# Prediction of Used Phone and Tablet Price


## Siddharth Nilakhe

## Prof. Hadi Safari Katesari



### Abstract

This project aims to predict the price of used and refurbished devices by considering various features in the dataset. The dataset contains features such as device brand, OS, screen size, 4G/5G availability, camera resolution, internal memory, RAM, battery capacity, weight, release year, days used, and new price. The target variable is the used price of the device. The project aims to identify the features that are responsible for deciding the price of the device. By predicting the used price of the device, consumers and businesses can make informed decisions while purchasing used or refurbished devices, which not only saves them money but also reduces environmental impact and helps in recycling and reducing waste.

This project will utilize various statistical tests to analyze the data. These tests include comparing two samples, the analysis of variance, the analysis of categorical data, linear regression, resampling methods, linear model selection, and regularization, and moving beyond linearity. The results of these tests will be used to identify the features that have the strongest influence on the used device price.


# Chapter 1

### Introduction

The used and refurbished device market has gained significant traction in recent years as consumers and businesses look for cost-effective alternatives to purchasing new devices. With the constant evolution of technology and frequent new device releases, the demand for used and refurbished devices has increased as a means to save money while still being able to enjoy modern technology. Additionally, buying used devices is an environmentally conscious choice, as it reduces the impact on the environment by extending the life cycle of a device and keeping it out of landfills.

The secondary device market provides a valuable service to consumers, offering them the opportunity to purchase devices that they may not have been able to afford otherwise. However, the market can be complex, with a range of factors influencing the price of a device, including brand, model, age, and condition. Therefore, predicting the price of a used or refurbished device can be challenging, requiring an understanding of the factors that influence the value of a device.

In this project, we aim to predict the price of used and refurbished devices by analyzing various features in the dataset, such as device brand, OS, screen size, 4G/5G availability, camera resolution, internal memory, RAM, battery capacity, weight, release year, days used, and new price. By identifying the features that are most influential in determining the price of a used or refurbished device, we hope to provide valuable insights to consumers and businesses seeking to purchase used devices. Additionally, our analysis will contribute to reducing waste and promoting sustainable practices by encouraging the use of used and refurbished devices.

# Chapter 2

### Data Description

Features of the dataset:

1)device_brand: Name of manufacturing brand
2)os: OS on which the device runs
3)screen_size: Size of the screen in cm
4)4g: Whether 4G is available or not
5)5g: Whether 5G is available or not
6)front_camera_mp: Resolution of the rear camera in megapixels
7)back_camera_mp: Resolution of the front camera in megapixels
8)internal_memory: Amount of internal memory (ROM) in GB
9)ram: Amount of RAM in GB
10)battery: Energy capacity of the device battery in mAh
11)weight: Weight of the device in grams
12)release_year: Year when the device model was released
13)days_used: Number of days the used/refurbished device has been used
14)new_price: Price of a new device of the same model

Target Variable:
used_price (TARGET): Price of the used/refurbished device

This dataset includes various features related to used and refurbished devices, such as device brand, operating system, screen size, camera resolution, internal memory, RAM, battery capacity, weight, release year, and number of days used. The target variable is the price of the used or refurbished device. This dataset can be used to predict the price of a used or refurbished device based on its features and to identify which features are most influential in determining the price of the device. By analyzing this data, businesses and consumers can make informed decisions about purchasing used or refurbished devices, reducing waste and promoting sustainable practices.

Dataset Link - https://www.kaggle.com/datasets/ahsan81/used-handheld-device-data

# Chapter 3

## Methodology

**Shapiro Wilk Test**

**The Shapiro-Wilk test is a statistical test used to determine if a sample of data is normally distributed. It calculates a test statistic and a p-value based on the difference between the observed distribution and the expected normal distribution. If the p-value is less than the significance level, the data is considered to be significantly different from normal distribution. It is commonly used in fields where normality assumptions are necessary for further analysis.**

**Multivariate Normality test**

**The multivariate normality test is a statistical test to check if a set of variables is normally distributed, taking into account their correlations. The test returns a test statistic, p-value, and Boolean variable indicating if the dataset is multivariate normal or not. In the code example provided, the test is conducted using the Pingouin library, and the results are printed based on a significance level of 0.05.**

$$HZ = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} e^{-\frac{\beta^2}{2}D_{ij}} - 2\left(1+\beta^2\right)^{-\frac{p}{2}}\sum_{i=1}^{n} e^{-\frac{\beta^2}{2\left(1+\beta^2\right)}D_i} + n\left(1+2\beta^2\right)^{-\frac{p}{2}}$$

**Z - Test**

**The Z-test is a statistical hypothesis test used to determine whether two population means are different when the population standard deviation is known. It compares a sample mean to the population mean, with the assumption that the sample is normally distributed.**

**The Z-test calculates a test statistic, which measures the distance between the sample mean and the population mean in terms of the standard error. The test statistic is then compared to a critical value from the standard normal distribution, using a specified level of significance. If the test statistic is larger than the critical value, the null hypothesis (that the means are equal) is rejected in favor of the alternative hypothesis (that the means are different).**

**The Z-test is commonly used in quality control, product testing, and other areas where it is important to compare sample means to a known population mean. It is also used as a building block for more complex statistical analyses, such as t-tests and ANOVA.**

**F - Test (One - Way ANOVA)**

**One-way ANOVA (analysis of variance) F-test is a statistical hypothesis test used to determine if there are significant differences between the means of three or more groups. It compares the variance between the means of the groups to the variance within the groups, and determines whether the differences are large enough to be statistically significant.**

**The test involves calculating an F-statistic, which is the ratio of the variance between the means to the variance within the groups. If the F-statistic is larger than the critical value at a given level of significance (usually 0.05), then the null hypothesis (that all group means are equal) is rejected, indicating that at least one group mean is significantly different from the others.**

**One-way ANOVA F-test is commonly used in experimental research, especially in social sciences, to determine whether there are significant differences between the means of different treatments or interventions. It can also be used in quality control and manufacturing to compare the means of different production lines or processes.**

**In order to perform the one-way ANOVA F-test, the data must meet certain assumptions, including normality and homogeneity of variance. If these assumptions are not met, other statistical tests or data transformations may be required.**

**Chi - square Test**

**The chi-square test is a statistical test used to determine if there is a significant association between two categorical variables. It does this by comparing the observed frequencies of the data to the expected frequencies under the assumption that there is no association between the variables. The test calculates a test statistic (chi-square statistic) and a p-value, which is compared to a significance level (alpha) to determine if the null hypothesis can be rejected or not. The null hypothesis states that there is no association between the variables, while the alternative hypothesis states that there is a significant association between the variables.**

$$\chi^2 = \Sigma \frac{(oij - eij)^2}{eij}$$

**Linear Regression**

**Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. It involves fitting a line through the data points to predict the value of the dependent variable based on the values of the independent variables. The line is determined by minimizing the sum of the squared differences between the observed values and the predicted values. Linear regression can be used for both simple and multiple regression problems.**

$$Y = \beta_0 + \beta_1 X + \epsilon$$

**Cross Validation**

**Cross-validation is a statistical technique used to evaluate how well a model will perform on an independent dataset. It involves partitioning the dataset into training and testing sets and then running the model multiple times, each time with a different subset of the data used for training and testing. The results are then averaged to get a more accurate estimate of the model's performance on new data. Cross-validation is commonly used to prevent overfitting and to select the best hyperparameters for a model.**

**PCA**

**PCA (Principal Component Analysis) is a technique used for data reduction and simplification by identifying a smaller number of latent variables, called principal components, that capture most of the variability in the original data. These principal components are derived from a linear combination of the original variables and are orthogonal to each other. PCA is widely used for exploratory data analysis, data compression, feature selection, and visualization.**

**Ridge Regression**

**Ridge Regression is a regularization technique used in linear regression to prevent overfitting. It involves adding a penalty term to the least-squares objective function, which forces the coefficients of the model to be small. This penalty term is controlled by a hyperparameter called the regularization parameter or lambda. Ridge regression shrinks the coefficients towards zero but does not eliminate them entirely, which can improve the performance of the model on new data. Ridge regression is particularly useful when there are many correlated predictors in the data, as it can help to reduce their impact on the model.**

**Lasso Regression**

**Lasso Regression is a type of linear regression that performs variable selection and regularization by shrinking the coefficients of less important variables to zero. This helps in preventing overfitting and improving the predictive accuracy of the model. The method uses L1 regularization technique, which adds a penalty term equal to the absolute value of the coefficients, to the linear regression cost function. This leads to the elimination of less important variables and selection of only the most important ones.**

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder
from scipy.stats import shapiro
import pingouin as pg
from scipy.stats import kruskal
from statsmodels.stats.weightstats import ztest
from scipy.stats import mannwhitneyu
from sklearn.metrics import mean_squared_error
from sklearn.utils import resample
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
```

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

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

Out[3]:

| | device_brand | os | screen_size | 4g | 5g | rear_camera_mp | front_camera_mp | internal_memory | ram | battery | weight | release_year | da |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | |

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

Out[4]:

| | screen_size | rear_camera_mp | front_camera_mp | internal_memory | ram | battery | weight | release_year | days_used |
|---|---|---|---|---|---|---|---|---|---|
| count | 3454.000000 | 3275.000000 | 3452.000000 | 3450.000000 | 3450.000000 | 3448.000000 | 3447.000000 | 3454.000000 | 3454.000000 |
| mean | 13.713115 | 9.460208 | 6.554229 | 54.573099 | 4.036122 | 3133.402697 | 182.751871 | 2015.965258 | 674.869716 |
| std | 3.805280 | 4.815461 | 6.970372 | 84.972371 | 1.365105 | 1299.682844 | 88.413228 | 2.298455 | 248.580166 |
| min | 5.080000 | 0.080000 | 0.000000 | 0.010000 | 0.020000 | 500.000000 | 69.000000 | 2013.000000 | 91.000000 |
| 25% | 12.700000 | 5.000000 | 2.000000 | 16.000000 | 4.000000 | 2100.000000 | 142.000000 | 2014.000000 | 533.500000 |
| 50% | 12.830000 | 8.000000 | 5.000000 | 32.000000 | 4.000000 | 3000.000000 | 160.000000 | 2015.500000 | 690.500000 |
| 75% | 15.340000 | 13.000000 | 8.000000 | 64.000000 | 4.000000 | 4000.000000 | 185.000000 | 2018.000000 | 868.750000 |
| max | 30.710000 | 48.000000 | 32.000000 | 1024.000000 | 12.000000 | 9720.000000 | 855.000000 | 2020.000000 | 1094.000000 |

In [5]:
```
df.shape
```

Out[5]:
```
(3454, 15)
```

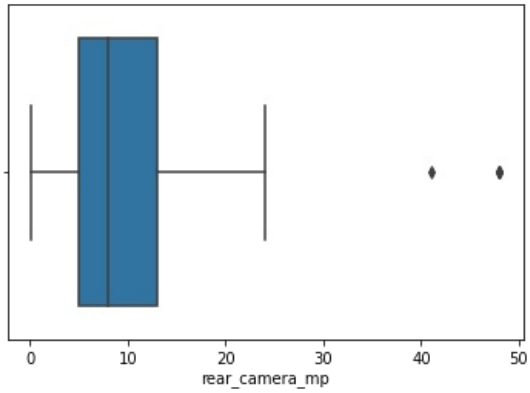
In [6]:
```
df.rename(columns={'4g':'Includes_4g'},inplace=True)
df.rename(columns={'5g':'Includes_5g'},inplace=True)
```

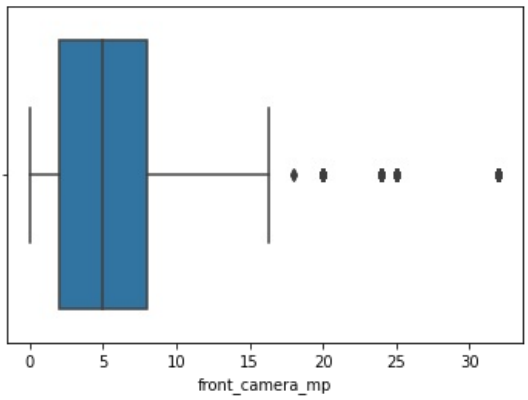
In [7]:
```
df.isnull().sum()
```

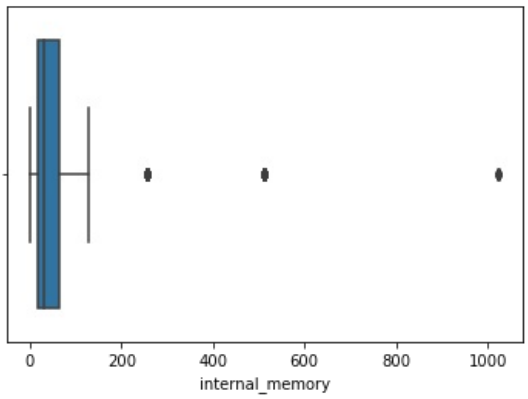
Out[7]:
```
device_brand         0
os                   0
screen_size          0
Includes_4g          0
Includes_5g          0
rear_camera_mp     179
front_camera_mp      2
internal_memory      4
ram                  4
battery              6
weight               7
release_year         0
days_used            0
used_price           0
new_price            0
dtype: int64
```

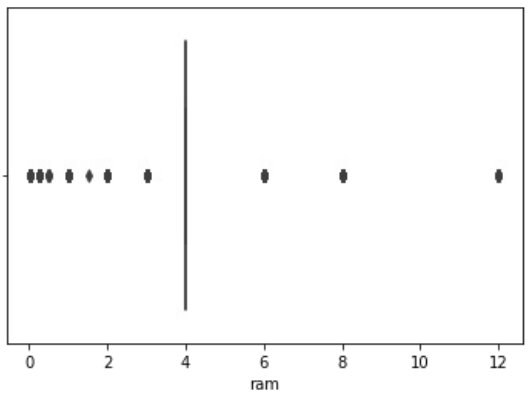
The dataset contains some null values, and removing them is not a viable option due to the small number of samples. Instead, we will handle them appropriately. Before doing so, we will investigate whether there are any outliers in the specified columns.

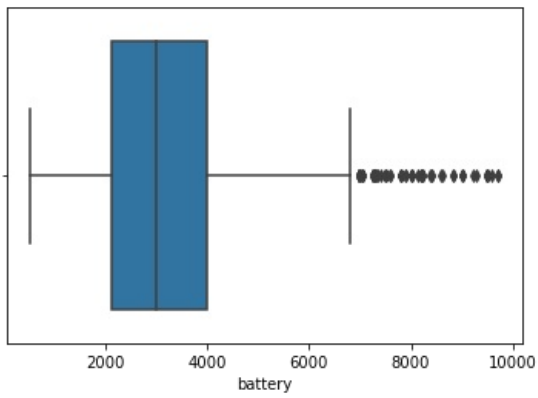
In [8]:
```
import seaborn as sns
import matplotlib.pyplot as plt

features=['rear_camera_mp','front_camera_mp','internal_memory','ram','battery','weight']

for i in features:
    sns.boxplot(df[i])
    plt.show()
```

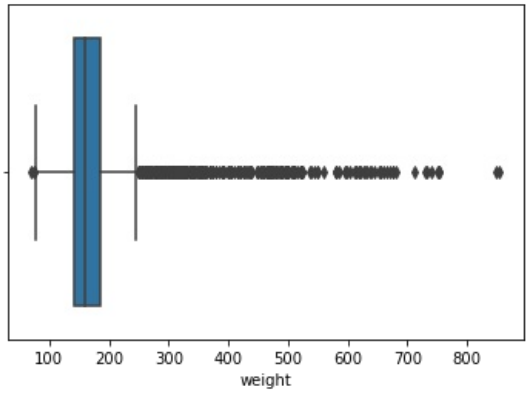

0
10
20
30
40
50
rear_camera_mp


0
5
10
15
20
25
30
front_camera_mp


0
200
400
600
800
1000
internal_memory


0
2
4
6
8
10
12
ram


2000
4000
6000
8000
10000
battery




**It has been observed that the data contains outliers.**
**It is not advisable to replace the null values with mean values in the presence of outliers, as the mean is highly sensitive to outliers and may significantly alter the central tendency of the data.**
**Hence, the median is preferred over the mean when the data contains outliers.**

In [9]:
```
for i in features:
    median = df[i].median()
    df[i]=df[i].fillna(median)
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
device_brand       0
os                 0
screen_size        0
Includes_4g        0
Includes_5g        0
rear_camera_mp     0
front_camera_mp    0
internal_memory    0
ram                0
battery            0
weight             0
release_year       0
days_used          0
used_price         0
new_price          0
dtype: int64
```

In [11]:
```
df['os'].unique()
```

Out[11]:
```
array(['Android', 'Others', 'iOS', 'Windows'], dtype=object)
```

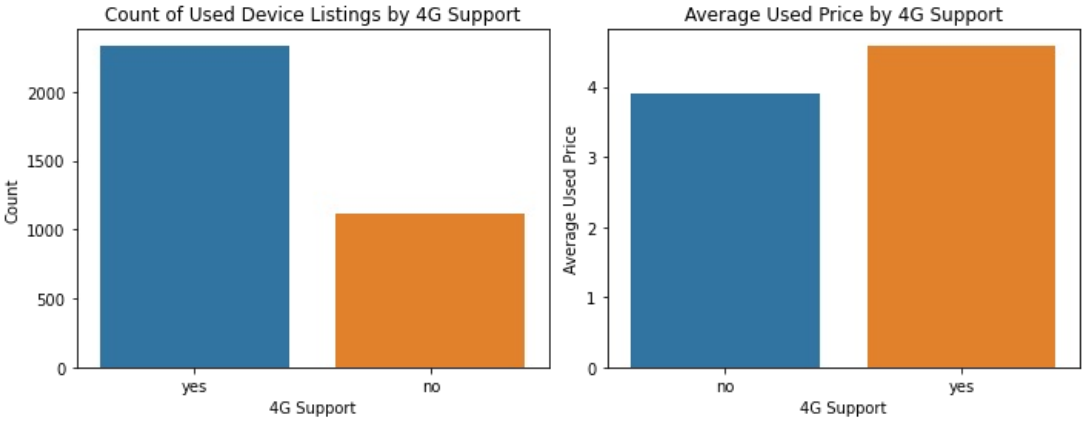
In [12]:
```
df.head()
```

Out[12]:

| | device_brand | os | screen_size | Includes_4g | Includes_5g | rear_camera_mp | front_camera_mp | internal_memory | ram | battery | weight |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Honor | Android | 14.50 | yes | no | 13.0 | 5.0 | 64.0 | 3.0 | 3020.0 | 146.0 |
| 1 | Honor | Android | 17.30 | yes | yes | 13.0 | 16.0 | 128.0 | 8.0 | 4300.0 | 213.0 |
| 2 | Honor | Android | 16.69 | yes | yes | 13.0 | 8.0 | 128.0 | 8.0 | 4200.0 | 213.0 |
| 3 | Honor | Android | 25.50 | yes | yes | 13.0 | 8.0 | 64.0 | 6.0 | 7250.0 | 480.0 |
| 4 | Honor | Android | 15.32 | yes | no | 13.0 | 8.0 | 64.0 | 3.0 | 5000.0 | 185.0 |

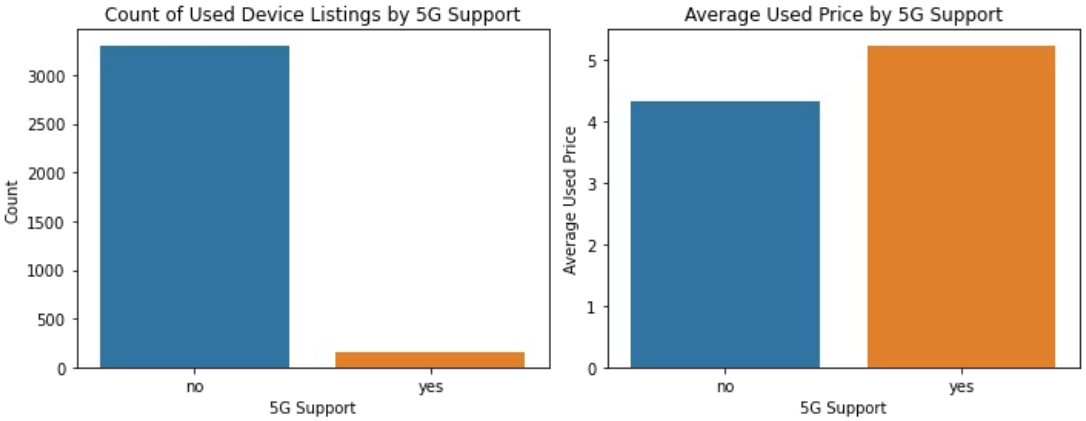
In [13]:
```
# Set the figure size
plt.figure(figsize=(10, 4))

# Create subplot 1 - Count plot of 4G support
plt.subplot(1, 2, 1)
sns.countplot(x='Includes_4g', data=df)
plt.title('Count of Used Device Listings by 4G Support')
plt.xlabel('4G Support')
plt.ylabel('Count')

# Create subplot 2 - Bar plot of average used price by 4G support
plt.subplot(1, 2, 2)
avg_price_by_4g = df.groupby('Includes_4g')['used_price'].mean().reset_index()
sns.barplot(x='Includes_4g', y='used_price', data=avg_price_by_4g)
plt.title('Average Used Price by 4G Support')
plt.xlabel('4G Support')
plt.ylabel('Average Used Price')

# Adjust the spacing between subplots
plt.tight_layout()

# Show the plots
```

```
plt.show()
```



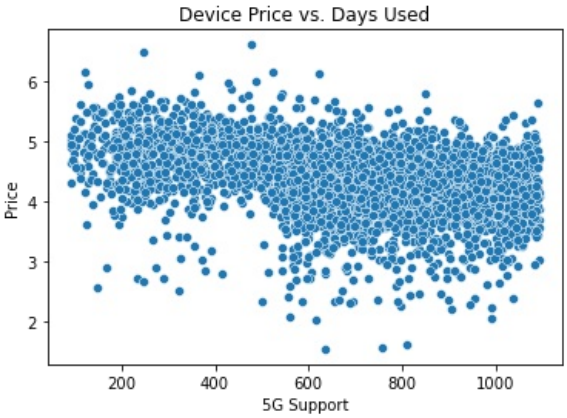

**The visualization helps to understand the relationship between the 4G support and the used price of the devices.**

In [14]:
```
# Set the figure size
plt.figure(figsize=(10, 4))

# Create subplot 1 - Count plot of 5G support
plt.subplot(1, 2, 1)
sns.countplot(x='Includes_5g', data=df)
plt.title('Count of Used Device Listings by 5G Support')
plt.xlabel('5G Support')
plt.ylabel('Count')

# Create subplot 2 - Bar plot of average used price by 5G support
plt.subplot(1, 2, 2)
avg_price_by_5g = df.groupby('Includes_5g')['used_price'].mean().reset_index()
sns.barplot(x='Includes_5g', y='used_price', data=avg_price_by_5g)
plt.title('Average Used Price by 5G Support')
plt.xlabel('5G Support')
plt.ylabel('Average Used Price')

# Adjust the spacing between subplots
plt.tight_layout()

# Show the plots
plt.show()
```



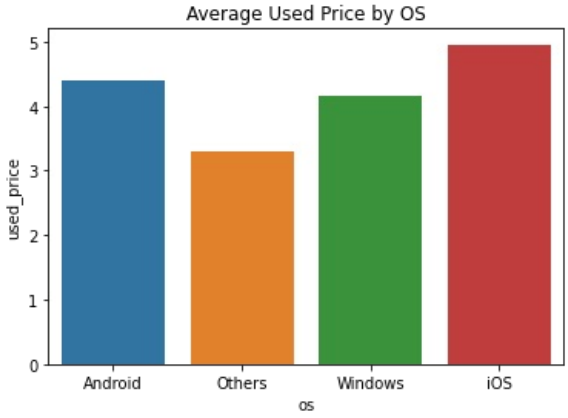

**The visualization helps to understand the relationship between the 5G support and the used price of the devices.**

In [15]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a scatter plot of device price vs. 5G support
sns.scatterplot(x='days_used', y='used_price', data=df)
plt.title('Device Price vs. Days Used')
plt.xlabel('5G Support')
plt.ylabel('Price')
plt.show()
```



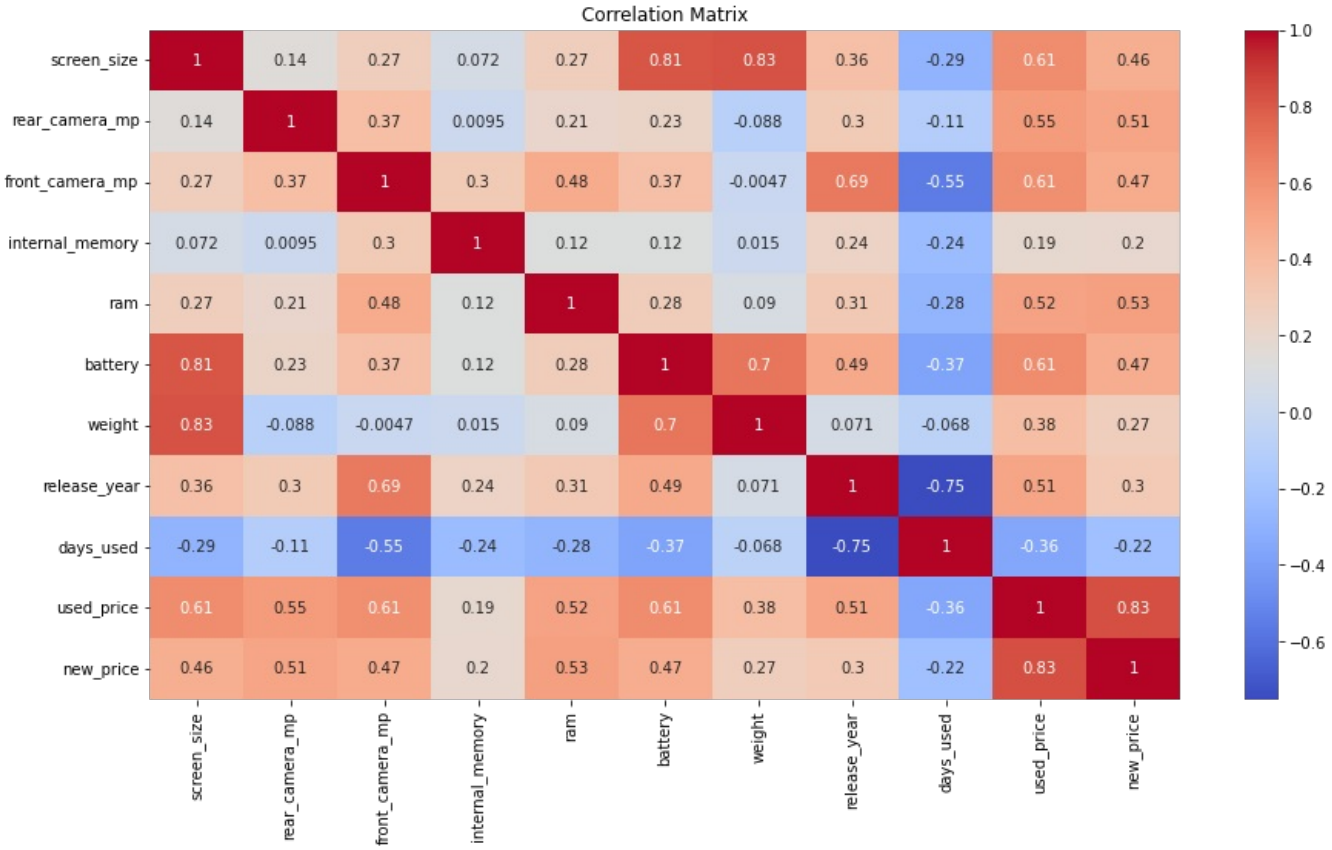

**The above visualization is a scatter plot showing the relationship between the number of days a device has been used and its used price.From the plot, we can observe whether there is any correlation between the number of days a device has been used and its price.**

In [16]:
```
# Calculate the average used price for each OS
avg_price_by_os = df.groupby('os')['used_price'].mean().reset_index()

# Create a barplot of the average used price by OS
sns.barplot(data=avg_price_by_os, x='os', y='used_price')
plt.title('Average Used Price by OS')
plt.show()
```



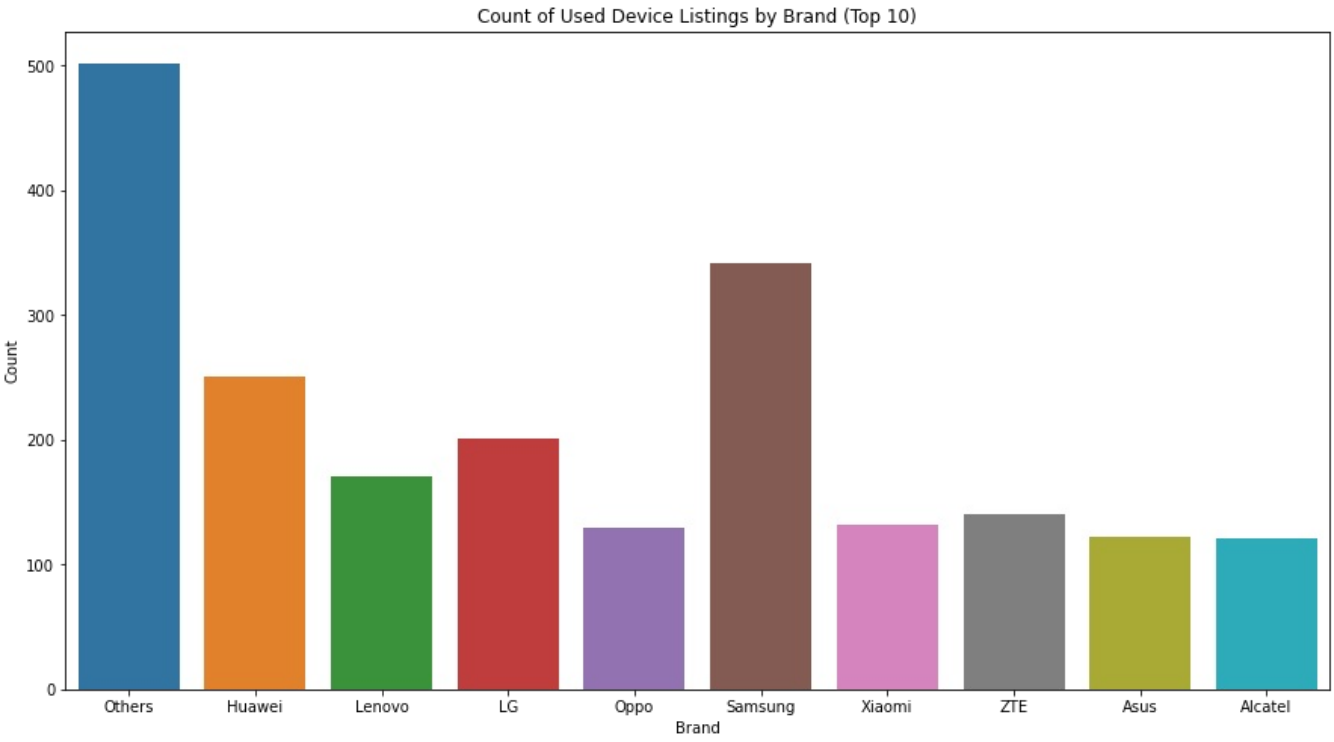

**The plot provides an easy way to compare the average prices across different OS and can help in identifying which OS has higher or lower average prices.**

In [17]:
```
# Compute the correlation matrix
corr = df.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(15, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```




**The factors of rear camera, front camera, battery, RAM, screen size, and new price have a strong correlation with the used device price, making them important in determining the final used device price.**

In [18]:
```
# Get the top 10 brands by count
top_10_brands = df['device_brand'].value_counts().nlargest(10).index.tolist()

# Filter the dataframe to only include the top 10 brands
df_top_10_brands = df[df['device_brand'].isin(top_10_brands)]

# Create the countplot
plt.figure(figsize=(15, 8))
sns.countplot(x='device_brand', data=df_top_10_brands)
plt.title('Count of Used Device Listings by Brand (Top 10)')
plt.xlabel('Brand')
plt.ylabel('Count')
plt.show()
```




**The plot helps in identifying the most popular brands among the used devices listed in the dataset.**

In [19]:
```
encoder = LabelEncoder()
```

```
categorical_features = ['os','Includes_4g','Includes_5g','device_brand']
for i in categorical_features:
    df[i] = encoder.fit_transform(df[i])
```

# Chapter 4

## 4.1 - Comparing Two Sample

**Shapiro Wilk Test**

**H0 - Data is normally distributed**

**H1 - Data is not normally distributed**

In [20]:
```
features = df.columns.tolist()
data = df.copy()

for i in features:
    sample = data[i]
    stat, p = shapiro(sample)
    alpha = 0.05
    print('Feature - ',i)
    if p > alpha:
        print("The sample is likely to have been drawn from a normal distribution. (Fail to Reject H0)")
    else:
        print("The sample is not likely to have been drawn from a normal distribution.(reject H0)")
    print('--------------\n')
```

```
Feature -  device_brand
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  os
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  screen_size
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  Includes_4g
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  Includes_5g
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  rear_camera_mp
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  front_camera_mp
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  internal_memory
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  ram
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  battery
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  weight
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  release_year
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  days_used
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  used_price
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------

Feature -  new_price
The sample is not likely to have been drawn from a normal distribution.(reject H0)
--------------
```

**Since, p - value < 0.05 we reject the Null Hypothesis in the Shapiro Wilk Test**

**Multivariate Normality Test**

**H0 - Data is normally distributed**

**H1 - Data is not normally distributed**

In [21]:
```
#Multivariate Normality Test
test_stat, p_value, normal = pg.multivariate_normality(df)

alpha = 0.05
if p_value < alpha:
    print("The dataset is not multivariate normal.")
else:
    print("The dataset is multivariate normal.")
```

```
The dataset is not multivariate normal.
```

**As the p - value < 0.05 we conclude that the data is not normally distributed and reject H0**

In [22]:
```
data = df.copy()
```

In [23]:
```
includes_4g_data = data[data["Includes_4g"] == 1]
excludes_4g_data = data[data["Includes_4g"] == 0]
```

**Z - test**

Since the data doesnot follow a normal distribution, but the sample size is greater than 30 we use Z test.

H0: The population mean of the two samples are equal.
H1: The population mean of the two samples are not equal.

In [24]:
```
_, p_value = ztest(includes_4g_data['used_price'], excludes_4g_data['used_price'])
print('Includes 4g vs Excludes 4g')
alpha = 0.05
if p_value < alpha:
    print('Reject null hypothesis (H0: The population mean of the two samples are equal.): There is significant
else:
    print('Fail to reject null hypothesis (H0: The population mean of the two samples are equal.): There is not
```

```
Includes 4g vs Excludes 4g
Reject null hypothesis (H0: The population mean of the two samples are equal.): There is significant evidence t
o suggest that the population means of the two samples are different.
```

In [25]:
```
includes_5g_data = data[data["Includes_5g"] == 1]
excludes_5g_data = data[data["Includes_5g"] == 0]
```

In [26]:
```
_, p_value = ztest(includes_5g_data['used_price'], excludes_5g_data['used_price'])
print('Includes 5g vs Excludes 5g')
alpha = 0.05
if p_value < alpha:
    print('Reject null hypothesis (H0: The population mean of the two samples are equal.): There is significant
else:
    print('Fail to reject null hypothesis (H0: The population mean of the two samples are equal.): There is not
```

```
Includes 5g vs Excludes 5g
Reject null hypothesis (H0: The population mean of the two samples are equal.): There is significant evidence t
o suggest that the population means of the two samples are different.
```

## 4.2 - Analysis of Variance

One-way ANOVA (F-test)

As we assume the data to be normal since sample size is greater than 30. We use the F - test for analysis of variance.

H0: The population means of all groups are equal.
H1: At least one population mean is different from the others.

In [27]:
```
import statsmodels.api as sm
from statsmodels.formula.api import ols

model = ols("used_price ~ os", data=data).fit()
anova_result = sm.stats.anova_lm(model, typ=1)
print(anova_result)
```

```
                df       sum_sq    mean_sq         F        PR(>F)
os             1.0    17.353165  17.353165  50.75617  1.267204e-12
Residual  3452.0  1180.213702   0.341893       NaN           NaN
```

From the above we can say that p value is less than 0.05 and hence we reject the null hypothesis.

## 4.3 - The Analysis of Categorical Data

The chi-square test of independence is a statistical method used to determine if there is a relationship between two categorical or nominal variables.

H0 - There is no association between the two features
H1 - There is an association between the two features

In [28]:
```
import pandas as pd
from scipy.stats import chi2_contingency

def chi_square_test(df, column1, column2, alpha=0.05):
    contingency_table = pd.crosstab(df[column1], df[column2])
    chi2, p_value, dof, expected = chi2_contingency(contingency_table)
    print(f"Hypothesis statements:")
    print(f"H0: There is no association between {column1} and {column2}.")
    print(f"H1: There is an association between {column1} and {column2}.")
    print(f"Chi-square statistic: {chi2:.2f}")
    print(f"Degrees of freedom: {dof}")
    print(f"P-value: {p_value:.4f}")
    if p_value < alpha:
        print(f"Since p-value ({p_value:.4f}) < alpha ({alpha}), we reject the null hypothesis.")
    else:
        print(f"Since p-value ({p_value:.4f}) >= alpha ({alpha}), we fail to reject the null hypothesis.")
    return chi2, p_value
```

In [29]:
```
chi_square_test(data, 'Includes_4g', 'Includes_5g')
```

```
Hypothesis statements:
H0: There is no association between Includes_4g and Includes_5g.
H1: There is an association between Includes_4g and Includes_5g.
Chi-square statistic: 74.66
Degrees of freedom: 1
P-value: 0.0000
Since p-value (0.0000) < alpha (0.05), we reject the null hypothesis.
```

Out[29]:
```
(74.65651245295918, 5.601654350200409e-18)
```

In [30]:
```
chi_square_test(df, 'Includes_4g', 'os')
```

```
Hypothesis statements:
H0: There is no association between Includes_4g and os.
H1: There is an association between Includes_4g and os.
Chi-square statistic: 170.10
Degrees of freedom: 3
P-value: 0.0000
Since p-value (0.0000) < alpha (0.05), we reject the null hypothesis.
```

Out[30]:
```
(170.0965908090805, 1.212868051325614e-36)
```

In [31]:
```
chi_square_test(df, 'os', 'Includes_5g')
```

```
Hypothesis statements:
H0: There is no association between os and Includes_5g.
H1: There is an association between os and Includes_5g.
Chi-square statistic: 11.87
Degrees of freedom: 3
P-value: 0.0078
Since p-value (0.0078) < alpha (0.05), we reject the null hypothesis.
```

Out[31]:
```
(11.87282923701839, 0.00783173562088424)
```

In [32]:
```
from sklearn.preprocessing import StandardScaler


# Extract the features and target variable
X = df.drop(['used_price'],axis=1)
y = df['used_price']

# Create a scaler object
scaler = StandardScaler()

# Fit and transform the scaler on the features
X_scaled = scaler.fit_transform(X)

# Create a new scaled dataframe
df = pd.DataFrame(X_scaled, columns=X.columns)

# Merge the scaled features and target variable back into a single dataframe
df['used_price'] = y

# Print the first five rows of the scaled dataframe
print(df.head())
```

```
   device_brand        os  screen_size  Includes_4g  Includes_5g  \
0     -0.958061 -0.245963     0.206818     0.692264    -0.214552
1     -0.958061 -0.245963     0.942744     0.692264     4.660867
2     -0.958061 -0.245963     0.782417     0.692264     4.660867
3     -0.958061 -0.245963     3.097957     0.692264     4.660867
4     -0.958061 -0.245963     0.422339     0.692264    -0.214552

   rear_camera_mp  front_camera_mp  internal_memory       ram   battery  \
0        0.769335        -0.222941         0.111324 -0.759524 -0.087163
1        0.769335         1.355830         0.865025  2.905851  0.898683
2        0.769335         0.207633         0.865025  2.905851  0.821663
3        0.769335         0.207633         0.111324  1.439701  3.170750
4        0.769335         0.207633         0.111324 -0.759524  1.437817

     weight  release_year  days_used  new_price  used_price
0 -0.415615      1.755669  -2.204315  -0.757832    4.307572
1  0.343018      1.755669  -1.407676   0.418281    5.162097
2  0.343018      1.755669  -2.063495   0.953164    5.111084
3  3.366229      1.755669  -1.327208   0.582051    5.135387
4  0.025977      1.755669  -1.536426  -0.417343    4.389995
```

## 4.4 Linear Regression

In [33]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd

X = df.drop(['used_price'],axis=1)
y = df['used_price']

x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=100)
```

```
model = LinearRegression().fit(x_train, y_train)
y_pred = model.predict(x_test)
```

In [34]:
```
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae

print("MSE =", mse(y_pred, y_test))
print("MAE =", mae(y_pred, y_test))
```

```
MSE = 0.06094559039832367
MAE = 0.19123345868446312
```

In [35]:
```
score = model.score(x_test, y_test)
print("R^2 score: ", score)
```

```
R^2 score:  0.8317623276310154
```

**After performing the Linear Regression model we can conclude that our model is performing pretty well and has received an accuracy of 83.17%**

## 4.5 - Resampling Methods

**Cross Validation**

In [36]:
```
X = df.drop(['used_price'],axis=1)
y = df['used_price']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)

# Specify the model and perform cross-validation
model = LinearRegression(n_jobs=-1)
cv_results = cross_validate(model, x_train, y_train, cv=5, return_train_score=True)

# Print the mean training and testing scores
print("Mean training score:", cv_results['train_score'].mean())
print("Mean testing score:", cv_results['test_score'].mean())
```

```
Mean training score: 0.844838999041027
Mean testing score: 0.8409337911231255
```

In [37]:
```
# Define the feature matrix and target vector
X = df.drop(columns=['used_price'])
y = df['used_price']

# Split the dataset into training and testing sets
kf = KFold(n_splits=5, shuffle=True, random_state=100)
train_scores, test_scores = [], []
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train a linear regression model
    model = LinearRegression(n_jobs=-1)
    model.fit(X_train, y_train)

    # Compute the training and testing scores
    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)

    # Append the scores to the lists
    train_scores.append(train_score)
    test_scores.append(test_score)

# Print the mean training and testing scores
print("Mean training score:", np.mean(train_scores))
print("Mean testing score:", np.mean(test_scores))
```

```
Mean training score: 0.8422375182221163
Mean testing score: 0.8399908583284109
```

**We can observe that after cross validation we are not able to get any performance improvement.**

**Bootstrapping**

In [38]:
```
# Define the feature matrix and target vector
X = df.drop(columns=['used_price'])
y = df['used_price']

# Split the dataset into training and testing sets
np.random.seed(100)
test_size = 0.2
n_bootstraps = 100
mse_scores = []

for i in range(n_bootstraps):
```

```
    # Create a new training set by sampling with replacement
    X_train_boot, y_train_boot = resample(X, y, random_state=i)

    # Train a linear regression model on the new training set
    model = LinearRegression(n_jobs=-1)
    model.fit(X_train_boot, y_train_boot)

    # Make predictions on the original test set
    y_pred = model.predict(X_test)

    # Calculate the mean squared error
    mse = mean_squared_error(y_test, y_pred)

    # Append the mse to the list of scores
    mse_scores.append(mse)

# Calculate the mean and standard error of the mean squared error
mse_mean = np.mean(mse_scores)
mse_std = np.std(mse_scores) / np.sqrt(n_bootstraps)
print(f"Mean squared error: {mse_mean:.4f} +/- {mse_std:.4f}")
```

```
Mean squared error: 0.0475 +/- 0.0000
```

**From the above we can clearly see a decrease in the Mean squared error as compared to the Linear Regression Model.**

## Linear Model Selection and Regularization

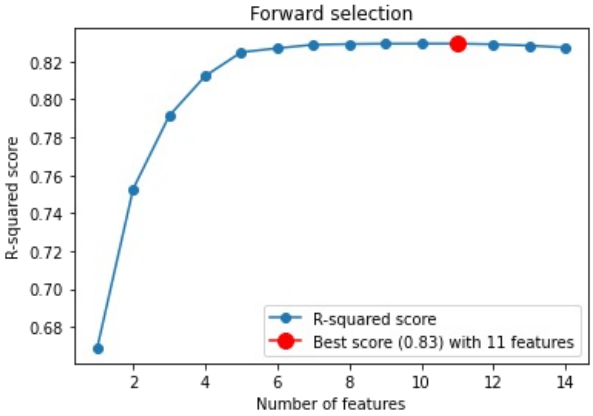
**Forward Selection for Linear Regression**

In [39]:

```
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.model_selection import cross_val_score

X = df.drop(columns=['used_price'])
y = df['used_price']

# Define the estimator
estimator = LinearRegression()

# Initialize an empty list to store the scores
scores = []

# Define the range of number of features to consider
num_features = range(1, len(X.columns) + 1)

# Iterate over the range of number of features
for k in num_features:
    # Define the forward feature selector
    forward_selector = SequentialFeatureSelector(estimator,
                                                 k_features=k,
                                                 forward=True,
                                                 scoring='r2',
                                                 cv=5)
    # Fit the selector to the data
    forward_selector.fit(X, y)

    # Compute the cross-validated score of the selected feature subset
    selected_features = list(X.columns[list(forward_selector.k_feature_idx_)])
    cv_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))

    # Append the score to the list of scores
    scores.append(cv_score)

# Find the best number of features
best_num_features = int(scores.index(max(scores))) + 1

# Train the estimator with the best number of features
forward_selector = SequentialFeatureSelector(estimator,
                                             k_features=best_num_features,
                                             forward=True,
                                             scoring='r2',
                                             cv=5)
forward_selector.fit(X, y)
selected_features = list(X.columns[list(forward_selector.k_feature_idx_)])
selected_features_count = len(selected_features)
selected_features_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))
print("Selected features:", selected_features)
print("Number of selected features:", selected_features_count)
print("R-squared score with selected features:", selected_features_score)
print('-----------------------------------------')

# Plot the scores vs. number of features
fig, ax = plt.subplots()
ax.plot(num_features, scores, marker='o')
ax.set_xlabel('Number of features')
ax.set_ylabel('R-squared score')
ax.set_title('Forward selection')
```

```
# Mark the point with the highest r2 score in red
best_score_index = scores.index(max(scores))
ax.plot(num_features[best_score_index], max(scores), marker='o', markersize=10, color="red")

# Add legend indicating the best r2 score and number of features
ax.legend(["R-squared score", f"Best score ({max(scores):.2f}) with {best_num_features} features"], loc="lower

plt.show()
```

```
Selected features: ['screen_size', 'Includes_4g', 'Includes_5g', 'rear_camera_mp', 'front_camera_mp', 'ram', 'b
attery', 'weight', 'release_year', 'days_used', 'new_price']
Number of selected features: 11
R-squared score with selected features: 0.8293690077427381
-----------------------------------------------
```

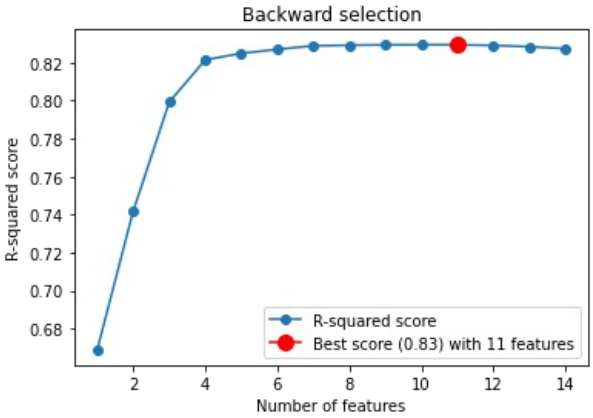



**Backward Selection for Linear Regression**

In [40]:
```
X = df.drop(columns=['used_price'])
y = df['used_price']

# Define the estimator
estimator = LinearRegression()

# Initialize an empty list to store the scores
scores = []

# Define the range of number of features to consider
num_features = range(1, len(X.columns) + 1)

# Iterate over the range of number of features
for k in num_features:
    # Define the backward feature selector
    backward_selector = SequentialFeatureSelector(estimator,
                                                  k_features=k,
                                                  forward=False,
                                                  scoring='r2',
                                                  cv=5)
    # Fit the selector to the data
    backward_selector.fit(X, y)

    # Compute the cross-validated score of the selected feature subset
    selected_features = list(X.columns[list(backward_selector.k_feature_idx_)])
    cv_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))

    # Append the score to the list of scores
    scores.append(cv_score)

# Find the best number of features
best_num_features = int(scores.index(max(scores))) + 1

# Train the estimator with the best number of features
backward_selector = SequentialFeatureSelector(estimator,
                                              k_features=best_num_features,
                                              forward=False,
                                              scoring='r2',
                                              cv=5)
backward_selector.fit(X, y)
selected_features = list(X.columns[list(backward_selector.k_feature_idx_)])
selected_features_count = len(selected_features)
selected_features_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))
print("Selected features:", selected_features)
print("Number of selected features:", selected_features_count)
print("R-squared score with selected features:", selected_features_score)
print('-----------------------------------------------')

# Plot the scores vs. number of features
fig, ax = plt.subplots()
ax.plot(num_features, scores, marker='o')
ax.set_xlabel('Number of features')
ax.set_ylabel('R-squared score')
```

```
ax.set_title('Backward selection')
# Mark the point with the highest r2 score in red
best_score_index = scores.index(max(scores))
ax.plot(num_features[best_score_index], max(scores), marker='o', markersize=10, color="red")

# Add legend indicating the best r2 score and number of features
ax.legend(["R-squared score", f"Best score ({max(scores):.2f}) with {best_num_features} features"], loc="lower

plt.show()
```

```
Selected features: ['screen_size', 'Includes_4g', 'Includes_5g', 'rear_camera_mp', 'front_camera_mp', 'ram', 'b
attery', 'weight', 'release_year', 'days_used', 'new_price']
Number of selected features: 11
R-squared score with selected features: 0.8293690077427381
--------------------------------------------
```




In [41]:
```
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae

# Load the data with the selected features from forward and backward
X = df[['screen_size', 'Includes_4g', 'Includes_5g', 'rear_camera_mp', 'front_camera_mp',
        'ram', 'battery', 'weight', 'release_year', 'days_used', 'new_price']]
y = df['used_price']

x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=100)

model = LinearRegression().fit(x_train, y_train)
y_pred = model.predict(x_test)

print("MSE =", mse(y_pred, y_test))
print("MAE =", mae(y_pred, y_test))

score = model.score(x_test, y_test)
print("R^2 score: ", score)
```

```
MSE = 0.060775979734404295
MAE = 0.19095985937745216
R^2 score:  0.8322305305497214
```

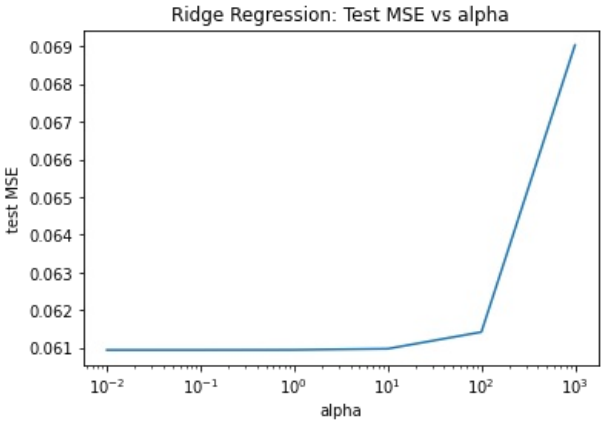
**After selection of the subset of features, we apply it to the Linear Model again. And we see a slight increase in the performance of the model as compared to the previous Linear Model.**

**Ridge Regression**

In [42]:
```
from sklearn.linear_model import Ridge

# Load the data with the selected features
X = df.drop(columns=['used_price'])
y = df['used_price']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=100)

# Ridge Regression
alphas = [0.01, 0.1, 1, 10, 100, 1000]  # Range of regularization strengths
ridge_mse = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
    ridge_mse.append(mean_squared_error(y_test, y_pred))

# Find the best alpha that gives the lowest MSE
best_alpha_ridge = alphas[ridge_mse.index(min(ridge_mse))]

# Fit the Ridge model with the best alpha and calculate the test MSE
ridge = Ridge(alpha=best_alpha_ridge)
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
ridge_test_mse = mean_squared_error(y_test, y_pred)
```

```
print("Ridge regression test MSE:", ridge_test_mse)
print('-------------------------------------------')
print("Best alpha for Ridge regression:", best_alpha_ridge)
print('-------------------------------------------')

# Plot the test MSE vs alpha for Ridge regression
plt.plot(alphas, ridge_mse)
plt.xscale('log')
plt.xlabel('alpha')
plt.ylabel('test MSE')
plt.title('Ridge Regression: Test MSE vs alpha')
plt.show()
```

```
Ridge regression test MSE: 0.06094562086624721
-------------------------------------------
Best alpha for Ridge regression: 0.01
-------------------------------------------
```




**We dont see any noticeable difference in the results.**

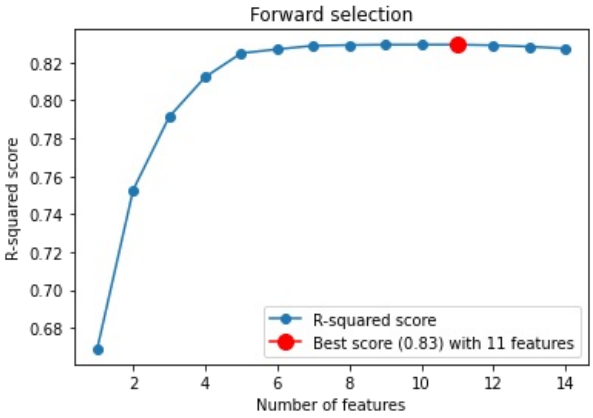
**Forward Selection with Ridge Regression**

In [43]:

```
from mlxtend.feature_selection import SequentialFeatureSelector

X = df.drop(columns=['used_price'])
y = df['used_price']

# Define the estimator
estimator = Ridge(alpha=0.01)

# Initialize an empty list to store the scores
scores = []

# Define the range of number of features to consider
num_features = range(1, len(X.columns) + 1)

# Iterate over the range of number of features
for k in num_features:
    # Define the forward feature selector
    forward_selector = SequentialFeatureSelector(estimator,
                                                 k_features=k,
                                                 forward=True,
                                                 scoring='r2',
                                                 cv=5)
    # Fit the selector to the data
    forward_selector.fit(X, y)

    # Compute the cross-validated score of the selected feature subset
    selected_features = list(X.columns[list(forward_selector.k_feature_idx_)])
    cv_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))

    # Append the score to the list of scores
    scores.append(cv_score)

# Find the best number of features
best_num_features = int(scores.index(max(scores))) + 1

# Train the estimator with the best number of features
forward_selector = SequentialFeatureSelector(estimator,
                                             k_features=best_num_features,
                                             forward=True,
                                             scoring='r2',
                                             cv=5)
forward_selector.fit(X, y)
selected_features = list(X.columns[list(forward_selector.k_feature_idx_)])
selected_features_count = len(selected_features)
selected_features_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))
print("Selected features:", selected_features)
print("Number of selected features:", selected_features_count)
```

```
print("R-squared score with selected features:", selected_features_score)
print('-------------------------------------------')

# Plot the scores vs. number of features
fig, ax = plt.subplots()
ax.plot(num_features, scores, marker='o')
ax.set_xlabel('Number of features')
ax.set_ylabel('R-squared score')
ax.set_title('Forward selection')
# Mark the point with the highest r2 score in red
best_score_index = scores.index(max(scores))
ax.plot(num_features[best_score_index], max(scores), marker='o', markersize=10, color="red")

# Add legend indicating the best r2 score and number of features
ax.legend(["R-squared score", f"Best score ({max(scores):.2f}) with {best_num_features} features"], loc="lower

plt.show()
```

```
Selected features: ['screen_size', 'Includes_4g', 'Includes_5g', 'rear_camera_mp', 'front_camera_mp', 'ram', 'b
attery', 'weight', 'release_year', 'days_used', 'new_price']
Number of selected features: 11
R-squared score with selected features: 0.8293690594532876
-------------------------------------------
```



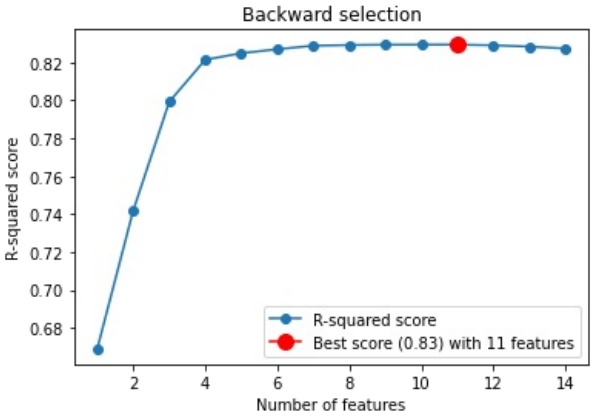

**Backward Selection with Ridge Regression**

In [44]:

```
X = df.drop(columns=['used_price'])
y = df['used_price']

# Define the estimator
estimator = Ridge(alpha=0.01)

# Initialize an empty list to store the scores
scores = []

# Define the range of number of features to consider
num_features = range(1, len(X.columns) + 1)

# Iterate over the range of number of features
for k in num_features:
    # Define the backward feature selector
    backward_selector = SequentialFeatureSelector(estimator,
                                                  k_features=k,
                                                  forward=False,
                                                  scoring='r2',
                                                  cv=5)
    # Fit the selector to the data
    backward_selector.fit(X, y)

    # Compute the cross-validated score of the selected feature subset
    selected_features = list(X.columns[list(backward_selector.k_feature_idx_)])
    cv_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))

    # Append the score to the list of scores
    scores.append(cv_score)

# Find the best number of features
best_num_features = int(scores.index(max(scores))) + 1

# Train the estimator with the best number of features
backward_selector = SequentialFeatureSelector(estimator,
                                              k_features=best_num_features,
                                              forward=False,
                                              scoring='r2',
                                              cv=5)
backward_selector.fit(X, y)
selected_features = list(X.columns[list(backward_selector.k_feature_idx_)])
selected_features_count = len(selected_features)
selected_features_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))
print("Selected features:", selected_features)
```

```
print("Number of selected features:", selected_features_count)
print("R-squared score with selected features:", selected_features_score)
print('-----------------------------------------')

# Plot the scores vs. number of features
fig, ax = plt.subplots()
ax.plot(num_features, scores, marker='o')
ax.set_xlabel('Number of features')
ax.set_ylabel('R-squared score')
ax.set_title('Backward selection')
# Mark the point with the highest r2 score in red
best_score_index = scores.index(max(scores))
ax.plot(num_features[best_score_index], max(scores), marker='o', markersize=10, color="red")

# Add legend indicating the best r2 score and number of features
ax.legend(["R-squared score", f"Best score ({max(scores):.2f}) with {best_num_features} features"], loc="lower

plt.show()
```

```
Selected features: ['screen_size', 'Includes_4g', 'Includes_5g', 'rear_camera_mp', 'front_camera_mp', 'ram', 'b
attery', 'weight', 'release_year', 'days_used', 'new_price']
Number of selected features: 11
R-squared score with selected features: 0.8293690594532876
-----------------------------------------
```




**Ridge Regression with selected features**

In [45]:
```
# Load the data with the selected features
X = df[['screen_size', 'Includes_4g', 'Includes_5g', 'rear_camera_mp', 'front_camera_mp',
        'ram', 'battery', 'weight', 'release_year', 'days_used', 'new_price']]
y = df['used_price']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=100)

# Ridge Regression
alpha = 0.01
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
ridge_test_mse = mean_squared_error(y_test, y_pred)

print("Ridge regression test MSE:", ridge_test_mse)
```

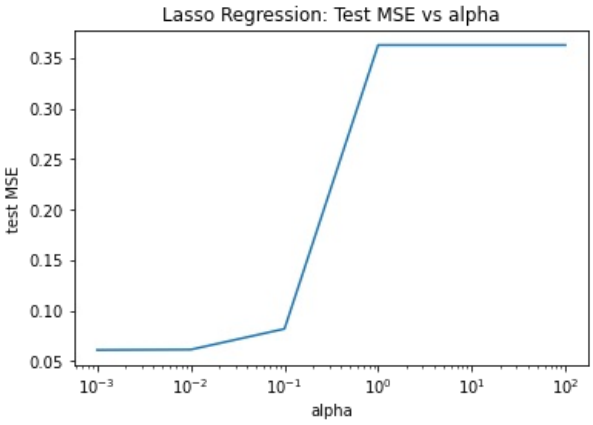
```
Ridge regression test MSE: 0.06077600457970349
```

**Lasso Regression**

In [46]:
```
from sklearn.linear_model import Lasso

# Load the data with the selected features
X = df.drop(columns=['used_price'])
y = df['used_price']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=100)

# Lasso Regression
alphas = [0.001, 0.01, 0.1, 1, 10, 100]  # Range of regularization strengths
lasso_mse = []
for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    y_pred = lasso.predict(X_test)
    lasso_mse.append(mean_squared_error(y_test, y_pred))

# Find the best alpha that gives the lowest MSE
best_alpha_lasso = alphas[lasso_mse.index(min(lasso_mse))]

# Fit the Lasso model with the best alpha and calculate the test MSE
```

```
lasso = Lasso(alpha=best_alpha_lasso)
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
lasso_test_mse = mean_squared_error(y_test, y_pred)

print("Lasso regression test MSE:", lasso_test_mse)
print('-----------------------------------------')
print("Best alpha for Lasso regression:", best_alpha_lasso)
print('-----------------------------------------')

# Plot the test MSE vs alpha for Lasso regression
plt.plot(alphas, lasso_mse)
plt.xscale('log')
plt.xlabel('alpha')
plt.ylabel('test MSE')
plt.title('Lasso Regression: Test MSE vs alpha')
plt.show()
```

```
Lasso regression test MSE: 0.06095411430037157
-----------------------------------------
Best alpha for Lasso regression: 0.001
-----------------------------------------
```




**We dont see any noticeable difference in the results.**

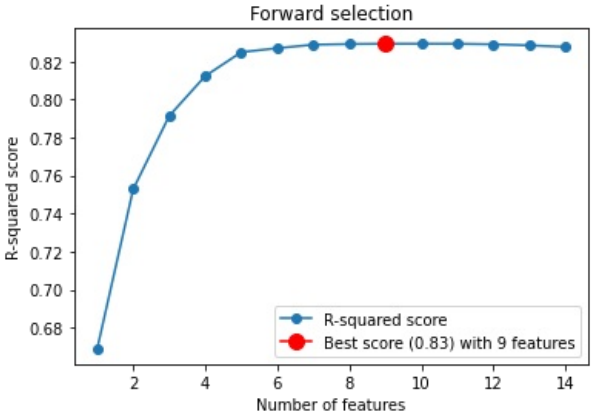
**Forward Selection with Lasso Regression**

In [47]:

```
from mlxtend.feature_selection import SequentialFeatureSelector

X = df.drop(columns=['used_price'])
y = df['used_price']

# Define the estimator
estimator = Lasso(alpha=0.001)

# Initialize an empty list to store the scores
scores = []

# Define the range of number of features to consider
num_features = range(1, len(X.columns) + 1)

# Iterate over the range of number of features
for k in num_features:
    # Define the forward feature selector
    forward_selector = SequentialFeatureSelector(estimator,
                                                 k_features=k,
                                                 forward=True,
                                                 scoring='r2',
                                                 cv=5)
    # Fit the selector to the data
    forward_selector.fit(X, y)

    # Compute the cross-validated score of the selected feature subset
    selected_features = list(X.columns[list(forward_selector.k_feature_idx_)])
    cv_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))

    # Append the score to the list of scores
    scores.append(cv_score)

# Find the best number of features
best_num_features = int(scores.index(max(scores))) + 1

# Train the estimator with the best number of features
forward_selector = SequentialFeatureSelector(estimator,
                                             k_features=best_num_features,
                                             forward=True,
                                             scoring='r2',
                                             cv=5)
forward_selector.fit(X, y)
selected_features = list(X.columns[list(forward_selector.k_feature_idx_)])
```

```
selected_features_count = len(selected_features)
selected_features_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))
print("Selected features:", selected_features)
print("Number of selected features:", selected_features_count)
print("R-squared score with selected features:", selected_features_score)
print('-----------------------------------------')

# Plot the scores vs. number of features
fig, ax = plt.subplots()
ax.plot(num_features, scores, marker='o')
ax.set_xlabel('Number of features')
ax.set_ylabel('R-squared score')
ax.set_title('Forward selection')
# Mark the point with the highest r2 score in red
best_score_index = scores.index(max(scores))
ax.plot(num_features[best_score_index], max(scores), marker='o', markersize=10, color="red")

# Add legend indicating the best r2 score and number of features
ax.legend(["R-squared score", f"Best score ({max(scores):.2f}) with {best_num_features} features"], loc="lower

plt.show()
```

```
Selected features: ['screen_size', 'Includes_4g', 'rear_camera_mp', 'front_camera_mp', 'ram', 'weight', 'releas
e_year', 'days_used', 'new_price']
Number of selected features: 9
R-squared score with selected features: 0.8292834831098105
-----------------------------------------
```



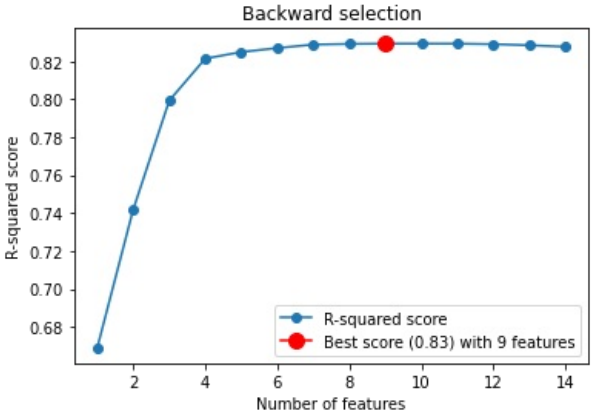

**Backward Selection with Lasso Regression**

In [48]:

```
X = df.drop(columns=['used_price'])
y = df['used_price']

# Define the estimator
estimator = Lasso(alpha=0.001)

# Initialize an empty list to store the scores
scores = []

# Define the range of number of features to consider
num_features = range(1, len(X.columns) + 1)

# Iterate over the range of number of features
for k in num_features:
    # Define the backward feature selector
    backward_selector = SequentialFeatureSelector(estimator,
                                                  k_features=k,
                                                  forward=False,
                                                  scoring='r2',
                                                  cv=5)
    # Fit the selector to the data
    backward_selector.fit(X, y)

    # Compute the cross-validated score of the selected feature subset
    selected_features = list(X.columns[list(backward_selector.k_feature_idx_)])
    cv_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))

    # Append the score to the list of scores
    scores.append(cv_score)

# Find the best number of features
best_num_features = int(scores.index(max(scores))) + 1

# Train the estimator with the best number of features
backward_selector = SequentialFeatureSelector(estimator,
                                              k_features=best_num_features,
                                              forward=False,
                                              scoring='r2',
                                              cv=5)
backward_selector.fit(X, y)
```

```
selected_features = list(X.columns[list(backward_selector.k_feature_idx_)])
selected_features_count = len(selected_features)
selected_features_score = np.mean(cross_val_score(estimator, X[selected_features], y, cv=5, scoring='r2'))
print("Selected features:", selected_features)
print("Number of selected features:", selected_features_count)
print("R-squared score with selected features:", selected_features_score)
print('-----------------------------------------')

# Plot the scores vs. number of features
fig, ax = plt.subplots()
ax.plot(num_features, scores, marker='o')
ax.set_xlabel('Number of features')
ax.set_ylabel('R-squared score')
ax.set_title('Backward selection')
# Mark the point with the highest r2 score in red
best_score_index = scores.index(max(scores))
ax.plot(num_features[best_score_index], max(scores), marker='o', markersize=10, color="red")

# Add legend indicating the best r2 score and number of features
ax.legend(["R-squared score", f"Best score ({max(scores):.2f}) with {best_num_features} features"], loc="lower

plt.show()
```

```
Selected features: ['screen_size', 'Includes_4g', 'rear_camera_mp', 'front_camera_mp', 'ram', 'weight', 'releas
e_year', 'days_used', 'new_price']
Number of selected features: 9
R-squared score with selected features: 0.8292834831098105
-----------------------------------------
```




**Lasso Regression with selected features**

In [49]:
```
# Load the data with the selected features
X = df[['screen_size', 'Includes_4g', 'rear_camera_mp', 'front_camera_mp', 'ram',
        'weight', 'release_year', 'days_used', 'new_price']]
y = df['used_price']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=100)

# Ridge Regression
alpha = 0.001
lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
lasso_test_mse = mean_squared_error(y_test, y_pred)

print("Lasso regression test MSE:", lasso_test_mse)
```

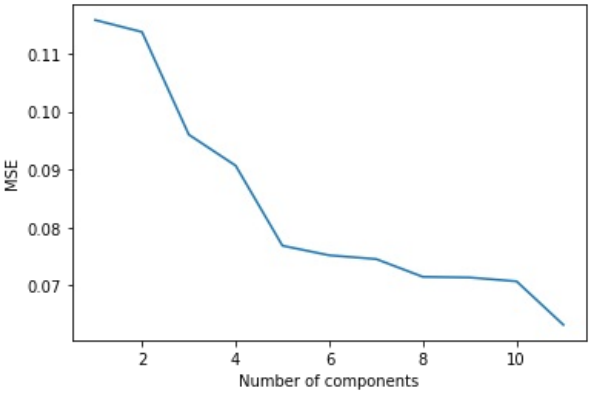
```
Lasso regression test MSE: 0.060872792211158946
```

**PCA**

In [50]:
```
from sklearn.model_selection import RepeatedKFold
from sklearn.decomposition import PCA
from sklearn import model_selection
from sklearn.metrics import mean_squared_error as sklearn_mse
from sklearn.metrics import mean_absolute_error as mae
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#define predictor and response variables
X = df.drop(columns=['used_price'])
y = df['used_price']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Perform PCA and determine the optimal number of components
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
explained_variances = pca.explained_variance_ratio_
cumulative_variances = np.cumsum(explained_variances)
n_components = np.argmax(cumulative_variances >= 0.95) + 1
print("Optimal number of components:", n_components)

# Retain the optimal number of components
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)

# Loop through a range of values for the number of components
mse_values = []
n_components_range = range(1, X_pca.shape[1]+1)
for n_components in n_components_range:
    # Split the data into training and testing sets
    x_train, x_test, y_train, y_test = train_test_split(X_pca[:,:n_components], y, test_size=0.2, random_state=

    # Fit the linear regression model and make predictions
    model = LinearRegression().fit(x_train, y_train)
    y_pred = model.predict(x_test)

    # Calculate the MSE and add it to the list of MSE values
    mse_values.append(sklearn_mse(y_pred, y_test))

# Plot the number of components vs. MSE
plt.plot(n_components_range, mse_values)
plt.xlabel('Number of components')
plt.ylabel('MSE')
plt.show()

# Fit the linear regression model with the optimal number of components
x_train, x_test, y_train, y_test = train_test_split(X_pca[:,:n_components], y, test_size=0.2, random_state=100)
model = LinearRegression().fit(x_train, y_train)
y_pred = model.predict(x_test)

# Calculate the evaluation metrics
print("MSE =", mse(y_pred, y_test))
print("MAE =", mae(y_pred, y_test))
```

```
Optimal number of components: 11
```




```
MSE = 0.0632018100363985
MAE = 0.19369845540513309
```

## 4.7 - Moving beyond linearity

**Polynomial Regression**

**Polynomial regression is a type of regression analysis where the relationship between the independent variable X and the dependent variable Y is modeled as an nth degree polynomial. In other words, instead of fitting a straight line to the data points, polynomial regression can fit a curve that passes through the data points.**

**This method is useful when the relationship between X and Y is not linear and cannot be captured by a simple straight line. By adding polynomial terms to the regression equation, the model can better fit the curvature of the data and provide more accurate predictions. However, adding too many polynomial terms can result in overfitting the model to the training data and poor performance on new, unseen data. Therefore, the degree of the polynomial should be chosen carefully based on the complexity of the relationship between X and Y and the available amount of data.**

In [51]:
```
# Import necessary libraries
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

```
X = df.drop(columns=['used_price'])
y = df['used_price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define a list of degrees to test
degrees = [1, 2, 3, 4, 5]

# Define empty lists to store the evaluation metrics for each degree
r2_scores = []
mses = []

# Loop over each degree and fit a polynomial regression model
for degree in degrees:
    # Transform X into a polynomial feature matrix
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    # Train the polynomial regression model
    poly_reg = LinearRegression()
    cv_results = cross_validate(poly_reg, X_train_poly, y_train, cv=5, scoring=('r2', 'neg_mean_squared_error')
    r2_scores.append(np.mean(cv_results['test_r2']))
    mses.append(-np.mean(cv_results['test_neg_mean_squared_error']))

# Identify the best degree that gives us the highest R-squared and lowest MSE
best_degree_r2 = degrees[np.argmax(r2_scores)]
best_degree_mse = degrees[np.argmin(mses)]

# Print the best degree and its corresponding evaluation metrics
print("Best degree for R-squared score:", best_degree_r2)
print("Best degree for Mean Squared Error:", best_degree_mse)
print("R-squared score:", r2_scores)
print("Mean Squared Error:", mses)
```

```
Best degree for R-squared score: 2
Best degree for Mean Squared Error: 2
R-squared score: [0.8391635026743991, 0.8428871868495316, -169762.33553605538, -4.19445128540403e+17, -1463224.
695079087]
Mean Squared Error: [0.0557202684571661, 0.05448701857842603, 61237.88122870125, 1.503279039590585e+17, 527044.
740399072]
```

In [52]:

```
# Import necessary libraries
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

X = df.drop(columns=['used_price'])
y = df['used_price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Transform X into a polynomial feature matrix
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Train the polynomial regression model
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)

# Predict the used prices for the test set
y_pred = poly_reg.predict(X_test_poly)

# Evaluate the model's performance using R-squared score, MSE, and MAE
r2_score = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

# Print the evaluation metrics
print("R-squared score:", r2_score)
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
```

```
R-squared score: 0.853913611971753
Mean Squared Error: 0.049747932466010156
Mean Absolute Error: 0.17847115890397292
```

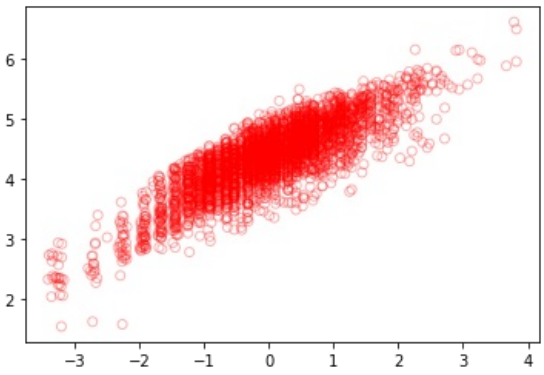
**We see an increase in accuracy from 83.17% to 85.39% after using polynomial regression.**

## Regression with Splines

**From correlation matrix in our EDA part we can see that new_price is having a good correlation with our target variable. Hence**

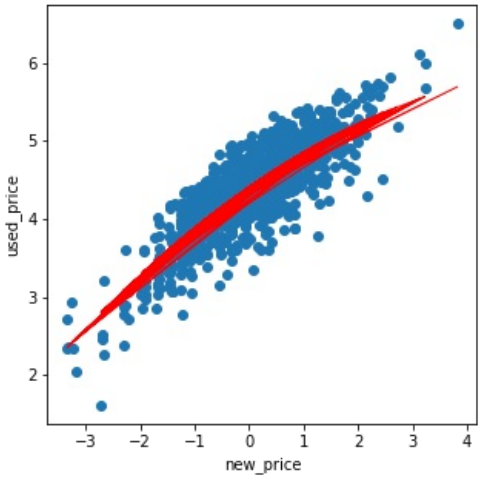
lets perform regression spline with it.

In [53]:
```
x = df['new_price'].values.reshape(-1,1)
y = data['used_price'].values.reshape(-1,1)
plt.scatter(x, y, facecolor='None', edgecolor='r', alpha=0.3)
plt.show()
```



Let us check for degree 2 first

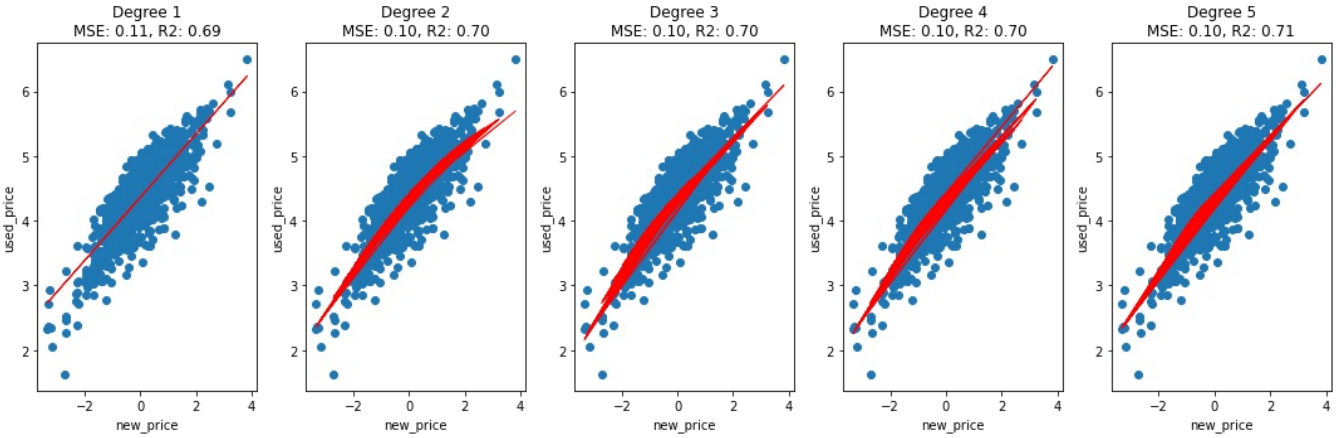
In [60]:
```
X = df['new_price']
y = df['used_price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

X_train_reshaped = X_train.values.reshape(-1, 1)
X_test_reshaped = X_test.values.reshape(-1, 1)

# Transform X into a polynomial feature matrix
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train_reshaped)
X_test_poly = poly.transform(X_test_reshaped)

# Train the polynomial regression model
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)

# Predict the used prices for the test set
y_pred = poly_reg.predict(X_test_poly)

plt.figure(figsize=(5,5))
plt.scatter(X_test, y_test)
plt.plot(X_test, y_pred, color='red', linewidth=1)
plt.xlabel('new_price')
plt.ylabel('used_price')
plt.show()
```




Now let us check for degrees 1 to 5.

In [63]:
```
from sklearn.metrics import mean_squared_error, r2_score
# Extract the relevant columns
X = df['new_price']
y = df['used_price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Set the range of degrees to test
degrees = [1, 2, 3, 4, 5]
```

```
# Create a figure to hold the subplots
fig, axes = plt.subplots(nrows=1, ncols=len(degrees), figsize=(15, 5))

# Iterate over the degrees and create a subplot for each
for i, degree in enumerate(degrees):

    # Transform X into a polynomial feature matrix
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train.values.reshape(-1, 1))
    X_test_poly = poly.transform(X_test.values.reshape(-1, 1))

    # Train the polynomial regression model
    poly_reg = LinearRegression()
    poly_reg.fit(X_train_poly, y_train)

    # Predict the used prices for the test set
    y_pred = poly_reg.predict(X_test_poly)

    # Compute the MSE and R2 scores
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Plot the results on the subplot
    ax = axes[i]
    ax.scatter(X_test, y_test)
    ax.plot(X_test, y_pred, color='red', linewidth=1)
    ax.set_xlabel('new_price')
    ax.set_ylabel('used_price')
    ax.set_title(f'Degree {degree}\nMSE: {mse:.2f}, R2: {r2:.2f}')

plt.tight_layout()
plt.show()
```




**From the above we can see and conclude that degree = 5 had the highest r2 score.**

## Conclusion

**We have successfully used statistical methods for the analysis of our data. We used Shapiro Wilk Test and Multivariate Normality to test the Normality of the data. There after, we used Z - test for comparing two samples and used the F-test in the Analysis of Variance. From the results of the Z - test we were able to conclude that the population mean of the two samples are not equal. From the F-test we were able to conlude that the population mean of all the samples are not equal. To analyse the categorical data in a better way we performed the chi-square test which told us that was an association between the categorical features.**

**The linear regression model was able to achieve an accuracy of 83.17%, but cross-validation did not result in any performance improvement. Bootstrapping was able to reduce the mean squared error compared to the linear regression model. Additionally, polynomial regression achieved a higher R-squared score of 0.8539. We also performed ridge regression and lasso regression which gave us similar accuracy when compared to that of Linear regression.**

**In summary, the dataset has significant differences and associations between some of the features, and various models were used to predict the target variable. Polynomial regression was found to be the best model for this dataset based on the higher R-squared score and lower mean squared error. However, further investigation and experimentation could be done to improve the model's performance.**

## References

**Dataset - https://www.kaggle.com/datasets/ahsan81/used-handheld-device-data**

In [ ]: