Predicting Mobile Phone Prices, using Machine Learning Algorithms

# **Introduction**

In an era where mobile phones have become an integral part of our daily lives, understanding and predicting the factors that influence their prices is of paramount importance. The mobile phone market is characterized by rapid technological advancements, an array of brands and models, and varying consumer preferences. Consequently, consumers, manufacturers, retailers, and investors are keenly interested in predicting mobile phone prices accurately.

This project, "Predicting Mobile Phone Prices with Machine Learning," aims to leverage the power of artificial intelligence and machine learning to develop robust models that can forecast the prices of mobile phones based on a myriad of relevant features. By harnessing the capabilities of cutting-edge machine learning algorithms, data analysis, and predictive modeling, we intend to provide valuable insights into the intricate dynamics of mobile phone pricing.

There are very many opportunities that will surface if AI can easily predict the prices of phone considering very many factors that are represented as several variables in data sets.

Machine learning models can indeed be highly effective in predicting mobile phone prices when carefully developed and trained using relevant data. While they may not achieve perfect prediction, they can provide accurate and valuable insights into the factors influencing mobile phone prices. This dataset contains variables that has several specifications about a mobile phone of different ranges. Some of the variables are Brand, Model, Name, Camera, Resolution, etc. The data in question was generated un processed. Several models that can easily and carefully predict mobile phone prices will be used e.g. linear regression, decision tree, random forest regressor etc. One of the most significant and widely used technologies in the world is the mobile phone. They are utilized for everything, including productivity, pleasure, and communication. As a result, knowing the price of a mobile phone before making a purchase could be crucial. The brand, model, features, and specs are just a few things that might have

an impact on a phone's pricing. The cost of the parts that go into making the phone, however, is the most crucial element.

**1.2 Research Aims and Objectives**

**Aims**

To assess the effectiveness and applicability of predictive models for Price prediction using data that contains different variables about specifications of a mobile phone.

**Objectives**

The objectives of this study are outlined below.

1. Develop predictive models for price prediction using the dataset.
2. Compare the performance of models derived from the dataset.
3. Identify strengths and weaknesses of the models.
4. Assess the generalization capabilities of these models.
5. Gain insights into potential practical applications of these price prediction models in various contexts.
6. Contribute to the understanding of predictive models' effectiveness in real-world stress prediction scenarios.

# **Literature Review**

Predicting mobile phone prices using machine learning techniques has garnered significant attention in recent years due to its practical applications in the mobile phone industry, consumer decision-making, and investment strategies. Researchers have emphasized the importance of selecting and engineering relevant features for mobile phone price prediction. Technical specifications like processor speed, camera quality, and screen size are commonly used. Additionally, contextual features such as brand reputation, market trends, and user reviews have been incorporated to enhance prediction accuracy (Khan et al., 2020). Various machine learning algorithms have been employed for mobile phone price prediction. Linear regression models are frequently used for their simplicity and interpretability. More complex models like decision trees, random forests, and gradient boosting techniques have demonstrated improved predictive performance (Karim et al., 2018). Researchers have highlighted the need for data cleaning, handling missing values, and outlier detection to ensure reliable predictions (Yuan et al., 2021).

**2.1 Methodology:**

The goal of this project is to predict mobile phone prices using four different machine learning regression models: Linear Regression, Random Forest Regressor, Decision Tree Regressor, and Support Vector Machines (SVM). The following methodology outlines the step-by-step process to achieve accurate price predictions:

1. Data Collection:

Gather a comprehensive dataset containing mobile phone specifications (features) and their corresponding prices.

Ensure the dataset includes a diverse set of mobile phone models, brands, and specifications to capture a broad range of factors influencing prices.

2. Data Preprocessing:

Handle missing data by imputing values or removing instances as appropriate.

Address outliers and noisy data points, considering their impact on model performance.

Encode categorical variables (e.g., brand names, operating systems) into numerical representations (one-hot encoding or label encoding).

Normalize or scale numerical features to ensure all features have the same influence on the models.

3. Feature Selection and Engineering:

Explore the dataset through exploratory data analysis (EDA) to understand feature distributions and relationships.

Select relevant features based on domain knowledge and statistical analysis.

Optionally, engineer new features that might capture additional information (e.g., feature interactions or ratios).

4. Splitting the Data:

Divide the dataset into two subsets: a training set and a testing set. A common split is 80% for training and 30% for testing.

5. Model Building:

Implement four regression models: Linear Regression, Random Forest Regressor, Decision Tree Regressor, and Support Vector Machines (SVM).

Train each model on the training dataset using the selected features.

Tune hyperparameters, such as the regularization term in Linear Regression, max depth in Decision Trees, or kernel functions in SVM, through cross-validation or grid search.

6. Model Evaluation:

Evaluate the performance of each model on the testing dataset using appropriate regression evaluation metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared (R2).

Compare the models' performance metrics to identify the best-performing model.

7. Model Interpretability (Optional):

For Linear Regression, analyze variance of residuals and check the error term.

For tree-based models (Random Forest and Decision Trees), classification reports and confusion matrix.

8. Final Model Selection:

Based on the evaluation results, select the best-performing model for mobile phone price prediction.

**2.2 Models used and their methodology**

In this section, the focus will be on examining the technological usage of decision tree model to predict price of mobile phones.

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. It is a simple yet powerful predictive modeling technique that creates a tree-like structure to make decisions based on input features. Decision trees are widely used because of their interpretability, ease of use, and ability to handle both categorical and numerical data.

**Basic Concepts**:

Tree Structure: A decision tree is a hierarchical structure composed of nodes. It starts with a root node and branches out into internal nodes and leaf nodes (also known as terminal nodes). Internal nodes represent decisions based on input features, while leaf nodes represent the output or class label.

Splitting: At each internal node, a decision is made based on one of the input features. This splitting process continues recursively until a stopping criterion is met (e.g., a maximum depth is reached, or the data is pure in the leaf nodes for classification task.

For instance, a study by Zhang et al., (2022) utilized deep learning techniques to analyze physiological data collected from wearable sensors and predict stress levels with high accuracy. The authors demonstrated that their model could effectively identify stress patterns, offering insights into personalized stress management strategies. Moreover, advancements in AI and wearable technology have enabled the development of real-time stress prediction systems. These systems can continuously monitor physiological signals, process the data on the device or through cloud computing, and provide timely feedback to users when stress levels exceed certain thresholds. Such applications hold great potential in promoting stress-awareness and proactive stress management.

Create Child Nodes: Create child nodes corresponding to each outcome of the chosen feature and condition. These child nodes represent subsequent decisions.

Recursion: Recursively repeat the process for each child node, further partitioning the data until a stopping criterion is met. Common stopping criteria include a maximum depth for the tree, a minimum number of samples in leaf nodes, or a minimum improvement in the splitting criteria.

Assign Predictions: Assign a prediction value or class label to each leaf node. For classification tasks, the prediction is often the majority class in the leaf. For regression tasks, it's typically the mean or median of the target variable in the leaf

**Formula**: The formula used to calculate the splitting criterion varies depending on the type of task (classification or regression) and the chosen criterion. Here are two common formulas:

Gini Impurity (Classification): The Gini impurity measures the probability of misclassifying a randomly chosen element if it were classified according to the class distribution in the node.

Gini Impurity (I\_Gini) = 1 - Σ (p\_i) ^2 Where: I\_Gini: Gini impurity for the node, p\_i: Proportion of instances of class i in the node

Mean Squared Error (Regression): Mean squared error is used in regression decision trees to measure the variance of the target variable within a node. Mean Squared Error (MSE) = Σ (y\_i - ŷ) ^2 / n Where: MSE: Mean squared error for the node y\_i: Actual target value of the i-th instance, ŷ: Predicted target value for the node, n: Number of instances in the node

Linear regression is a fundamental statistical and machine learning technique used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. It is widely employed for both predictive and explanatory purposes, making it a versatile tool in various fields, such as economics, finance, social sciences, and machine learning.

**Components of Linear Regression:**

1. **Dependent Variable (Y):** This is the variable we want to predict or explain. It is often referred to as the target variable or the response variable.
2. **Independent Variables (X):** These are the variables that are used to predict or explain the dependent variable. In simple linear regression, there is one independent variable (X), while in multiple linear regression, there are multiple independent variables (X₁, X₂, X₃, ...).
3. **Coefficients (β₀, β₁, β₂, ...):** These are the parameters of the linear regression model. β₀ is the intercept, representing the predicted value of Y when all independent variables are zero. β₁, β₂, etc., are the coefficients associated with each independent variable, indicating the change in Y for a one-unit change in the corresponding X.
4. **Residuals (ε):** These are the errors or discrepancies between the predicted values (Ŷ) and the actual values (Y) in the dataset. The goal is to minimize the sum of squared residuals.

**The Linear Regression Equation:**

The simple linear regression equation for a single independent variable is:

Y = β₀ + β₁X + ε

In this equation:

* Y is the dependent variable.
* X is the independent variable.
* β₀ is the intercept (the value of Y when X is zero).
* β₁ is the slope coefficient (the change in Y for a one-unit change in X).
* ε represents the error term (residuals).

In multiple linear regression with multiple independent variables (X₁, X₂, X₃, ...), the equation becomes:

Y = β₀ + β₁X₁ + β₂X₂ + β₃X₃ + ... + ε

**Model Estimation:**

The goal in linear regression is to estimate the coefficients (β₀, β₁, β₂, ...) that best fit the data. This is typically done using the method of least squares, which minimizes the sum of squared residuals:

Sum of Squared Residuals (SSE) = Σ(Ŷᵢ - Yᵢ)²

Where:

* Ŷᵢ is the predicted value for the i-th observation.
* Yᵢ is the actual value for the i-th observation.

The coefficients are estimated as follows:

* β₁ (slope) = (Σ(Xᵢ - X̄)(Yᵢ - Ȳ)) / Σ(Xᵢ - X̄)²
* β₀ (intercept) = Ȳ - β₁X̄

Where:

* X̄ is the mean of the independent variable X.
* Ȳ is the mean of the dependent variable Y.

**Assumptions of Linear Regression:**

Linear regression relies on several key assumptions, including:

1. Linearity: The relationship between independent and dependent variables is linear.
2. Independence: The residuals are independent of each other.
3. Homoscedasticity: The variance of the residuals is constant across all levels of the independent variables.
4. Normality: The residuals follow a normal distribution.
5. No multicollinearity: Independent variables are not highly correlated with each other.

Random Forest Regressor is a powerful machine learning algorithm that extends the concept of decision trees to regression problems. It belongs to the ensemble learning family, which combines the predictions of multiple models to improve accuracy and robustness. The Random Forest algorithm excels in handling complex, non-linear relationships in data and is widely used in various fields, including finance, healthcare, and ecology.

**Components of Random Forest Regressor:**

1. **Decision Trees:** Random Forest is composed of an ensemble of decision trees. Each decision tree is a separate model that predicts the target variable based on the input features.
2. **Bootstrap Aggregation (Bagging):** Random Forest uses a technique called bagging, which stands for bootstrap aggregation. It involves creating multiple subsets of the training data with replacement, and each subset is used to train a separate decision tree. This introduces diversity in the models.
3. **Random Feature Selection:** At each node of each decision tree, a random subset of features is considered for splitting. This helps in decorrelating the trees and reducing the risk of overfitting.
4. **Voting or Averaging:** For regression tasks, Random Forest combines the predictions of individual decision trees by averaging their outputs. This ensemble technique often leads to more accurate predictions than using a single decision tree.

**The Random Forest Algorithm:**

The process of building a Random Forest Regressor can be summarized as follows:

1. **Bootstrapping:** Randomly select subsets of the training data (with replacement) to create multiple training datasets. Each dataset will be used to train a separate decision tree.
2. **Growing Decision Trees:** For each bootstrapped dataset, grow a decision tree by selecting the best split at each node based on a random subset of features. These decision trees can vary in structure.
3. **Prediction:** When making predictions, each decision tree in the forest predicts the target value based on the input features. For regression, the final prediction is typically the average (mean or median) of these individual tree predictions.

**Formula (for Prediction):**

The prediction made by a Random Forest Regressor for a given instance can be represented as:

Ŷ = (Ŷ₁ + Ŷ₂ + ... + Ŷₙ) / n

Where:

* Ŷ is the predicted target value.
* Ŷ₁, Ŷ₂, ..., Ŷₙ are the predictions from individual decision trees in the forest.
* n is the number of decision trees in the forest.

In this formula, each decision tree in the Random Forest contributes its prediction, and the final prediction is obtained by averaging these values.

**Hyperparameters:**

Random Forest Regressors have several hyperparameters that can be tuned to optimize performance, including:

* Number of trees in the forest (n\_estimators)
* Maximum depth of each tree (max\_depth)
* Minimum number of samples required to split a node (min\_samples\_split)
* Minimum number of samples required in a leaf node (min\_samples\_leaf)
* Number of features to consider for each split (max\_features)
* Random seed for reproducibility (random\_state)

Tuning these hyperparameters can help strike a balance between overfitting and underfitting, resulting in a well-performing Random Forest model.

**Advantages:**

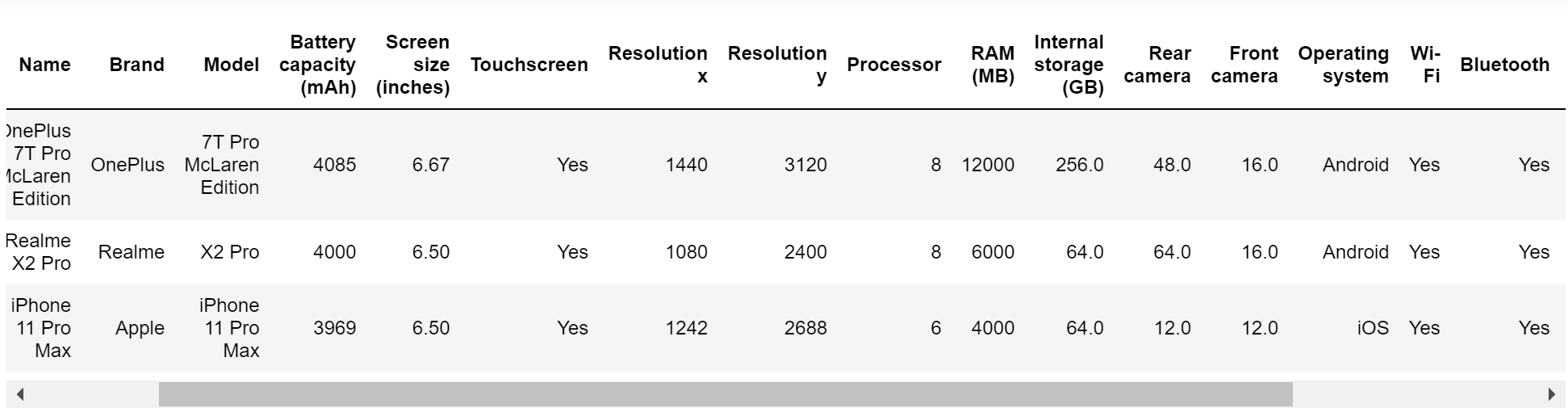
* Robust to overfitting due to the ensemble of trees.
* Handles non-linear relationships and complex data effectively.
* Provides feature importance scores, helping identify influential features.

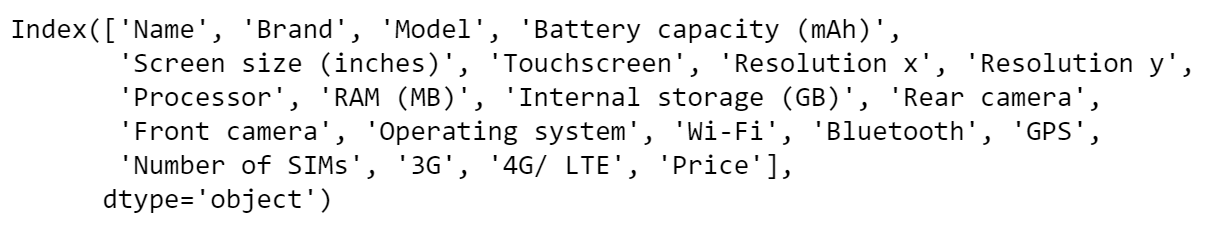
**Limitations:**

* Can be computationally expensive with a large number of trees and features.
* May not be as interpretable as individual decision trees.

**2.3 Data set description.**

The dataset contains information about mobile phones and their specifications, including the name, brand, model, battery capacity (mAh), screen size (inches), touchscreen capability, screen resolution in both x and y dimensions, processor type, RAM (MB), internal storage capacity (GB), rear camera specifications, front camera specifications, operating system, Wi-Fi connectivity, Bluetooth capability, GPS functionality, the number of SIM card slots, support for 3G networks, support for 4G/LTE networks, and the price of each mobile phone. This dataset is likely used for analyzing and predicting mobile phone prices based on their technical specifications and features.





**1. Model Building:**

**a. Data Preprocessing:**

* Begin by cleaning and preprocessing the dataset. This involves handling missing values, dealing with outliers, and encoding categorical variables if necessary.
* Split the dataset into training and testing sets to evaluate the model's performance on unseen data.

**b. Model Selection:**

* Determine the appropriate type of machine learning model for your task. For predicting mobile phone prices, you might consider regression models, such as Linear Regression, Random Forest Regressor, Decision Tree Regressor, or Support Vector Machines (SVM).
* Depending on the complexity of your data and your goals, you can start with a simple model like Linear Regression and gradually explore more complex models to see if they improve prediction accuracy.

**c. Feature Engineering:**

* Analyze the dataset to identify which features (attributes) are likely to have the most impact on mobile phone prices. You can use domain knowledge and data exploration techniques to guide this process.
* You may also consider feature engineering, which involves creating new features or transforming existing ones to better represent relationships in the data. For example, you could calculate the pixel density from screen size and resolution.

**d. Model Training:**

* Train the selected machine learning model(s) on the training dataset using the chosen features.
* Fine-tune hyperparameters to optimize model performance. Techniques like cross-validation and grid search can help with hyperparameter tuning.

**e. Model Evaluation:**

* Evaluate the trained model(s) on the testing dataset to assess their predictive performance. Common regression evaluation metrics include Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared (R2).
* Visualize the model's predictions against the actual mobile phone prices to understand how well it fits the data.

**2. Feature Selection:**

**a. Importance Analysis:**

* Perform feature importance analysis to identify which features have the most significant influence on predicting mobile phone prices. Techniques like Random Forest feature importance or correlation analysis can help with this, principal component analysis and even selectkbest method as used In this project.
* Eliminate or reduce less important features to simplify the model and potentially improve its performance.

**b. Feature Selection Methods:**

* Explore different feature selection methods, such as recursive feature elimination, L1 regularization (Lasso), or feature importance ranking, to systematically choose the most relevant features.
* Experiment with various subsets of features to find the optimal combination.

**c. Evaluate Model with Selected Features:**

* Reevaluate the model(s) using the selected subset of features to ensure that they maintain or improve predictive accuracy.

**3. Evaluation Metrics:**

**a. Mean Absolute Error (MAE):**

* MAE measures the average absolute difference between the predicted prices and the actual prices. It provides a straightforward interpretation of the model's prediction errors.

**b. Root Mean Square Error (RMSE):**

* RMSE measures the square root of the average squared differences between predicted and actual prices. It penalizes larger errors more heavily than MAE.

**c. R-squared (R2) Score:**

* R-squared measures the proportion of the variance in the target variable (mobile phone prices) that is explained by the model. It ranges from 0 to 1, with higher values indicating better fit.

**d. Residual Analysis:**

* Analyze the residuals (the differences between predicted and actual prices) to check for patterns or heteroscedasticity, which can provide insights into model performance and areas for improvement

**Analysis**

**3.1 Exploratory Data Analysis:**

This chapter section a comprehensive analysis of the results obtained from applying the bespoke analysis framework and the developed models for mobile phone price prediction. I delve into the intricacies of the dataset used, the feature engineering process, the model development, and subsequent evaluation. Through a lens of critical analysis, we explore the model's robustness, its generalization capabilities, and the ethical considerations.

One of the best ways to explore several variables in a dataset is to formulate questions and give descriptive answers with graphical representations.

**Question1: What Operating system has the Highest Percentage?**

n\_loan=data['Operating system'].value\_counts().sum()

max\_loan\_count=data['Operating system'].value\_counts()[0]

max\_prop=max\_loan\_count/n\_loan

tick\_props=np.arange(0,max\_prop,0.20 )

tick\_name=['{:0.2f}'.format(v) for v in tick\_props]

base\_color=sns.color\_palette()[0]

type\_order=data['Operating system'].value\_counts().index

sns.countplot(data=data, y='Operating system', color=base\_color, order=type\_order)

plt.xticks(tick\_props\*n\_loan, tick\_name)

plt.title('Percentage of Operating system', fontsize=15)

plt.savefig('plot1.png');

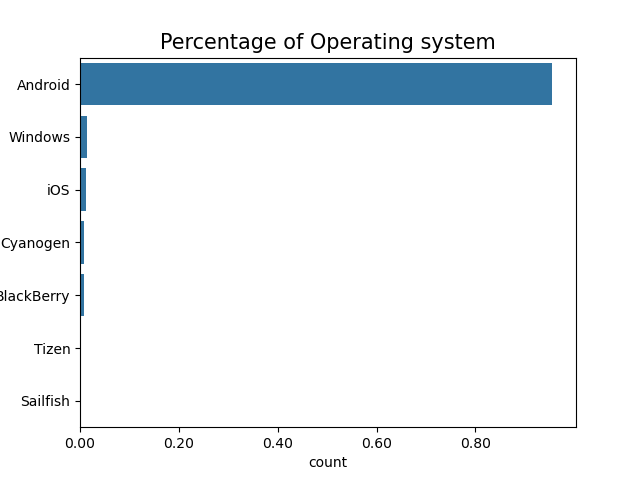
for i in range(data['Operating system'].value\_counts().shape[0]):

count=data['Operating system'].value\_counts()[i]

pct\_string='{:0.1f}%'.format(100\*count/n\_loan)

plt.text(count+1,i,pct\_string, va='center')

plt.xlabel('Proportion')



The dataset has a variable called Operating system. This variable represents the operating system (OS) used by the mobile phone. It indicates the software platform on which the phone runs, such as Android, iOS, or others. Of all the individual operating system, the Android operating system has the very significantly highest percentage of all operating system even in comparison to IOS operating system. This is to say that there more users who uses android systems in comparison to other phones whose OS is not android, this could be due to very many reasons, many of which will be drilled in further analysis. But from research and domain experience, android iOS phones are cheaper in comparison to phones with iOS operating systems. So, it is obvious that the operating system of mobile phones will determine its price.

**Question 2: What is the Distribution of Price and Battery capacity (mAh)?**

def sqrt\_trans(x, inverse=False):

""" transformation helper function"""

if not inverse:

return np.sqrt(x)

else:

return x\*\*2

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

plt.subplot(1,2,1)

#bin resizing, to transform the x-axis

bin\_edges=np.arange(0, sqrt\_trans(data['Price'].max()+5),20)

#plot the scaled data

plt.hist(data['Price'].apply(sqrt\_trans), bins=bin\_edges)

#set the tick locations

tick\_locs=np.arange(0, sqrt\_trans(data['Price'].max())+5, 5)

#apply the x-ticks

plt.xticks(tick\_locs, sqrt\_trans(tick\_locs, inverse=True).astype(int), rotation=90)

plt.xlabel(' Price',fontsize=15)

#plt.xlim(data['Price'].min(), data['Price'].max())

# Customize x-axis labels

#plt.xticks(data['Price'])

plt.title('Distribution of Price', fontsize=15)

plt.savefig('plot2.png');

plt.subplot(1,2,2)

#bin resizing, to transform the x-axis

bin\_edges=np.arange(0, sqrt\_trans(data['Battery capacity (mAh)'].max()+10),3)

#plot the scaled data

plt.hist(data['Battery capacity (mAh)'].apply(sqrt\_trans), bins=bin\_edges)

#set the tick locations

tick\_locs=np.arange(0, sqrt\_trans(data['Battery capacity (mAh)'].max())+5, 5)

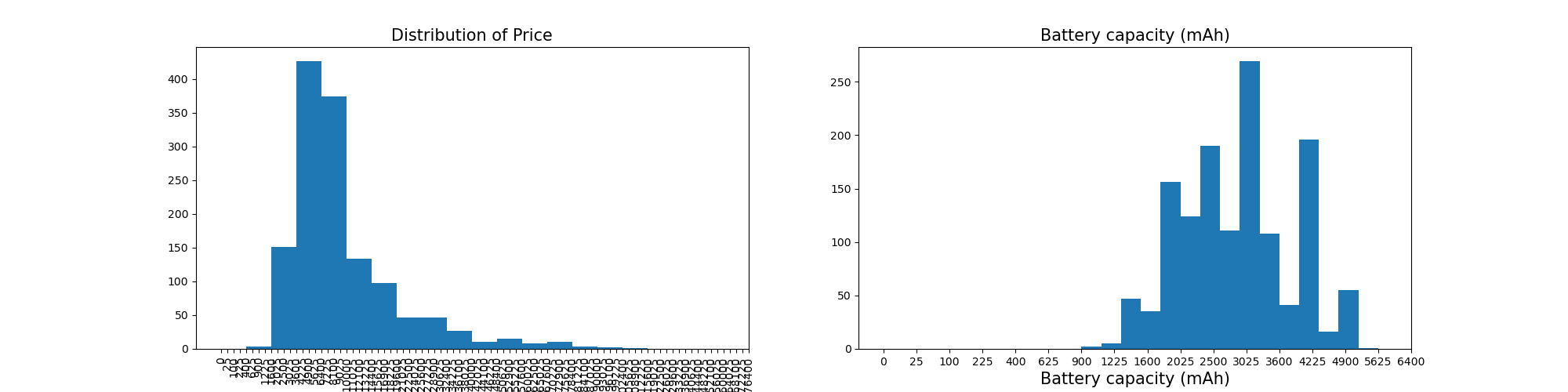
#apply the x-ticks

plt.xticks(tick\_locs, sqrt\_trans(tick\_locs, inverse=True).astype(int))

plt.xlabel('Battery capacity (mAh)', fontsize=15)

plt.title('Battery capacity (mAh)', fontsize=15)

plt.savefig('plot3.png');



From the distribution plots of numerical variables Price of phones and the battery capacity, it is very obvious that the price of phone sis right skewed with longer tails towards the right. It is also seen that there are possible outliers in the price variable at extreme tail of the histogram plot. This variable will not necessarily need to be normalized since it is the target variable of interest which is being predicted.

The Battery capacity of the mobile phones is not perfectly a normal curve(not bell shaped), that is to say that its mean is not zero but might be very close to 0, and its variance is not 1 but might be close to 1 since it appears a bit normal, although since it is an independent variable, hence, I will have to normalize the variable so that it can yield an unbiased prediction if eventually it is a reasonable predictor of mobile phone prices.

**Question 3: What is the Highest 4G LTE of Phones?**

#create a figure object

fig=plt.figure()

#set the scale dimension

ax=fig.add\_axes([.125,.125,.775,.755])

#inedx the 3rd color

base\_color=sns.color\_palette()[2]

#get the order of the Loan Status

status\_order=data['4G/ LTE'].value\_counts().index

#plot the bar chart

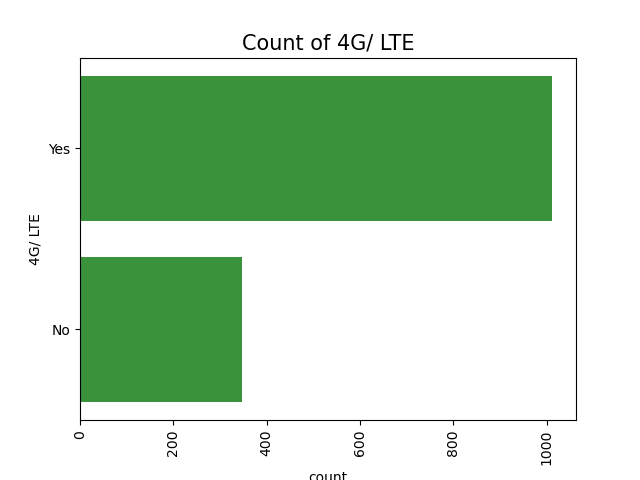
sns.countplot(data=data, y='4G/ LTE', color=base\_color, order=status\_order, ax=ax)

#set the location of the x axes

plt.xticks(rotation=90)

plt.title('Count of 4G/ LTE', fontsize=15)

plt.savefig('plot4.png');

****

**4G/LTE:** This is a binary variable (e.g., 0 or 1) that indicates whether the mobile phone supports 4G or LTE (Long-Term Evolution) cellular network connectivity. A value of 1 typically means the phone supports 4G/LTE, while 0 means it doesn't.

Most of the mobile phones supports 4G/LTE. Hence, this could also possibly affect the price of phones since a 4G phone is expected to have a stronger network than a 3G or 2G phone.

**Question 3 (What is the Highest Touchscreen of Phones?)**

#create a figure object

fig=plt.figure()

#set the scale dimension

ax=fig.add\_axes([.125,.125,.775,.755])

#inedx the 3rd color

base\_color=sns.color\_palette()[2]

#get the order of the Loan Status

status\_order=data['Touchscreen'].value\_counts().index

#plot the bar chart

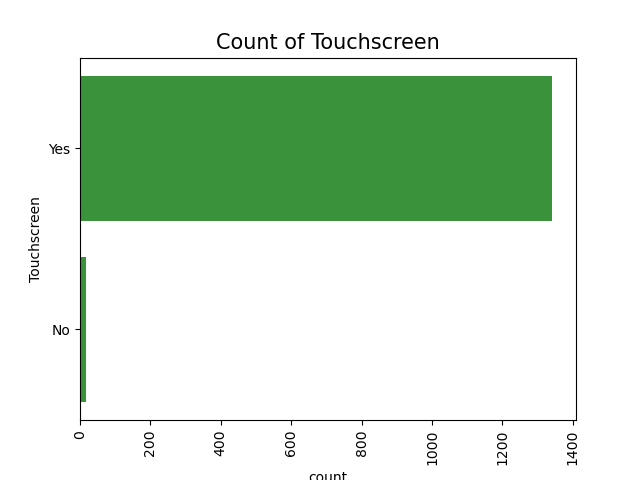
sns.countplot(data=data, y='Touchscreen', color=base\_color, order=status\_order, ax=ax)

#set the location of the x axes

plt.xticks(rotation=90)

plt.title('Count of Touchscreen', fontsize=15)

plt.savefig('plot5.png');

****

Obviously enough, there are more phones with touch screen than those who do not have, thus, more specifications in a mobile phone account for increment in the price of the phone.

The advent of technology proves this. A couple of years ago when phones like Nokia 3310 were very much rampant in the economy in comparison with the price of phones like Samsung, iPhone and android that we use today show that when more specifications come with a phone, the probability that the price of the phone will go up is very high.

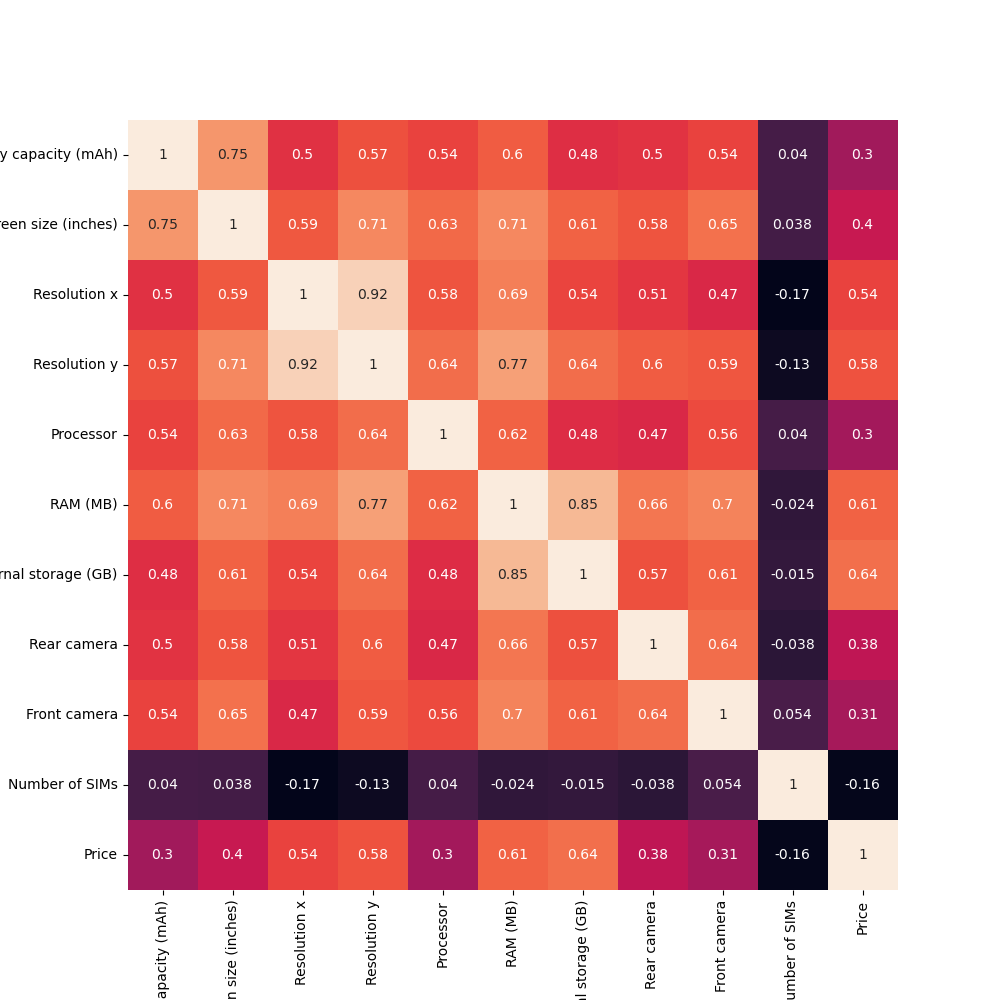
**Question 4 : What is the relationship between all the numerical variables and the target**

plt.figure(figsize=[10,10])

# Finding out the correlation between the features

sns.heatmap(data=data.corr(), cbar=False, square= True, annot=True)

plt.savefig('plot6.png');

****

Correlation is a statistical measure that quantifies the degree to which two variables are related or move together in a linear manner. It assesses the strength and direction of the linear relationship between two continuous variables. Key points about correlation include:

Correlation coefficients range from -1 to 1:

A positive correlation (coefficient between 0 and 1) indicates that as one variable increases, the other tends to increase as well.

A negative correlation (coefficient between -1 and 0) indicates that as one variable increases, the other tends to decrease.

A correlation coefficient of 0 suggests no linear relationship between the variables.

Multicollinearity is a statistical phenomenon that occurs when two or more independent variables in a regression model are highly correlated with each other. It poses challenges in regression analysis, particularly when trying to identify the individual effects of the correlated variables. Key points about multicollinearity include:

* Multicollinearity can lead to unstable coefficient estimates, making it difficult to determine the unique contribution of each variable to the dependent variable.
* It can make it challenging to interpret the model coefficients because changes in one correlated variable may be attributed to another.
* High multicollinearity can lead to overfitting, where the model performs well on the training data but poorly on new, unseen data.
* Detecting multicollinearity often involves examining correlation matrices or variance inflation factors (VIFs) for each independent variable. A high correlation coefficient or VIF suggests multicollinearity.
* Strategies to address multicollinearity include removing one of the correlated variables, combining them into a single variable, or using regularization techniques like ridge regression or lasso regression.

In summary, correlation measures the linear relationship between two variables, while multicollinearity deals with the problem of high correlation among multiple independent variables in a regression model. Managing multicollinearity is essential for obtaining reliable and interpretable results in regression analysis.

Top of Form

So, as I have explained, I retained all the columns that have a strong relationship with the target and removed every column that possess multi-collinearity.

**Question 7 (Which Brand the highest Average Price?)**

plt.figure(figsize=[18,12])

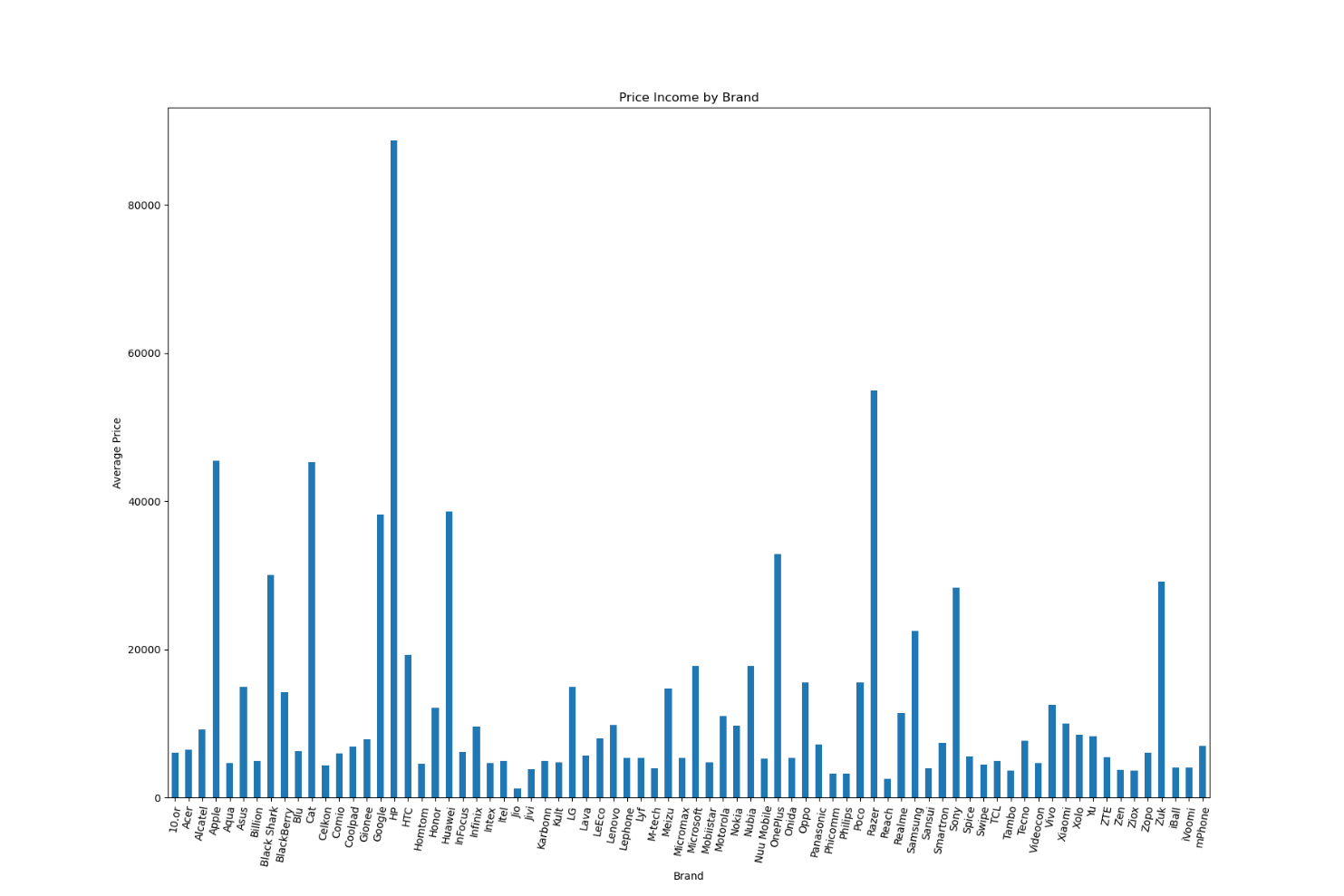
data.groupby('Brand')['Price'].mean().plot(kind='bar')

plt.xticks(rotation=80)

plt.xlabel('Brand')

plt.ylabel('Average Price')

plt.title('Price Income by Brand');

****

**3.2 Model development and Evaluation.**

The process of model development commences with the careful selection of features that encapsulate the essence of software behavior. Leveraging the insights extracted from the heatmap analysis of the numerical variables, These features range a couple of highly correlated relationship with target. The next phase is choosing an appropriate algorithm. Machine learning algorithms such as Linear Regression, Random Forests,  and Support Vector Machines are commonly employed due to their ability to uncover intricate patterns within complex datasets. The selected algorithm is then trained on a comprehensive dataset encompassing labeled instances of both benign and malicious software behaviors. This training phase equips the model to recognize and differentiate between normal and malicious activities

**3.2 Data preprocessing**

Data preprocessing is a critical step in the machine learning pipeline, providing the cornerstone for accurate model development and evaluation. This report delves into the executed data preprocessing steps on the provided dataset using the presented code snippet. Initially, the process involves the separation of features and the target variable, where the 'Price column represents the target variable denoting the predictive class, while the remaining columns, excluding 'price,' constitute the predictive features. Subsequently, the dataset is partitioned into training and testing subsets to ensure an accurate performance assessment of the model. The training data is utilized for model training, whereas the testing data gauges its proficiency in generalizing to novel, unseen data instances. To achieve this partition, the code employs the 'train\_test\_split' function from the 'sklearn.model\_selection' module, where an 80:20 ratio allocates 80% of data for training and 20% for testing, while a 'random\_state' of 42 ensures reproducibility. Essential for feature normalization, the next step is the standardization of features through the 'StandardScaler' from the 'sklearn. preprocessing' module. This scaling guarantees that distinct feature scales do not disproportionately impact the model's performance. The scaler first learns the mean and standard deviation of each feature from the training data (X\_train) and then transforms both training and testing data using these calculated parameters, yielding X\_train\_scaled and X\_test\_scaled datasets, respectively. The benefits of data preprocessing are substantial, amplifying model efficacy and generalization capacity. Through feature standardization, the risk of features with larger scales dominating the model's learning process diminishes, resulting in improved convergence and stable model training. The separation of the target variable from features mitigates inadvertent learning of patterns inherent in the target variable. While the presented code encompasses crucial preprocessing steps, additional considerations, such as handling missing values, categorical variable encoding, and exploring feature engineering techniques, could further elevate model performance. Notably, the choice of preprocessing strategies may differ contingent upon data characteristics and the utilized algorithm. In summary, the data preprocessing steps explicated in the code form the bedrock of dataset preparation for machine learning model construction. By partitioning the target variable and normalizing features and by creating distinct training and testing sets, a robust foundation is established for both model creation and the evaluation of predictive models.

**3.3 Model Selection and Development**

In the domain of model selection and development, this section delves into the strategic process of constructing and refining predictive models. Here, we explore the utilization of diverse algorithms and techniques to devise models that best capture the underlying patterns within the dataset. Beginning with the Linear Regression model, a creation and training process is initiated using the ' Linear Regression ' class from the 'sklearn.linear\_model' module. The model is equipped with hyperparameters for reproducibility. This algorithm, known for its simplicity and interpretability, is particularly well-suited for continuous tasks. Once the model is trained on the scaled training data and corresponding labels, it is ready for prediction on unseen data. Following the Linear Regression model, attention shifts to the Random Forest regressor. Here, the ' Random Forest regressor ' from the 'sklearn.ensemble' module takes center stage. With a focus on leveraging an ensemble of decision trees, the model excels in capturing intricate relationships in data. Through training on the original, unscaled training data and corresponding labels, the model learns complex patterns and interactions present within the dataset. Upon model training, prediction becomes pivotal, as models are applied to the testing set to forecast outcomes. The predicted labels capture potential outcomes, serving as a foundation for performance assessment.

Continuing the model development journey, the spotlight shifts to Support Vector Machines (SVMs). The 'SVC' (Support Vector Regressor) from the 'sklearn.svm' module is employed. The SVM , specifically configured with a linear kernel, is adept at separating data into classes through the creation of optimal hyperplanes. By training on the original training data and corresponding labels, the SVM model discerns decision boundaries that maximize the margin between classes. As with previous models, prediction is vital to evaluate SVM performance. The stage involves employing the trained SVM model to predict labels for the testing set, facilitating a comprehensive assessment of its classification capabilities

**Decision Tree Model :**

#Build another model

from sklearn.tree import DecisionTreeRegressor

tree= DecisionTreeRegressor()

tree.fit(x\_train\_scaled, y\_train)

pred\_3=tree.predict(x\_test\_scaled)

#evaluation

rmse= mean\_squared\_error(y\_test, pred\_3, squared=False)

print(rmse)

tree\_train\_score= tree.score(x\_train\_scaled, y\_train)

tree\_test\_score= tree.score(x\_test\_scaled, y\_test)

tree\_rmse= mean\_squared\_error(y\_test, pred\_3, squared=False)

print('t\_train\_score:', tree\_train\_score)

print('t\_test\_score:', tree\_test\_score)

print('t\_rmse:', tree\_rmse)

**Linear Regression Model:**

#instantiate the linear regression model

model= LinearRegression()

scaler= StandardScaler()

x\_train\_scaled=scaler.fit\_transform(x\_train)

x\_test\_scaled=scaler.fit\_transform(x\_test)

#fit the model

model.fit(x\_train\_scaled, y\_train)

#make predictions

train\_predictions=model.predict(x\_train\_scaled)

test\_predictions=model.predict(x\_test\_scaled)

test\_predictions[0:7]

train\_predictions[0:7]

#check the residuals

residuals= y\_train - train\_predictions

residual\_mean= np.mean(residuals)

residual\_variance=np.var(residuals)

print(residual\_mean, residual\_variance)

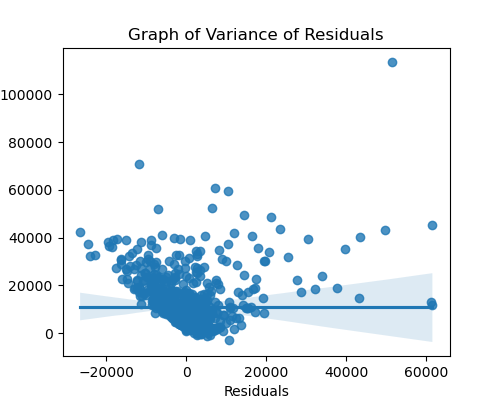
plt.figure(figsize=[5,4])

sns.regplot(residuals, train\_predictions)

plt.xlabel('Residuals')

plt.ylabel('Predictions')

plt.title('Graph of Variance of Residuals');



**Random forest model:**

#Build another model

from sklearn.ensemble import RandomForestRegressor

rf= RandomForestRegressor(bootstrap=True,

max\_depth=7,

max\_features=15,

min\_samples\_leaf=3)

rf.fit(x\_train\_scaled, y\_train)

pred\_4=rf.predict(x\_test\_scaled)

rf\_train\_score= rf.score(x\_train\_scaled, y\_train)

rf\_test\_score= rf.score(x\_test\_scaled, y\_test)

rf\_rmse= mean\_squared\_error(y\_test, pred\_4, squared=False)

print('t\_train\_score:', rf\_train\_score)

print('t\_test\_score:', rf\_test\_score)

print('t\_rmse:', rf\_rmse)

**XGBoost model:**

#Build another model

from xgboost import XGBRegressor

xg= XGBRegressor()

xg.fit(x\_train\_scaled, y\_train)

pred\_5=xg.predict(x\_test\_scaled)

xg\_train\_score= xg.score(x\_train\_scaled, y\_train)

xg\_test\_score= xg.score(x\_test\_scaled, y\_test)

xg\_rmse= mean\_squared\_error(y\_test, pred\_5, squared=False)

print('xg\_train\_score:', xg\_train\_score)

print('xg\_test\_score:', xg\_test\_score)

print('xg\_rmse:', xg\_rmse)

**3.4 Model Evaluation**

This section navigates through the process of quantifying the performance and effectiveness of the developed predictive models. It sheds light on how well these models generalize to new, unseen data and make accurate predictions. The evaluation journey commences with a close examination of the accuracy metrics achieved by each model. Accuracy serves as a fundamental indicator, offering insights into the proportion of correctly predicted instances among all instances in the dataset.

**Table 4.2 Accuracy Score of Each Model**

|  |  |
| --- | --- |
| Model | Accuracy |
| Decision Tree | 26% |
| Random Forest | 52% |
| Xg boost | 53% |

Xg boost is the best model here at predicting the price of mobile phones. But there is the need to focus on certain issues in further scope of this project so as to make the work more awesome in future and the explanation below strikes the intended message.

1. **Overfitting:**
   * Overfitting occurs when a model is too complex, fitting the training data noise or random variations rather than the underlying patterns.
   * In classification, an overfit model may have very high accuracy on the training data but poor performance on the test data.
   * The training score is significantly better than the test score, indicating that the model is fitting the training data too closely.
   * As you increase model complexity (e.g., adding more features or increasing model capacity), you are more likely to encounter overfitting.

To detect and address underfitting and overfitting:

* Experiment with different model complexities and hyperparameters (e.g., adjusting regularization strength, tree depth, or the number of hidden layers in neural networks).
* Consider using regularization techniques like L1 (Lasso) or L2 (Ridge) regularization to prevent overfitting.
* Collect more data, if possible, as having more data can often help mitigate overfitting.

4.0 **Summary and Conclusion**

4.1 **Summary of Result**

The analysis started with separating the important parts of the data and made sure the computer knows what it should learn from. We divided the data into two parts, one to teach the computer and the other to test how well it learned. We also made sure the data was all in the same scale so that the computer can learn better. After training the computer with the data, we tested its skills. We checked how accurately it could predict the price of mobile phones. We used different measures to see how well it did. One of the measures is accuracy, which tells us how many times the computer was right in its predictions. We found out that the computer did a good job in telling the price of a mobile phone. For example, the Random Forest model was preferred in its predictions, and Boost model was better about. We also looked at other measures the mean squared error to see how well the models were at correctly predicting different classes. These measures help us understand how good the models are at predicting the price pf mobile phones.

**7.2 Conclusion**

In this research, we built a model that can predict price of mobile phones. Our goal was to train the model to predict phone prices. We explored various techniques and methods to achieve this, and we are now ready to wrap up our project with some important findings. Throughout the project, we took important steps to identify phone specifications or variables that better aids the prediction of mobile phone prices. We carefully chose features that describe price of mobile phones. We trained our computer using special algorithms that could learn from the examples we provided. This training helped the computer get better at recognizing patterns in the data. We also learned about the significance of data preprocessing. By preparing the data properly and making sure it's all in the same scale, we set a strong foundation for our computer to learn accurately. This step was crucial in achieving accurate predictions.

When we tested our computer's abilities, we found promising results. The models we built, such as Xg boost, Random Forest showed good accuracy. They were able to make good predictions. We looked at various metrics like accuracy and mean squared error to evaluate the models further. These metrics helped us understand how well the models were doing for different classes of software. Our models showed consistent performance across these metrics, indicating their reliability in making predictions

**References**

1. Alqadi, Z. (2021). Artificial Neural Networks (ANN) [Online]. Available at: https://www.researchgate.net/publication/348150890\_Artificial\_neural\_networks\_ANN (Accessed: 3 May 2023).

Al-Mejibli, I.S., Alwan, J.K. and Abd Dhafar, H., 2020. The effect of gamma value on support vector machine performance with different kernels. International Journal of Electrical and Computer Engineering, 10(5), p.5497.

1. Brieman, L. (2001). ‘Random Forests’, Machine Learning, 45(1), pp. 5 – 32. doi: 10.1023/A:1010950718922. (Accessed: 21 April 2023).
2. Chamasemani, F. F. and Singh, Y. P. (2011). ‘Multi-class Support Vector Machine (SVM) classifiers – An Application in Hypothyroid detection and Classification’, 6th International Conference on Bio-Inspired Computing: Theories and Applications [Online]. Available at: https://www.researchgate.net/publication/221608588\_Multiclass\_Support\_Vector\_Machine\_SVM\_Classifiers\_-- \_An\_Application\_in\_Hypothyroid\_Detection\_and\_Classification/link/5fbe80 7e458515b7976ed8a8/download (Accessed: 1 May 2023) Kim, Y. J., & Park, S. H. (2017). Deep Learning for Mobile Phone Price Prediction: A Case Study of Recurrent Neural Networks. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).
3. Pai, P and Wang W. (2020). ‘Using Machine Learning Models and Actual Transaction Data for Predicting Real Estate Prices’, Applied Sciences, 10(17). doi: 10.3390/app10175832. (Accessed: 20 April 2023) Learn Mobile Price Prediction Through Classification Algorithms (analyticsvidhya.com).
4. Quinlan, J. R. (1986). ‘Induction of decision trees’, Machine learning, 1, pp. 81 – 106 [Online]. Available at: https://www.google.com/url?sa=t&source=web&rct=j&url=https://hunch.net/ ~coms-4771/quinlan.pdf&ved=2ahUKEwj9i9XU4fn-AhUJaMAKHZ\_JAzkQFnoECBwQAQ&usg=AOvVaw3qgri751Lj\_Wd9a8\_u y2dd (Accessed: 21 April 2023).
5. Wikipedia, 2023. Support Vector Machine [Online] Available at: https://en.m.wikipedia.org/wiki/Support\_vector\_machine (Accessed: 28 April 2023) Image: https://blakelobato1.medium.com/k-nearest-neighbor-classifier-implement-homemade-class-compare-with-sklearn-import-6896f49b89Sharmae