.0	-9.572115e-16	1.000146	-3.442099	-0.657127	0.014012	0.644251
brand_name_Alcatel	3432.0	3.525641e-02	0.184454	0.000000	0.000000	0.000000	0.000000
brand_name_Apple	3432.0	1.136364e-02	0.106008	0.000000	0.000000	0.000000	0.000000
brand_name_Asus	3432.0	3.554779e-02	0.185187	0.000000	0.000000	0.000000	0.000000
brand_name_BlackBerry	3432.0	6.410256e-03	0.079819	0.000000	0.000000	0.000000	0.000000
brand_name_Celkon	3432.0	9.615385e-03	0.097600	0.000000	0.000000	0.000000	0.000000
brand_name_Coolpad	3432.0	6.410256e-03	0.079819	0.000000	0.000000	0.000000	0.000000
brand_name_Gionee	3432.0	1.631702e-02	0.126710	0.000000	0.000000	0.000000	0.000000
brand_name_Google	3432.0	3.787879e-03	0.061438	0.000000	0.000000	0.000000	0.000000
brand_name_HTC	3432.0	3.205128e-02	0.176162	0.000000	0.000000	0.000000	0.000000
brand_name_Honor	3432.0	3.379953e-02	0.180739	0.000000	0.000000	0.000000	0.000000
brand_name_Huawei	3432.0	7.313520e-02	0.260396	0.000000	0.000000	0.000000	0.000000
brand_name_Infinix	3432.0	2.913753e-03	0.053908	0.000000	0.000000	0.000000	0.000000
brand_name_Karbonn	3432.0	8.449883e-03	0.091547	0.000000	0.000000	0.000000	0.000000
brand_name_LG	3432.0	5.856643e-02	0.234846	0.000000	0.000000	0.000000	0.000000
brand_name_Lava	3432.0	1.048951e-02	0.101895	0.000000	0.000000	0.000000	0.000000
brand_name_Lenovo	3432.0	4.982517e-02	0.217615	0.000000	0.000000	0.000000	0.000000
brand_name_Meizu	3432.0	1.719114e-02	0.130002	0.000000	0.000000	0.000000	0.000000
brand_name_Micromax	3432.0	3.409091e-02	0.181489	0.000000	0.000000	0.000000	0.000000
brand_name_Microsoft	3432.0	6.118881e-03	0.077995	0.000000	0.000000	0.000000	0.000000
brand_name_Motorola	3432.0	3.088578e-02	0.173033	0.000000	0.000000	0.000000	0.000000
brand_name_Nokia	3432.0	2.826340e-02	0.165749	0.000000	0.000000	0.000000	0.000000
brand_name_OnePlus	3432.0	6.410256e-03	0.079819	0.000000	0.000000	0.000000	0.000000
brand_name_Oppo	3432.0	3.758741e-02	0.190224	0.000000	0.000000	0.000000	0.000000
brand_name_Others	3432.0	1.462704e-01	0.353429	0.000000	0.000000	0.000000	0.000000
brand_name_Panasonic	3432.0	1.369464e-02	0.116237	0.000000	0.000000	0.000000	0.000000
brand_name_Realme	3432.0	1.194639e-02	0.108661	0.000000	0.000000	0.000000	0.000000
brand_name_Samsung	3432.0	9.935897e-02	0.299187	0.000000	0.000000	0.000000	0.000000
brand_name_Sony	3432.0	2.505828e-02	0.156325	0.000000	0.000000	0.000000	0.000000
brand_name_Spice	3432.0	8.741259e-03	0.093099	0.000000	0.000000	0.000000	0.000000

	count	mean	std	min	25%	50%	75%
brand_name_Vivo	3432.0	3.409091e-02	0.181489	0.000000	0.000000	0.000000	0.000000
brand_name_XOLO	3432.0	1.223776e-02	0.109961	0.000000	0.000000	0.000000	0.000000
brand_name_Xiaomi	3432.0	3.846154e-02	0.192336	0.000000	0.000000	0.000000	0.000000
brand_name_ZTE	3432.0	4.079254e-02	0.197838	0.000000	0.000000	0.000000	0.000000
os_Others	3432.0	3.729604e-02	0.189514	0.000000	0.000000	0.000000	0.000000
os_Windows	3432.0	1.893939e-02	0.136331	0.000000	0.000000	0.000000	0.000000
os_iOS	3432.0	1.048951e-02	0.101895	0.000000	0.000000	0.000000	0.000000
4g_yes	3432.0	6.780303e-01	0.467300	0.000000	0.000000	1.000000	1.000000
5g_yes	3432.0	4.428904e-02	0.205767	0.000000	0.000000	0.000000	0.000000

- The quantative variables now have values with comparable order of magnitude
- Categorical variables have been incorporated with one-hot encoding
- After one-hot encoding and normalization, it may not make a lot of sense to repeat the earlier visualizations, so let's proceed to model building

Model Building - Linear Regression

In [41]:
```
# defining X and y variables
X = df.drop(["normalized_used_price"], axis=1)
y = df["normalized_used_price"]

print(X.shape)
print(y.shape)
```

```
(3432, 48)
(3432,)
```

In [42]:
```
x_train1, x_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.3, random_
print("Number of rows in train data =", x_train1.shape[0])
print("Number of rows in test data =", x_test1.shape[0])

Lr = LinearRegression()
Lr.fit(x_train1, y_train1)
Lr.score(x_test1, y_test1)
```

```
Number of rows in train data = 2402
Number of rows in test data = 1030
```

Out[42]:
```
0.8371637712696953
```

In [43]:
```
# Make predictions on the training data
y_train_pred = Lr.predict(x_train1)

# Calculate evaluation metrics on the training data
mae_train = mean_absolute_error(y_train1, y_train_pred)
mse_train = mean_squared_error(y_train1, y_train_pred)
rmse_train = np.sqrt(mse_train)
r2_train = r2_score(y_train1, y_train_pred)

# Print the evaluation metrics for the training data
print(f"Training Mean Absolute Error (MAE): {mae_train}")
print(f"Training Mean Squared Error (MSE): {mse_train}")
```

```
print(f"Training Root Mean Squared Error (RMSE): {rmse_train}")
print(f"Training R-squared (R2) Score: {r2_train}")
```

```
Training Mean Absolute Error (MAE): 0.18018236789236586
Training Mean Squared Error (MSE): 0.05371788234437584
Training Root Mean Squared Error (RMSE): 0.2317711853194349
Training R-squared (R2) Score: 0.8453003157813386
```

In [44]:

```
y_pred1 = Lr.predict(x_test1)
mae = mean_absolute_error(y_test1, y_pred1)
mse = mean_squared_error(y_test1, y_pred1)
rmse = np.sqrt(mse)
r2 = r2_score(y_test1, y_pred1)

# Print the evaluation metrics
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2) Score: {r2}")
```

```
Mean Absolute Error (MAE): 0.18065245083485493
Mean Squared Error (MSE): 0.053578578481747796
Root Mean Squared Error (RMSE): 0.23147046999941007
R-squared (R2) Score: 0.8371637712696953
```

In [45]:

```
#For model building using OLS Library
X = sm.add_constant(X)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
# Fit an OLS model using statsmodels to the training data
ols_model = sm.OLS(y_train, sm.add_constant(x_train)).fit()
```

Model Performance Check

In [46]:

```
# 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
```

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

Training Performance

Out[46]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.231771	0.180182	0.8453	0.842077	4.317485

In [47]:

```python
olsmodel_test_perf = model_performance_regression(ols_model, x_test, y_test)
olsmodel_test_perf
```

Out[47]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.23147	0.180652	0.837164	0.829022	4.357254

In [48]:

```python
# Display OLS model summary
print("\nOLS Model Summary:")
print(ols_model.summary())
```

```
OLS Model 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:                     267.9
Date:                   Mon, 18 Sep 2023   Prob (F-statistic):               0.00
Time:                           17:51:35   Log-Likelihood:                 103.44
No. Observations:                   2402   AIC:                            -108.9
Df Residuals:                       2353   BIC:                             174.5
Df Model:                             48
Covariance Type:               nonrobust
==================================================================================
====
                           coef    std err          t      P>|t|      [0.025      0.
975]
----------------------------------------------------------------------------------
----
const                    4.3604      0.041    107.326      0.000       4.281
4.440
screen_size              0.0900      0.013      6.892      0.000       0.064
0.116
main_camera_mp           0.0956      0.007     13.125      0.000       0.081
0.110
selfie_camera_mp         0.0964      0.008     12.077      0.000       0.081
0.112
int_memory               0.0051      0.005      1.008      0.313      -0.005
0.015
ram                      0.0319      0.007      4.612      0.000       0.018
```

```
0.046
battery                  -0.0168      0.010     -1.698      0.090      -0.036
0.003
weight                    0.0832      0.012      7.171      0.000       0.060
0.106
release_year              0.0637      0.011      5.958      0.000       0.043
0.085
days_used                 0.0136      0.008      1.779      0.075      -0.001
0.029
normalized_new_price      0.2930      0.008     34.663      0.000       0.276
0.310
brand_name_Alcatel       -0.0096      0.047     -0.204      0.838      -0.102
0.082
brand_name_Apple         -0.0121      0.148     -0.082      0.935      -0.302
0.278
brand_name_Asus           0.0278      0.047      0.591      0.555      -0.064
0.120
brand_name_BlackBerry     0.0248      0.070      0.354      0.723      -0.112
0.162
brand_name_Celkon        -0.2030      0.067     -3.016      0.003      -0.335        -
0.071
brand_name_Coolpad        0.0232      0.071      0.326      0.745      -0.116
0.163
brand_name_Gionee        -0.0243      0.057     -0.427      0.669      -0.136
0.087
brand_name_Google        -0.0531      0.098     -0.544      0.587      -0.245
0.138
brand_name_HTC            0.0279      0.048      0.576      0.565      -0.067
0.123
brand_name_Honor          0.0272      0.049      0.558      0.577      -0.068
0.123
brand_name_Huawei        -0.0264      0.044     -0.604      0.546      -0.112
0.059
brand_name_Infinix        0.1078      0.089      1.205      0.228      -0.068
0.283
brand_name_Karbonn       -0.0058      0.065     -0.089      0.929      -0.133
0.122
brand_name_LG            -0.0382      0.045     -0.855      0.393      -0.126
0.049
brand_name_Lava          -0.0347      0.064     -0.546      0.585      -0.159
0.090
brand_name_Lenovo         0.0356      0.045      0.793      0.428      -0.052
0.124
brand_name_Meizu          0.0046      0.056      0.082      0.935      -0.106
0.115
brand_name_Micromax      -0.0172      0.048     -0.359      0.720      -0.112
0.077
brand_name_Microsoft      0.0636      0.087      0.728      0.467      -0.108
0.235
brand_name_Motorola      -0.0246      0.048     -0.509      0.611      -0.119
0.070
brand_name_Nokia          0.0882      0.051      1.736      0.083      -0.011
0.188
brand_name_OnePlus        0.0388      0.074      0.526      0.599      -0.106
0.184
brand_name_Oppo           0.0212      0.047      0.449      0.654      -0.071
0.114
brand_name_Others        -0.0254      0.041     -0.616      0.538      -0.106
0.056
brand_name_Panasonic      0.0044      0.055      0.079      0.937      -0.104
0.113
brand_name_Realme         0.0478      0.062      0.774      0.439      -0.073
0.169
brand_name_Samsung       -0.0297      0.042     -0.699      0.485      -0.113
0.054
brand_name_Sony          -0.0299      0.051     -0.580      0.562      -0.131
0.071
brand_name_Spice         -0.0441      0.063     -0.697      0.486      -0.168
0.080
```

```
brand_name_Vivo             -0.0289      0.048     -0.600      0.548      -0.123
0.066
brand_name_XOLO             -0.0531      0.060     -0.884      0.377      -0.171
0.065
brand_name_Xiaomi            0.0856      0.047      1.823      0.068      -0.006
0.178
brand_name_ZTE              -0.0026      0.047     -0.055      0.956      -0.094
0.089
os_Others                   -0.0460      0.032     -1.430      0.153      -0.109
0.017
os_Windows                  -0.0287      0.046     -0.627      0.531      -0.119
0.061
os_iOS                      -0.0671      0.147     -0.456      0.649      -0.356
0.222
4g_yes                       0.0285      0.016      1.753      0.080      -0.003
0.060
5g_yes                      -0.0398      0.031     -1.283      0.200      -0.101
0.021
==============================================================================
Omnibus:                      206.737   Durbin-Watson:                   2.041
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              473.247
Skew:                          -0.525   Prob(JB):                    1.72e-103
Kurtosis:                       4.905   Cond. No.                         103.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

Checking Linear Regression Assumptions

- In order to make statistical inferences from a linear regression model, it is important to ensure that the assumptions of linear regression are satisfied.

1. TEST FOR MULTICOLLINEARITY

In [49]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [50]:
```
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 [51]:
```
checking_vif(x_train)
```

Out[51]:

	feature	VIF
0	const	72.300471
1	screen_size	7.498516
2	main_camera_mp	2.331871
3	selfie_camera_mp	2.861443
4	int_memory	1.259827
5	ram	2.223959

	feature	VIF
6	battery	4.187287
7	weight	5.817094
8	release_year	5.057926
9	days_used	2.622416
10	normalized_new_price	3.178215
11	brand_name_Alcatel	3.327137
12	brand_name_Apple	13.729323
13	brand_name_Asus	3.338693
14	brand_name_BlackBerry	1.595714
15	brand_name_Celkon	1.718911
16	brand_name_Coolpad	1.469013
17	brand_name_Gionee	1.923613
18	brand_name_Google	1.214968
19	brand_name_HTC	3.027913
20	brand_name_Honor	3.192975
21	brand_name_Huawei	5.793516
22	brand_name_Infinix	1.309676
23	brand_name_Karbonn	1.608783
24	brand_name_LG	4.755392
25	brand_name_Lava	1.678174
26	brand_name_Lenovo	4.191599
27	brand_name_Meizu	2.049853
28	brand_name_Micromax	3.101260
29	brand_name_Microsoft	1.801805
30	brand_name_Motorola	3.259563
31	brand_name_Nokia	3.157554
32	brand_name_OnePlus	1.481584
33	brand_name_Oppo	3.788902
34	brand_name_Others	9.249734
35	brand_name_Panasonic	2.044338
36	brand_name_Realme	1.859573
37	brand_name_Samsung	7.105487
38	brand_name_Sony	2.598120
39	brand_name_Spice	1.668308
40	brand_name_Vivo	3.469702
41	brand_name_XOLO	1.755842

	feature	VIF
42	brand_name_Xiaomi	3.892557
43	brand_name_ZTE	3.629829
44	os_Others	1.753029
45	os_Windows	1.687993
46	os_iOS	12.471348
47	4g_yes	2.528553
48	5g_yes	1.823114

Removing Multicollinearity

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

In [52]:
```
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 [53]:
```
col_list = ["screen_size", "weight",'release_year']

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

Out[53]:

	col	Adj. R-squared after_dropping col	RMSE after dropping col
0	release_year	0.839831	0.235882
1	screen_size	0.839026	0.236474
2	weight	0.838763	0.236667

In [54]:
```
col_to_drop = "release_year"
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  release_year
```

Out[54]:

	feature	VIF
0	const	71.411182
1	screen_size	7.271866
2	main_camera_mp	2.315230
3	selfie_camera_mp	2.499934
4	int_memory	1.252987
5	ram	2.223497
6	battery	4.060436
7	weight	5.642909
8	days_used	1.907735
9	normalized_new_price	2.935728
10	brand_name_Alcatel	3.326997
11	brand_name_Apple	13.709035
12	brand_name_Asus	3.338676
13	brand_name_BlackBerry	1.593636
14	brand_name_Celkon	1.710506
15	brand_name_Coolpad	1.468586
16	brand_name_Gionee	1.923497
17	brand_name_Google	1.210364
18	brand_name_HTC	3.027057
19	brand_name_Honor	3.191263
20	brand_name_Huawei	5.791862
21	brand_name_Infinix	1.309044

	feature	VIF
22	brand_name_Karbonn	1.602210
23	brand_name_LG	4.754916
24	brand_name_Lava	1.677378
25	brand_name_Lenovo	4.191432
26	brand_name_Meizu	2.048604
27	brand_name_Micromax	3.101184
28	brand_name_Microsoft	1.794573
29	brand_name_Motorola	3.258093
30	brand_name_Nokia	3.141847
31	brand_name_OnePlus	1.480085
32	brand_name_Oppo	3.788425
33	brand_name_Others	9.247318
34	brand_name_Panasonic	2.042476
35	brand_name_Realme	1.853383
36	brand_name_Samsung	7.104963
37	brand_name_Sony	2.596159
38	brand_name_Spice	1.664232
39	brand_name_Vivo	3.469683
40	brand_name_XOLO	1.754007
41	brand_name_Xiaomi	3.892498
42	brand_name_ZTE	3.629705
43	os_Others	1.752492
44	os_Windows	1.675745
45	os_iOS	12.468001
46	4g_yes	2.153915
47	5g_yes	1.795974

In [55]:
```
col_to_drop = "screen_size"
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  screen_size
```

Out[55]:

	feature	VIF
0	const	72.087315
1	main_camera_mp	2.331683

	feature	VIF
2	selfie_camera_mp	2.859580
3	int_memory	1.255028
4	ram	2.223878
5	battery	3.825789
6	weight	2.887418
7	release_year	4.905045
8	days_used	2.611625
9	normalized_new_price	3.123210
10	brand_name_Alcatel	3.326538
11	brand_name_Apple	13.671312
12	brand_name_Asus	3.335371
13	brand_name_BlackBerry	1.592941
14	brand_name_Celkon	1.718418
15	brand_name_Coolpad	1.468493
16	brand_name_Gionee	1.921712
17	brand_name_Google	1.212837
18	brand_name_HTC	3.023025
19	brand_name_Honor	3.192400
20	brand_name_Huawei	5.792632
21	brand_name_Infinix	1.309117
22	brand_name_Karbonn	1.608500
23	brand_name_LG	4.744400
24	brand_name_Lava	1.678022
25	brand_name_Lenovo	4.190931
26	brand_name_Meizu	2.048188
27	brand_name_Micromax	3.100332
28	brand_name_Microsoft	1.801719
29	brand_name_Motorola	3.250320
30	brand_name_Nokia	3.149734
31	brand_name_OnePlus	1.481564
32	brand_name_Oppo	3.784694
33	brand_name_Others	9.205279
34	brand_name_Panasonic	2.044119
35	brand_name_Realme	1.858631
36	brand_name_Samsung	7.095542
37	brand_name_Sony	2.595552

	feature	VIF
38	brand_name_Spice	1.667061
39	brand_name_Vivo	3.469626
40	brand_name_XOLO	1.755176
41	brand_name_Xiaomi	3.888491
42	brand_name_ZTE	3.626871
43	os_Others	1.525762
44	os_Windows	1.687878
45	os_iOS	12.362523
46	4g_yes	2.527305
47	5g_yes	1.819338

- Now that the VIF score of all continuous variables is under 5, it can be said that multi-collinearity has been mitigated
- Let's check the performance of our model now

In [56]:
```
ols_model2 = sm.OLS(y_train, sm.add_constant(x_train2)).fit()

# Display OLS model summary
print("\nOLS Model Summary:")
print(ols_model2.summary())
```

```
OLS Model Summary:
                              OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.842
Model:                               OLS   Adj. R-squared:                  0.839
Method:                    Least Squares   F-statistic:                     267.3
Date:                   Mon, 18 Sep 2023   Prob (F-statistic):               0.00
Time:                           17:51:35   Log-Likelihood:                 79.440
No. Observations:                   2402   AIC:                            -62.88
Df Residuals:                       2354   BIC:                             214.8
Df Model:                             47
Covariance Type:               nonrobust
==================================================================================
====
                         coef    std err          t      P>|t|      [0.025      0.
975]
----------------------------------------------------------------------------------
----
const                  4.3756      0.041    106.809      0.000       4.295
4.456
main_camera_mp         0.0960      0.007     13.059      0.000       0.082
0.110
selfie_camera_mp       0.0978      0.008     12.138      0.000       0.082
0.114
int_memory             0.0030      0.005      0.578      0.563      -0.007
0.013
ram                    0.0322      0.007      4.608      0.000       0.019
0.046
battery                0.0032      0.010      0.339      0.735      -0.015
0.022
weight                 0.1400      0.008     16.954      0.000       0.124
0.156
release_year           0.0765      0.011      7.196      0.000       0.056
0.097
days_used              0.0102      0.008      1.327      0.185      -0.005
```

```
0.025
normalized_new_price      0.3007      0.008     35.532      0.000       0.284
0.317
brand_name_Alcatel       -0.0052      0.047     -0.111      0.912      -0.098
0.088
brand_name_Apple          0.0541      0.149      0.363      0.716      -0.238
0.346
brand_name_Asus           0.0175      0.047      0.370      0.711      -0.075
0.111
brand_name_BlackBerry     0.0047      0.071      0.066      0.947      -0.134
0.143
brand_name_Celkon        -0.2108      0.068     -3.103      0.002      -0.344      -
0.078
brand_name_Coolpad        0.0139      0.072      0.194      0.846      -0.127
0.155
brand_name_Gionee        -0.0366      0.057     -0.638      0.524      -0.149
0.076
brand_name_Google        -0.0813      0.099     -0.825      0.409      -0.275
0.112
brand_name_HTC            0.0145      0.049      0.296      0.767      -0.081
0.110
brand_name_Honor          0.0317      0.049      0.644      0.520      -0.065
0.128
brand_name_Huawei        -0.0301      0.044     -0.683      0.495      -0.117
0.056
brand_name_Infinix        0.0951      0.090      1.052      0.293      -0.082
0.272
brand_name_Karbonn        0.0002      0.066      0.003      0.998      -0.129
0.129
brand_name_LG            -0.0529      0.045     -1.176      0.240      -0.141
0.035
brand_name_Lava          -0.0389      0.064     -0.606      0.545      -0.165
0.087
brand_name_Lenovo         0.0317      0.045      0.699      0.485      -0.057
0.121
brand_name_Meizu         -0.0065      0.057     -0.114      0.910      -0.118
0.105
brand_name_Micromax      -0.0230      0.049     -0.473      0.636      -0.118
0.072
brand_name_Microsoft      0.0595      0.088      0.674      0.501      -0.114
0.233
brand_name_Motorola      -0.0423      0.049     -0.868      0.385      -0.138
0.053
brand_name_Nokia          0.0708      0.051      1.381      0.168      -0.030
0.171
brand_name_OnePlus        0.0370      0.075      0.496      0.620      -0.109
0.183
brand_name_Oppo           0.0104      0.048      0.217      0.828      -0.083
0.104
brand_name_Others        -0.0452      0.042     -1.086      0.278      -0.127
0.036
brand_name_Panasonic      0.0004      0.056      0.008      0.994      -0.109
0.110
brand_name_Realme         0.0382      0.062      0.613      0.540      -0.084
0.161
brand_name_Samsung       -0.0406      0.043     -0.948      0.343      -0.125
0.043
brand_name_Sony          -0.0410      0.052     -0.790      0.430      -0.143
0.061
brand_name_Spice         -0.0561      0.064     -0.877      0.381      -0.182
0.069
brand_name_Vivo          -0.0305      0.049     -0.626      0.531      -0.126
0.065
brand_name_XOLO          -0.0612      0.061     -1.009      0.313      -0.180
0.058
brand_name_Xiaomi         0.0752      0.047      1.585      0.113      -0.018
0.168
brand_name_ZTE           -0.0118      0.047     -0.249      0.803      -0.104
0.081
```

```
os_Others                -0.1257      0.030     -4.153      0.000      -0.185      -
0.066
os_Windows               -0.0313      0.046     -0.677      0.498      -0.122
0.059
os_iOS                   -0.1618      0.148     -1.094      0.274      -0.452
0.128
4g_yes                    0.0260      0.016      1.585      0.113      -0.006
0.058
5g_yes                   -0.0495      0.031     -1.583      0.114      -0.111
0.012
==============================================================================
Omnibus:                      201.803   Durbin-Watson:                   2.034
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              441.511
Skew:                          -0.527   Prob(JB):                     1.34e-96
Kurtosis:                       4.817   Cond. No.                         97.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

Dealing with high p-value variables

- 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

In [57]:

```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
```

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

```
['const', 'main_camera_mp', 'selfie_camera_mp', 'ram', 'weight', 'release_year', 'nor
malized_new_price', 'brand_name_Celkon', 'brand_name_Lenovo', 'brand_name_Nokia', 'br
and_name_Xiaomi', 'os_Others', '4g_yes']
```

In [58]:
```
x_train3 = x_train2[selected_features]
x_test3 = x_test2[selected_features]
```

In [59]:
```
ols_model3 = sm.OLS(y_train, x_train3).fit()
print(ols_model3.summary())
```

```
                            OLS Regression Results
=================================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.840
Model:                               OLS   Adj. R-squared:                  0.839
Method:                    Least Squares   F-statistic:                     1044.
Date:                   Mon, 18 Sep 2023   Prob (F-statistic):               0.00
Time:                           17:51:36   Log-Likelihood:                 61.895
No. Observations:                   2402   AIC:                            -97.79
Df Residuals:                       2389   BIC:                            -22.60
Df Model:                             12
Covariance Type:               nonrobust
=====================================================================================
===
                         coef    std err          t      P>|t|      [0.025      0.9
75]
-------------------------------------------------------------------------------------
---
const                  4.3453      0.012    372.326      0.000       4.322       4.
368
main_camera_mp         0.1021      0.007     15.461      0.000       0.089       0.
115
selfie_camera_mp       0.1002      0.008     13.267      0.000       0.085       0.
115
ram                    0.0284      0.006      4.608      0.000       0.016       0.
040
weight                 0.1420      0.005     25.931      0.000       0.131       0.
153
release_year           0.0697      0.008      8.938      0.000       0.054       0.
085
normalized_new_price   0.2946      0.008     39.117      0.000       0.280       0.
309
brand_name_Celkon     -0.1904      0.054     -3.529      0.000      -0.296      -0.
085
brand_name_Lenovo      0.0574      0.022      2.562      0.010       0.013       0.
101
brand_name_Nokia       0.0819      0.030      2.702      0.007       0.022       0.
141
brand_name_Xiaomi      0.0930      0.025      3.790      0.000       0.045       0.
141
os_Others             -0.1294      0.028     -4.685      0.000      -0.184      -0.
075
4g_yes                 0.0326      0.015      2.109      0.035       0.002       0.
063
==============================================================================
Omnibus:                      203.173   Durbin-Watson:                   2.031
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              454.850
Skew:                          -0.523   Prob(JB):                     1.70e-99
Kurtosis:                       4.857   Cond. No.                         19.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

1. TEST FOR LINEARITY AND INDEPENDENCE

- Predictor variables must have a linear relation with the dependent variable
- Error terms must be independent

In [60]:
```
# 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"] = ols_model3.fittedvalues  # predicted values
df_pred["Residuals"] = ols_model3.resid  # residuals

df_pred.head()
```

Out[60]:

	Actual Values	Fitted Values	Residuals
683	4.824225	4.776353	0.047872
3364	4.145354	4.326045	-0.180691
2313	3.774139	3.729749	0.044390
1569	4.433907	4.793494	-0.359587
1169	4.553772	4.499421	0.054350

In [61]:
```
# let's plot the fitted values vs residuals

plt.figure(figsize=(15, 8))
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 residuals don't show any trend with the fitted values, hence they can be assumed to be independent. Absence of any pattern also indicates that our assumption of predictor variables having linear relationship with response variable is valid

1. TEST FOR NORMALITY

- 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.

In [62]:
```
plt.figure(figsize=(15, 8))
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 [63]:
```
import pylab
import scipy.stats as stats
```

In [64]:
```python
plt.figure(figsize=(15, 8))
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 [65]:
```python
stats.shapiro(df_pred["Residuals"])
```

Out[65]:
```
ShapiroResult(statistic=0.9678758978843689, pvalue=9.480866098147382e-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.

1. TEST FOR HOMOSCEDASTICITY

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

In [66]:
```python
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
```

In [67]:
```python
name = ["F statistic", "p-value"]
test = sms.het_goldfeldquandt(df_pred["Residuals"], x_train2)
lzip(name, test)
```

Out[67]:
```
[('F statistic', 1.0101084835088836), ('p-value', 0.4322211423858719)]
```

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

Final Model

In [68]:
```python
# predictions on the test set
pred = ols_model3.predict(x_test3)

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

Out[68]:

	Actual	Predicted
984	4.204095	4.125319
1230	4.061649	4.174662
3182	5.452411	5.371356
2396	4.338336	4.371746
2008	4.251206	4.013936
1373	4.898437	5.066077
1896	2.753024	2.738016

	Actual	Predicted
1816	4.104130	4.186352
1728	4.415945	4.474357
1872	4.021953	3.862274

In [69]:

```
x_train_final = x_train3.copy()
x_test_final = x_test3.copy()

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

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     normalized_used_price   R-squared:                       0.840
Model:                               OLS   Adj. R-squared:                  0.839
Method:                    Least Squares   F-statistic:                     1044.
Date:                   Mon, 18 Sep 2023   Prob (F-statistic):               0.00
Time:                           17:51:37   Log-Likelihood:                 61.895
No. Observations:                   2402   AIC:                            -97.79
Df Residuals:                       2389   BIC:                            -22.60
Df Model:                             12
Covariance Type:               nonrobust
==================================================================================
===
                          coef    std err          t      P>|t|      [0.025      0.9
75]
----------------------------------------------------------------------------------
---
const                   4.3453      0.012    372.326      0.000       4.322       4.
368
main_camera_mp          0.1021      0.007     15.461      0.000       0.089       0.
115
selfie_camera_mp        0.1002      0.008     13.267      0.000       0.085       0.
115
ram                     0.0284      0.006      4.608      0.000       0.016       0.
040
weight                  0.1420      0.005     25.931      0.000       0.131       0.
153
release_year            0.0697      0.008      8.938      0.000       0.054       0.
085
normalized_new_price    0.2946      0.008     39.117      0.000       0.280       0.
309
brand_name_Celkon      -0.1904      0.054     -3.529      0.000      -0.296      -0.
085
brand_name_Lenovo       0.0574      0.022      2.562      0.010       0.013       0.
101
brand_name_Nokia        0.0819      0.030      2.702      0.007       0.022       0.
141
brand_name_Xiaomi       0.0930      0.025      3.790      0.000       0.045       0.
141
os_Others              -0.1294      0.028     -4.685      0.000      -0.184      -0.
075
4g_yes                  0.0326      0.015      2.109      0.035       0.002       0.
063
==============================================================================
Omnibus:                      203.173   Durbin-Watson:                   2.031
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              454.850
Skew:                          -0.523   Prob(JB):                     1.70e-99
Kurtosis:                       4.857   Cond. No.                         19.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

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

Training Performance

Out[70]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.235815	0.183395	0.839855	0.838983	4.401094

In [71]:
```
# checking model performance on test set (seen 30% data)
print("Test Performance")
olsmodel_final_test_perf = model_performance_regression(
    olsmodel_final, x_test_final, y_test
)
olsmodel_final_test_perf
```

Test Performance

Out[71]:

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.234808	0.183584	0.832434	0.83029	4.439329

Actionable Insights and Recommendations

- The regression model has R-squared score of 0.8324 on unseen data indicating that approximately 83.24% of the variance in the target variable is explained by the model's features
- An adjusted R-squared of 0.8303 is still quite high on unseen data, indicating that the model's performance remains strong even after adjusting for the number of predictors
- Feature engineering can be used to create new features that could improve the model
- We left the outliers as they were. Perhaps a model can be tried where we make changes to data to remove outliers
- Techniques such as cross-validation can be used to validate the model's predictive performance on unseen data
- Looking at the coefficients allotted by regression model, normalized_new_price is a very important variable. For unit increase in new_price, old_price goes up by 0.29 units
- The coefficient of approximately -0.1294 for os_others suggests that devices with an operating system categorized as "Others" (compared to Android, Windows, iOS) are associated with an average decrease of approximately 0.1294 units in the normalized_used_price.
- More data should be collected for OS other than Android devices
- Other regression models can be tried as well
- Data engineers can look for more parameters such as update frequency
- Features such as camera Mega-Pixels, RAM are significant and have positive coefficients. Companies should pay special attention to these features to increase or decrease price of their devices as per the needs

---