HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Capstone project report

Phone price prediction

Submitted by

Ngo Trung Dung 20225487

Nguyen Tung Phong 20225517

Tran Thi Hien 20214896

Nguyen Viet Long 20225506

Bui Thi Thu Uyen 20225537

Guided by

Assoc.Prof. Than Quang Khoat

Hanoi, December 2024

Abstract

In this project, we dive into the fascinating world of machine learning to predict smartphone prices based on their technical specifications. With the growing complexity of the smartphone market, it can be tough for both consumers and manufacturers to make smart decisions when it comes to pricing. So, we decided to put machine learning to the test and see if it could provide some clarity.

We gathered a diverse dataset of over 800 smartphone models from well-known brands like Apple, Samsung, Xiaomi, and others. After cleaning and preprocessing the data, we experimented with several popular machine learning models, including Random Forest, Gradient Boosting, XGBoost, and a Stacking Regressor model. Each model was trained on the data to predict the prices, and we evaluated their performance based on two key metrics: Mean Squared Error (MSE) and R-squared (R^2).

XGBoost came out on top, delivering the most accurate predictions with an MSE of 17,634.4 and R^2 of 0.79. Not far behind was the Stacking Regressor, which showed strong performance but with slightly less precision. Random Forest and Gradient Boosting, while still decent, struggled a bit more with handling the non-linear patterns in the data.

In the end, this project shows how machine learning can help tackle real-world problems like smartphone price prediction. It not only allows for smarter consumer choices but also gives manufacturers valuable insights into pricing strategies. Looking ahead, we plan to improve the model by expanding the dataset and incorporating more features, such as brand reputation and market trends, to make the predictions even more accurate.

Contents

Part I
Introduction

1 Project Background

In the era of rapid technological development, smartphones have become an indispensable part of modern life. Mobile devices not only meet communication needs but also serve purposes such as entertainment, work, and learning. However, with thousands of models from major brands such as Apple, Samsung, Xiaomi, Oppo, and more, determining reasonable prices for each device based on technical specifications has become a significant challenge.

The lack of price transparency not only affects consumers in choosing products but also creates difficulties for manufacturers and retailers in devising appropriate pricing strategies. This has created an urgent need to apply advanced technologies such as data science and machine learning to address this issue.

This project aims to build a predictive model for smartphone prices based on detailed technical specifications. The model will help consumers evaluate product value while supporting businesses in making strategic and effective pricing decisions.

2 Objectives and Scope of Application

This project focuses on applying machine learning and data science methods to solve the problem of predicting smartphone prices based on technical specifications. The primary objectives and scope of application of the project include:

2.1 Project Objectives

- **Build an accurate machine learning model for price prediction:**
  - Develop and test multiple machine learning algorithms to identify the optimal model, using input features such as RAM, internal storage, screen size, and battery capacity.
  - Evaluate model performance using metrics such as MSE (Mean Square Error), and R^2 (coefficient of determination).
- **Create a tool to assist in product value assessment:**
  - Help consumers assess the true value of products and compare smartphone models to make informed purchasing decisions.

  - Provide manufacturers and retailers with a tool to optimize pricing strategies, ensuring competitive and market-appropriate pricing.

- **Promote research and practical applications:**
  - Lay the groundwork for research in machine learning and commerce, particularly for problems related to product pricing and market analysis.
  - Develop an application or online tool for smartphone price prediction, integrating it into e-commerce platforms or price management systems.

2.2 Scope of Application

The project not only focuses on solving the price prediction problem but also expands its applicability to various areas and societal aspects:

- **Consumers:**
  - Evaluate the true value of products and find the smartphone model that best fits their budget.
  - Increase price transparency by clarifying the relationship between technical specifications and product value.
- **Manufacturers and Retailers:**
  - Support product pricing based on actual value and competitiveness.
  - Identify target customer segments and devise pricing strategies for different product lines.
  - Develop effective marketing strategies by highlighting the distinctive features of products within each price segment.
- **Research and Deployment Fields:**
  - Analyze factors influencing smartphone prices, from technical specifications to non-technical aspects such as brand reputation, user reviews, and market trends.
  - Expand machine learning applications to other fields, such as pricing technology products, optimizing sales prices, or market analysis.
- **E-commerce and Price Management Systems:**
  - Integrate the price prediction model into e-commerce platforms such as Shopee, Lazada, and Amazon to help users find products that fit their budget.
  - Create an easy-to-use price prediction tool for retailers or product managers.

The objectives and scope of the project are designed to provide a comprehensive solution for product pricing challenges in the technology industry while opening opportunities to expand machine learning applications to other domains.

Part II
Data

1 Data Sources

The data used in this project were collected from a reputable technology website, notably **GSMArena** [3]. This website provides detailed information on technical specifications, pricing, and features of numerous smartphone models from major brands such as **Apple, Samsung, Xiaomi, Huawei, Oppo, Realme, Asus**, etc.

2 Data Collection

The data were collected using Python's **Scrapy** [5] library to automate the extraction process. The scraping process involved sending HTTP requests to retrieve URLs for individual smartphone models and extracting specifications using *XPath*. However, the scraping process encountered challenges such as **IP blocking** due to high request frequency and bot detection mechanisms.

2.1 Web Scraping Strategies

To overcome these challenges:

- **Request Delays:** Random delays were introduced between requests to simulate human-like browsing behavior.
- **Rotating User-Agents:** Various user-agents were rotated to mimic real users, including different browsers and devices. This involved sending HTTP headers with randomized user-agent strings.
- **Proxy Rotation:** Free rotating proxies were used to minimize the likelihood of IP blocks, though they were found to be unreliable.
- **Third-Party API:** As a final solution, a third-party API was used to scrape data directly, which significantly improved the data collection process with fewer IP blocks.

2.2 Result of Web Scraping

The web scraping process resulted in a dataset comprising **833 smartphone models** across various brands. This dataset includes comprehensive technical specifications and price information. A breakdown of the collected data by brand is presented in Table 1.

The dataset comprises models from premium brands such as **Apple** and **Samsung** as well as budget-friendly brands like **Xiaomi** and **Realme**. This diversity ensures the robustness of the predictive models across different market segments.

Table 1 . Number of Smartphones Collected by Brand

Phone Brand	Number of Phones	Phone Brand	Number of Phones
Realme	128	Oppo	112
Xiaomi	120	Samsung	110
Alcatel	99	Apple	93
Huawei	93	Asus	78

3 Data Description

The dataset is organized in a tabular format where each record corresponds to a smartphone model and the attributes describe its technical specifications. It includes several hardware and software features, screen details, and supplementary information relevant to each device.

3.1 Hardware Specifications

- **RAM:** The RAM capacity of each device is measured in gigabytes (GB). The majority of smartphones in the dataset have 1GB to 512GB of RAM.
- **Internal Storage:** Storage capacity is also measured in gigabytes (GB). Common sizes are 128GB and 256GB.
- **CPU:** The number of CPU cores varies, with high-end devices typically featuring more powerful processors.
- **Battery:** Battery capacity (in mAh) varies from 307 mAh to 11200 mAh, with an average of 5010.51 mAh.

3.2 Display Specifications

- **Size:** Screen diagonal size is measured in inches, with most devices having a screen size between 5 and 7 inches.
- **Resolution:** Measured in pixels, with most smartphones having a resolution of 1080 x 2400 pixels.

3.3 Connectivity and Additional Features

- **Connectivity:** Includes support for 5G, Wi-Fi, Bluetooth, and NFC.

3.4 Supplementary Information

- **Price:** Listed in USD, with a range from 40 USD to 2099 USD, and an average of 342.95 USD.
- **Brand and Model:** The dataset includes the brand name and model name for each smartphone.

4 Data Preprocessing

Data preprocessing involves cleaning and standardizing the dataset to prepare it for analysis and model training. Key steps include:

4.1 Data Cleaning

- Remove incomplete records or those missing essential information.
- Handle invalid or inconsistent values, such as ”N/A” or ”unknown”.
- Standardize data formats, such as converting all currency values to USD.

4.2 Data Encoding

Transform categorical variables (e.g., operating system or display type) into numerical data for machine learning models using techniques like label encoding or one-hot encoding.

4.3 Handling Missing Data

- Remove irrelevant records.
- Fill missing values with the mean, mode, or predictions from imputation models.

4.4 Data Normalization

- Normalize numerical attributes (e.g., RAM, battery capacity, price) to ensure no attribute disproportionately influences the model.
- Use techniques like Min-Max Scaling or Standard Scaling.

4.5 Data Integration

- Combine data from multiple sources, ensuring consistency in format and content.
- Eliminate duplicate records or redundant fields.

4.6 Preprocessing Results

After preprocessing, the dataset is cleaned, consistent, and ready for analysis and machine learning model training. This ensures high-quality data, optimizing the performance and accuracy of the predictive model.

Part III
Exploratory Data Analysis (EDA)

1 Descriptive Statistical Analysis

Descriptive statistical analysis provides a comprehensive understanding of the dataset, focusing on the characteristics of key numerical attributes, their distributions, and relationships.

1.1 Data Overview

The dataset contains:

- **Number of records:** 833 smartphone samples.
- **Number of attributes:** 15 attributes including dimensions, weight, battery capacity, RAM, internal storage, screen specifications, CPU cores, NFC support, and price.

A summary of all attributes:

Table 2 . Summary of Dataset Attributes

#	Attribute	Description	Type
1	Brand	Smartphone brand name	Categorical
2	Dimension Length (mm)	Device length in millimeters	Numerical (float)
3	Dimension Width (mm)	Device width in millimeters	Numerical (float)
4	Dimension Thickness (mm)	Device thickness in millimeters	Numerical (float)
5	Weight (g)	Weight of the smartphone in grams	Numerical (int)
6	Battery Type	Type of battery (encoded)	Categorical
7	Battery Capacity (mAh)	Battery capacity in milliamp-hours	Numerical (int)
8	Screen Size (cm^2)	Area of the screen in square centimeters	Numerical (float)
9	Resolution Height (pixels)	Screen resolution height in pixels	Numerical (int)
10	Resolution Width (pixels)	Screen resolution width in pixels	Numerical (int)
11	CPU (cores)	Number of CPU cores	Numerical (int)
12	Internal Storage (GB)	Internal storage capacity in gigabytes	Numerical (int)
13	RAM (GB)	RAM capacity in gigabytes	Numerical (int)
14	NFC Support	Binary: Indicates NFC availability (0 or 1)	Numerical (int)
15	Price (USD)	Smartphone price in USD	Numerical (float)

1.2 Descriptive Statistics

Descriptive statistics for numerical attributes are as follows:

Table 3 . Descriptive Statistics of Key Numerical Attributes

#	Attribute	Mean	Median (50%)	Std	Min	Max
1	Dimension Length (mm)	173.47	163.10	35.71	104.90	326.40
2	Dimension Width (mm)	88.88	76.00	35.45	52.80	220.60
3	Dimension Thickness (mm)	8.18	8.00	1.37	5.10	20.10
4	Weight (g)	231.64	190.00	118.80	113.00	750.00
5	Battery Type	0.56	1.00	0.50	0.00	1.00
6	Battery Capacity (mAh)	5010.51	5000.00	1721.97	307.00	11200.00
7	Screen Size (cm²)	138.46	107.40	102.35	24.30	617.80
8	Resolution Height (pixels)	1094.32	1080.00	387.45	240.00	2880.00
9	Resolution Width (pixels)	2152.91	2400.00	545.11	320.00	3200.00
10	CPU (cores)	6.00	6.00	0.00	6.00	6.00
11	Internal Storage (GB)	130.28	128.00	96.57	1.00	512.00
12	RAM (GB)	9.23	6.00	39.24	1.00	512.00
13	NFC Support	0.53	1.00	0.50	0.00	1.00
14	Price (USD)	342.95	240.00	302.72	40.00	2099.00

1.3 Data Distribution Visualization

1.3.1 RAM and Internal Storage

- **RAM:** The distribution shows that most devices have **4GB to 8GB** of RAM, representing the highest proportion. Very few devices have RAM above **12GB**, which only appears in premium models.

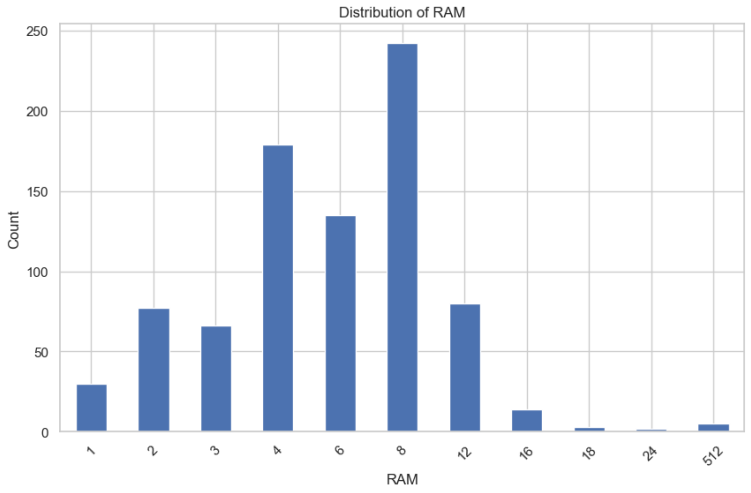

Figure 1 . Distribution of RAM (GB) in Smartphones

- **Internal Storage:** The most common storage sizes are **128GB** and **256GB**, as shown by the pie chart. Higher capacities like **512GB** are only found in premium devices.

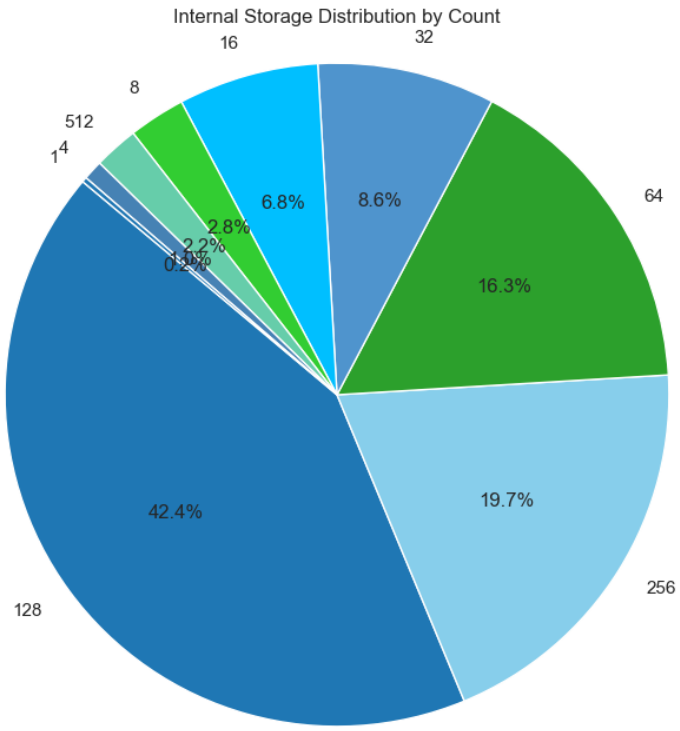

Figure 2 . Distribution of Internal Storage by Percentage

1.3.2 Price Range Analysis

Most smartphones are priced between **$100 and $300**, representing the largest share in the bar chart. Premium models priced above **$1000** account for only a small portion of the dataset.

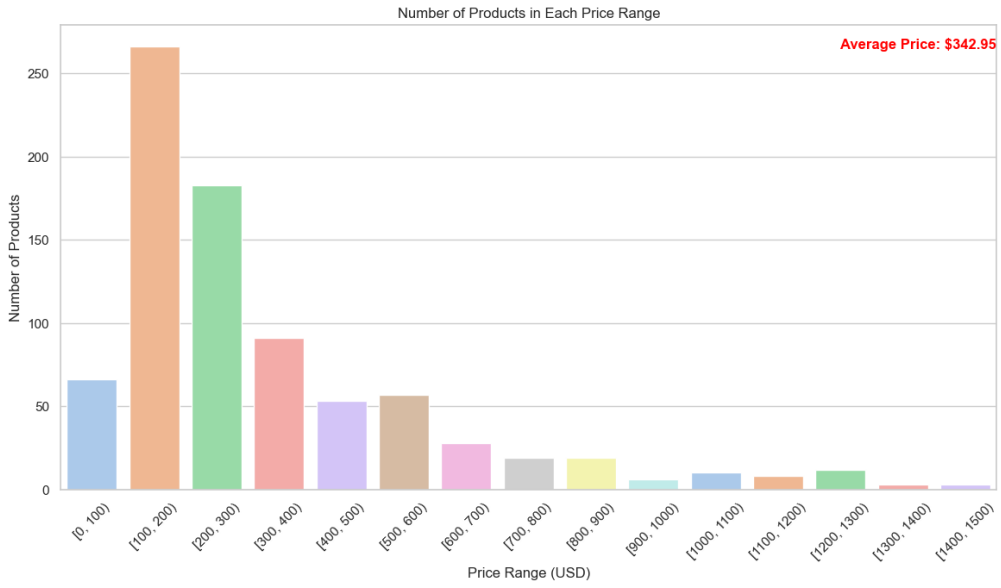

Figure 3 . Number of Products in Each Price Range

1.4 Correlation Analysis

The correlation analysis examines the relationships between key numerical attributes in the dataset. Two visualizations are presented to highlight these relationships: a heatmap showing the overall correlations and a bar chart illustrating the specific correlations between price and other features.

1.4.1 Heatmap of Correlations

The heatmap in Figure 4 provides an overview of the pairwise correlations among numerical attributes. Key observations include:

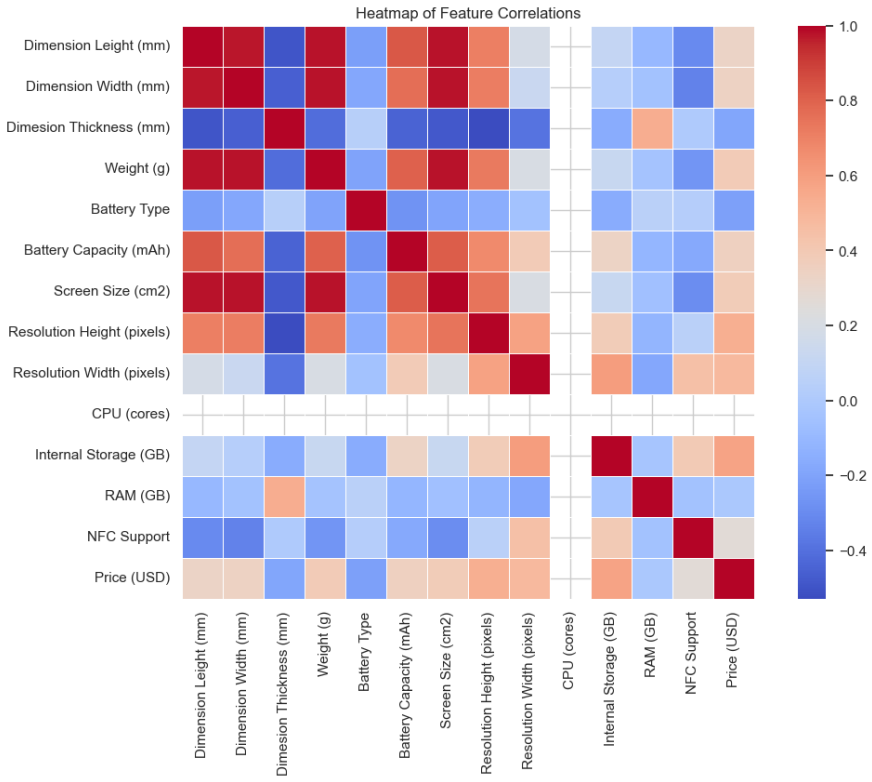

Figure 4 . Heatmap of Correlations between Numerical Attributes

- **Price and Internal Storage:** Price has a strong positive correlation with internal storage, with a correlation coefficient of **0.58**. This suggests that as internal storage increases, the price of the device also increases significantly. Premium devices with larger storage capacities (**256GB** and **512GB**) tend to have higher prices.
- **Price and Screen Resolution:** Screen resolution (both height and width) has a moderate positive correlation with price:
  - **Resolution Height:** Correlation coefficient of **0.53**.
  - **Resolution Width:** Correlation coefficient of **0.48**.

Devices with higher resolution screens generally have higher prices, reflecting their position in the premium segment of the market.

- **Price and Battery Capacity:** Battery capacity shows a weak positive correlation with price, with a coefficient of **0.35**. Devices with larger battery capacities often have slightly higher prices, although this relationship is less pronounced.
- **Screen Size and Battery Capacity:** Screen size has a strong positive correlation with battery capacity (**0.67**), indicating that larger screens tend to require larger batteries to support their energy consumption.
- **Weight and Dimensions:** The weight of devices has a moderate positive correlation with their physical dimensions (**length, width, and thickness**). This is logical since larger devices generally weigh more.
- **RAM and Price:** Interestingly, RAM shows almost no correlation with price, with a coefficient close to **-0.01**. This suggests that RAM capacity is not a significant determinant of device price in this dataset.

1.4.2 Specific Correlations with Price

Figure 5 provides a detailed visualization of the correlations between **Price** and other numerical attributes. The following attributes exhibit notable relationships:

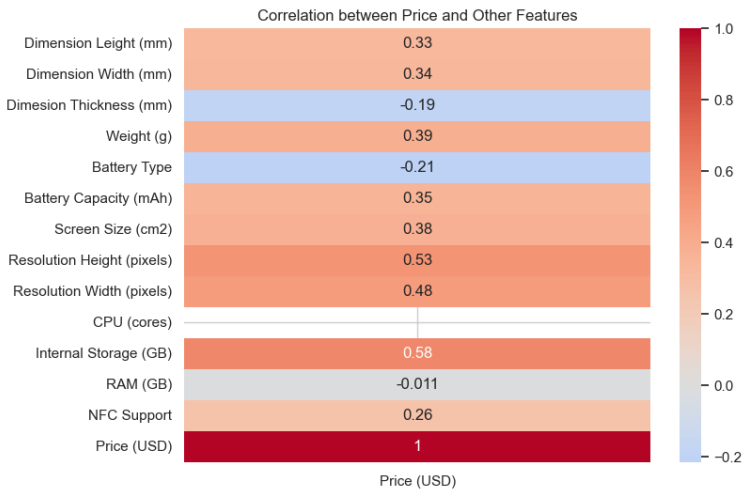

Figure 5 . Correlation between Price and Other Features

- **Internal Storage:** The strongest correlation with price (**0.58**).
- **Resolution Height and Width:** Moderate correlations (**0.53** and **0.48**, respectively).
- **Battery Capacity and Screen Size:** Weak correlations with price (**0.35** and **0.38**).
- **Weight and Dimensions:** Slight positive correlations, but less impactful on price.

- **NFC Support:** Weak correlation with price (**0.26**), indicating that devices with NFC support tend to have slightly higher prices.

The analysis reveals that internal storage and screen resolution are the most significant factors influencing the price of smartphones. Specifically, devices with higher internal storage capacities, such as 256GB or 512GB, tend to be priced significantly higher compared to those with lower storage options. Similarly, smartphones featuring high-resolution displays (in both width and height) are typically positioned in the premium segment and command higher prices.

On the other hand, attributes like RAM and battery capacity play a relatively minor role in determining the price. This is reflected in the low correlation coefficients between RAM, battery capacity, and price. Devices with higher RAM or larger batteries do not necessarily fall into the higher price range, suggesting that both consumers and manufacturers prioritize features such as internal storage and display quality when determining the value of a smartphone.

1.5 Relationships Between Variables

1.5.1 Price vs Internal Storage

The price increases significantly with larger internal storage capacities, especially beyond **256GB**. Devices with **512GB** of internal storage typically fall into the premium segment with very high prices.

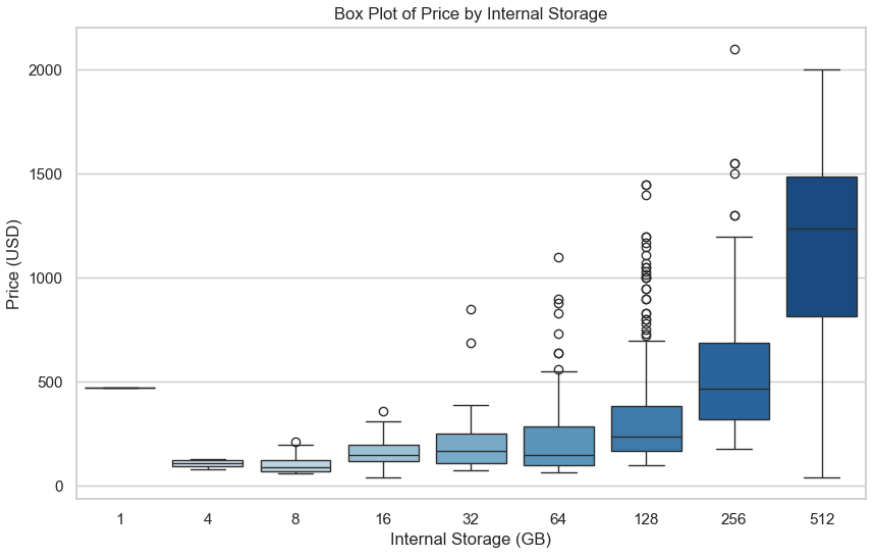

Figure 6 . Box Plot of Price by Internal Storage (GB)

1.5.2 Price vs Battery Capacity

There is a weak positive correlation between battery capacity and price, as shown by the slightly increasing trend line in the plot below.

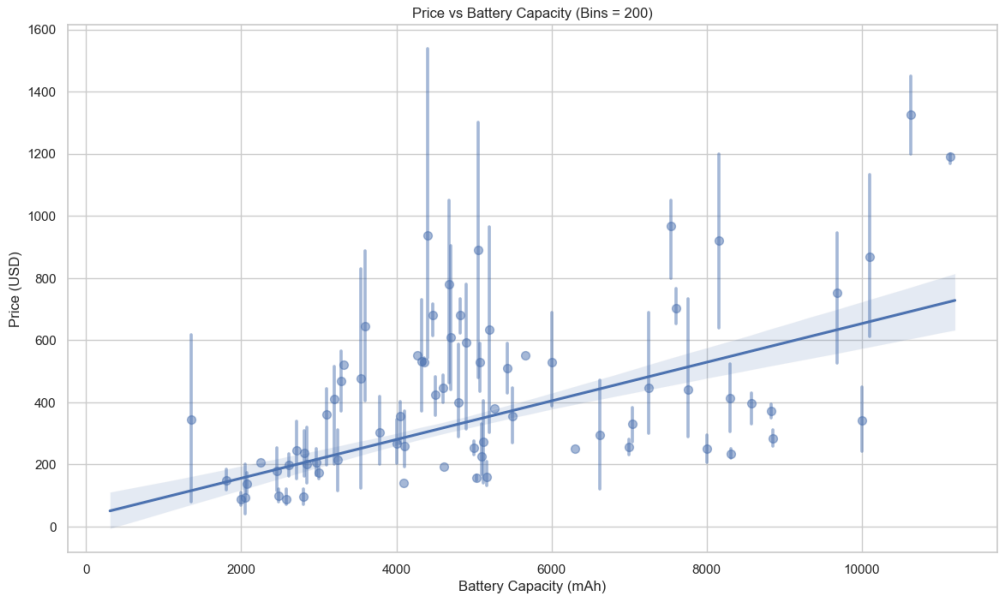

Figure 7 . Price vs Battery Capacity (mAh)

1.5.3 Price vs Screen Resolution

Higher resolution (both width and height) has a positive correlation with price. The 3D plot below shows that devices with higher screen resolution generally have higher prices.

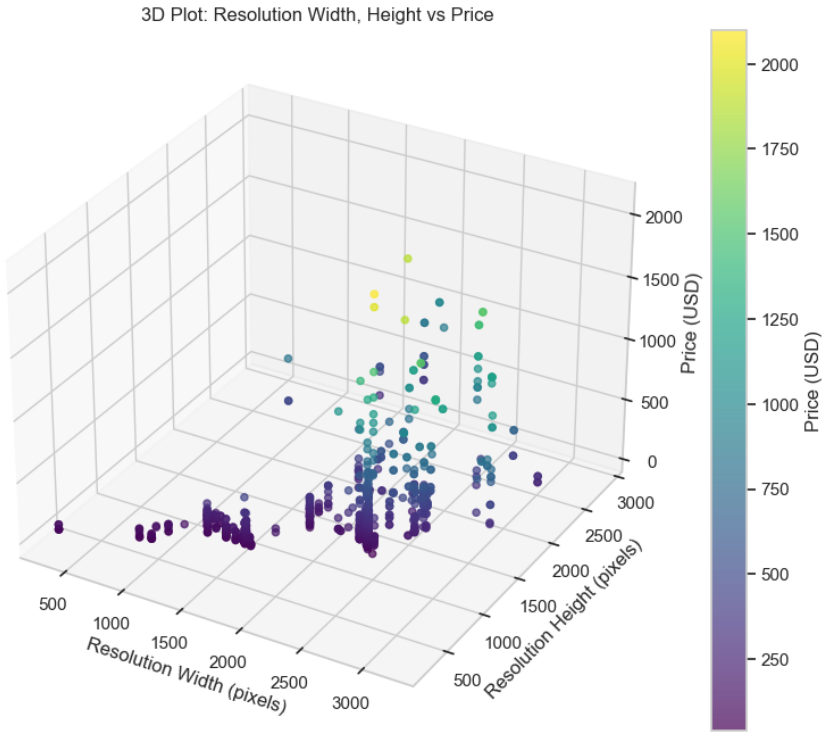

Figure 8 . 3D Plot: Resolution Width, Height vs Price

Conclusion

The EDA highlights the following key findings:

- Most smartphones have **4GB to 8GB of RAM** and **128GB to 256GB** of internal storage.
- **Price** has a strong correlation with **internal storage** (correlation coefficient **0.58**) and a moderate correlation with **screen resolution**.
- Premium devices with **512GB** of internal storage, larger battery capacities, and higher screen resolutions typically have very high prices.

Part IV
Model Development

1 Objective of Model Development

The primary goal of this section is to apply machine learning algorithms to predict smartphone prices based on their technical specifications. Each model is trained, evaluated, and compared based on its performance. The evaluation focuses on the following criteria:

- Prediction accuracy, measured by metrics like MSE, and R^2.
- Ability to handle complex data and outliers effectively.
- Stability and efficiency for deployment in real-world scenarios.

2 Models Used

2.1 Random Forest [1]

- **Description:** Random Forest is a machine learning model belonging to the **Ensemble Learning** category, which uses a collection of decision trees. Each tree is trained on a random subset of the data. The model combines the predictions from all the trees to make a final prediction. In the case of regression, the final prediction is the average of all the predictions from the trees. This process helps to reduce overfitting and improve accuracy by leveraging the strengths of multiple trees.
- **Ensemble Learning:** Ensemble Learning is a machine learning technique in which multiple models (in this case, decision trees) are used to solve the same problem. The main goal is to improve performance compared to using a single model. By combining the outputs of multiple models, Ensemble Learning helps to minimize errors, increasing both the accuracy and stability of the predictions.

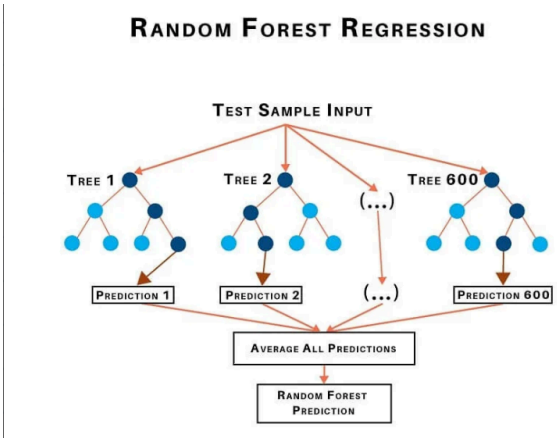

Figure 9 . Random Forest

- **Formula of Random Forest:** In a Random Forest model, the prediction of each tree in the forest is calculated using the following formula:

$$\hat{y}_{\text{RF}} = \frac{1}{N} \sum_{i=1}^{N} \hat{y}_i$$

Where:

  - $\hat{y}_{\text{RF}}$ is the final prediction of the Random Forest.
  - $\hat{y}_i$ is the prediction from the i-th tree out of N trees.
  - N is the total number of trees in the forest.

The final prediction is the average of all the predictions from the trees. This method helps reduce variance and avoid overfitting compared to using a single decision tree.

- **How Random Forest Works:**

  - As shown in the diagram, a test sample is passed through each decision tree in the forest (e.g., Tree 1, Tree 2, Tree 600).
  - Each tree makes its own prediction (e.g., Prediction 1, Prediction 2, Prediction 600).
  - The final Random Forest prediction is calculated by averaging all the predictions from the trees. This averaging process reduces the model's sensitivity to individual trees and helps prevent overfitting, making Random Forest more robust than a single decision tree.

- **Implementation:**
  - The Random Forest Regressor model is implemented using the `scikit-learn` library [4], with key parameters: `n_estimators=100` (number of trees) and `max_depth=10` (maximum depth of each tree).
  - `GridSearchCV` is used to fine-tune hyperparameters, optimizing parameters such as the number of trees (`n_estimators`), maximum depth of the trees (`max_depth`), and other parameters like `min_samples_split`, `min_samples_leaf`, and `max_features`.
  - The data is preprocessed by standardizing features using `StandardScaler`. Additionally, the data is split into training and testing sets using `train_test_split`.
  - The Random Forest algorithm trains models for each unique brand in the dataset, saving the best model for each brand after evaluation on both the training and test datasets.
  - Evaluation metrics such as Mean Squared Error (MSE) and R-squared (R^2) are used to assess model performance.
- **Significance of Tuning Hyperparameters:**
  - **n_estimators:** This is the number of trees in the forest. Tuning this parameter helps find the optimal number of trees to minimize both overfitting and underfitting. Using too few trees can make the model too simple (underfitting), while using too many trees can increase computational cost without significantly improving accuracy.
  - **max_depth:** The maximum depth of each tree. Tuning this parameter helps adjust the complexity of the tree. If the depth is too large, the tree may overfit the training data. If the depth is too small, the tree may not capture enough patterns in the data (underfitting).
  - **min_samples_split:** The minimum number of samples required to split an internal node. Tuning this parameter adjusts the level of granularity in the tree. Using too small a value can result in overfitting as the tree learns very detailed patterns.
  - **min_samples_leaf:** The minimum number of samples required to be at a leaf node. Tuning this parameter helps reduce the depth of the tree and prevents overfitting by avoiding the tree from learning overly specific patterns from the data.
  - **max_features:** The maximum number of features to consider when splitting a node. Using too many features can lead to overfitting, while using too few can reduce the model's accuracy.
- **Advantages:**
  - **Versatility:** Random Forest can handle both regression and classification tasks, making it a widely applicable model.

  - **Robustness:** Random Forest is resistant to overfitting, especially when dealing with noisy data. By averaging predictions from multiple trees, it reduces the model's sensitivity to individual anomalies.
  - **Handles Non-linear Data:** Unlike linear models, Random Forest can capture complex, non-linear relationships in the data.

- **Limitations:**
  - **Computational Complexity:** The model can be resource-intensive, especially for large datasets. The more trees used, the greater the computational and memory costs.
  - **Interpretability:** Random Forest, being an ensemble of many trees, lacks the transparency and ease of interpretation provided by simpler models like linear regression.

2.2 Gradient Boosting

- **Description:** Gradient Boosting is a powerful machine learning algorithm used for both classification and regression tasks. It builds an ensemble of decision trees in a sequential manner, where each tree is trained to correct the errors made by the previous tree. The model focuses on the residuals (errors) from prior trees and adjusts the predictions accordingly, making it a highly effective technique for improving the model's performance through iterative learning.
- **Ensemble Learning:** Gradient Boosting is a type of Ensemble Learning, specifically a boosting method, where multiple weak learners (typically decision trees) are combined to create a strong learner. Each subsequent tree in the model is trained to correct the mistakes made by the previous trees. This iterative approach enhances the overall performance by reducing both bias and variance in the model.

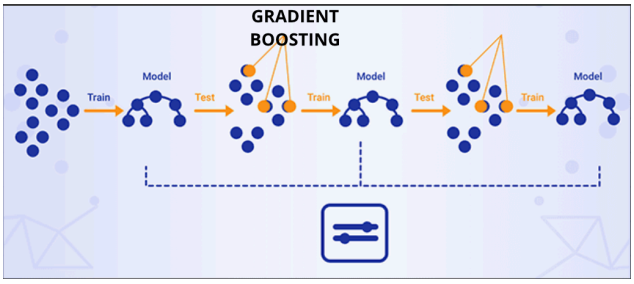

Figure 10 . Gradient Boosting

- **Formula of Gradient Boosting:** In Gradient Boosting, the prediction is updated at each stage by minimizing the residual error. The prediction of the model is a cumulative sum of all the weak learners (trees) trained in each iteration:

$$F_M(x) = F_{M-1}(x) + \eta \cdot h_M(x)$$

  Where:

  - $F_M(x)$ is the prediction of the model after the M-th tree.
  - $F_{M-1}(x)$ is the prediction of the model after the previous tree.
  - η is the learning rate, controlling the contribution of each tree to the model's final prediction.
  - $h_M(x)$ is the prediction from the M-th tree.

  The learning rate η shrinks the contribution of each tree, and this iterative process continues until the model converges.

- **Implementation:**

  - The Gradient Boosting Regressor model is implemented using the `scikit-learn` library, with key hyperparameters such as `n_estimators=100` (number of boosting stages), `learning_rate=0.1` (shrinkage factor), `max_depth=3` (maximum depth of each tree), and `subsample=1.0` (fraction of samples used for each tree).
  - `GridSearchCV` is employed to fine-tune the hyperparameters, optimizing the model's performance by selecting the best combination of values. The hyperparameters being tuned include:
    - `n_estimators`: Number of boosting stages (trees).
    - `learning_rate`: Shrinks the contribution of each tree.
    - `max_depth`: Maximum depth of the individual trees.
    - `subsample`: Fraction of samples used for fitting each tree.

  - The dataset is preprocessed by standardizing the features using `StandardScaler` to ensure that the model is not biased by features with larger scales.
  - The data is split into training and testing sets using `train_test_split`, with a 20% test set to evaluate the model's performance.
  - Cross-validation with 5 folds is used to ensure the model generalizes well and avoids overfitting.

- **Cross-Validation:** To avoid overfitting and ensure that the model generalizes well, 5-fold cross-validation is used during the hyperparameter tuning process. This helps in evaluating the model's performance on multiple subsets of the data, ensuring stability and robustness.

- **Advantages:**
  - **Powerful and Flexible:** Gradient Boosting is highly effective for capturing complex relationships between features and the target variable. It can handle non-linear data and high-dimensional feature spaces.
  - **High Performance:** By iteratively correcting errors, Gradient Boosting can significantly improve model accuracy, making it suitable for a wide range of regression and classification problems.
  - **Hyperparameter Tuning:** Gradient Boosting allows fine-tuning of multiple hyperparameters (such as the learning rate, number of trees, and tree depth), making it a versatile model that can be optimized for various tasks.

- **Limitations:**
  - **Overfitting:** If the hyperparameters are not tuned correctly, Gradient Boosting can easily overfit the training data. This is why careful tuning and cross-validation are necessary to avoid this issue.
  - **Computational Cost:** Gradient Boosting can be computationally expensive, especially when using a large number of trees. The sequential nature of the model (where each tree is built on the previous one) can also result in longer training times compared to parallel algorithms like Random Forest.
  - **Interpretability:** Like other ensemble methods, Gradient Boosting can be difficult to interpret due to the complex interactions between the multiple trees in the model.

2.3 XGBoost [2]

- **Description:** XGBoost (Extreme Gradient Boosting) is an advanced and highly effective boosting algorithm used for both classification and regression tasks. It builds an ensemble of decision trees in a sequential manner, where each tree is trained to correct the errors of the previous tree. XGBoost incorporates regularization techniques to prevent overfitting, making it highly efficient and scalable for large datasets. The model focuses on minimizing the residual errors from prior trees and adjusts predictions accordingly, providing an optimized solution for predictive tasks.

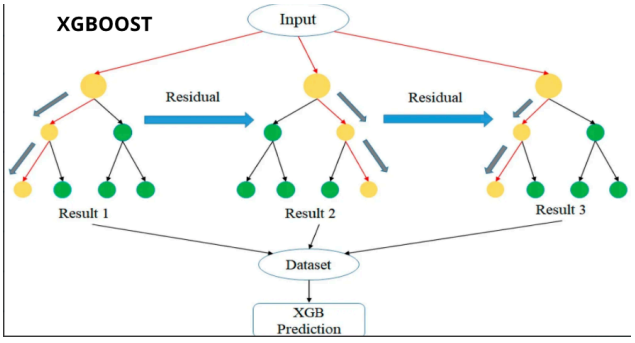

Figure 11 . XGBoost

- **Ensemble Learning:** XGBoost is a form of Ensemble Learning, specifically a boosting algorithm, where weak learners (decision trees) are combined to form a strong learner. Each subsequent tree is trained to correct the mistakes made by the previous trees, thus improving overall model accuracy. The iterative nature of XGBoost helps reduce both bias and variance in the model, making it robust and accurate.
- **Formula of XGBoost:** In XGBoost, the prediction is updated at each stage by focusing on the residuals (errors) from the previous trees. The model prediction is calculated as the cumulative sum of all the weak learners (trees) trained in each iteration:

$$F_M(x) = F_{M-1}(x) + \eta \cdot h_M(x)$$

  Where:

  - $F_M(x)$ is the prediction of the model after the M-th tree.
  - $F_{M-1}(x)$ is the prediction of the model after the previous tree.
  - η is the learning rate, controlling the contribution of each tree to the final prediction.
  - $h_M(x)$ is the prediction from the M-th tree.

  The learning rate η shrinks the contribution of each tree, and the iterative process continues until the model converges, resulting in the best prediction.

- **Implementation:**

  - The XGBoost Regressor is implemented using the `xgboost` library, with key hyperparameters such as `n_estimators=100` (number of boosting rounds), `learning_rate=0.1` (shrinkage factor), `max_depth=3` (maximum depth of individual trees), and `subsample=1.0` (fraction of samples used for each tree).
  - `GridSearchCV` is used to fine-tune the hyperparameters for the XGBoost model. Two grids are defined for optimization:
    - **Grid 1**: Focuses on the key parameters: `n_estimators`, `learning_rate`, `max_depth`, `min_child_weight`, and the objective function (`reg:squarederror`).

    - **Grid 2**: Focuses on fine-tuning parameters such as `subsample`, `colsample_bytree`, `gamma`, `reg_alpha`, and `reg_lambda`.

  Both grids are tuned using `GridSearchCV` with 10-fold cross-validation for Grid 1 and 5-fold cross-validation for Grid 2.

- The dataset is preprocessed by standardizing the features using `StandardScaler` to ensure that the model is not biased by features with larger scales.
- The data is split into training and testing sets using `train_test_split`, with a test size of 20% for model evaluation.

- **Cross-Validation:** To avoid overfitting and ensure that the model generalizes well, cross-validation is used during the hyperparameter tuning process. For Grid 1, 10-fold cross-validation is employed, and for Grid 2, 5-fold cross-validation is used to improve model accuracy and prevent overfitting.
- **Advantages:**
  - **High Performance:** XGBoost is known for its high performance, as it handles large datasets efficiently and performs well on complex data with non-linear relationships.
  - **Regularization:** XGBoost incorporates regularization techniques (`reg_alpha` and `reg_lambda`) to avoid overfitting, making it more stable and reliable.
  - **Handling Missing Values:** XGBoost automatically handles missing values and outliers in the data, making it robust for real-world datasets.
- **Limitations:**
  - **Hyperparameter Tuning:** Tuning the hyperparameters for XGBoost can be challenging, and it may require extensive computational resources for large datasets.
  - **Computationally Expensive:** XGBoost is more computationally expensive than simpler models like Decision Trees and Linear Regression, particularly when dealing with large datasets.
  - **Interpretability:** Like other ensemble methods, XGBoost models can be difficult to interpret due to the complexity of the multiple trees involved in the learning process.

2.4 Stacking

- **Description:** Stacking is an ensemble method that combines multiple base models (such as K-Nearest Neighbors and Decision Trees) to form a stronger meta-model that makes the final prediction. The base models are trained independently, and their predictions are then combined by a meta-model to improve accuracy. Stacking leverages the strengths of different algorithms by using them together, resulting in a more robust and accurate model.

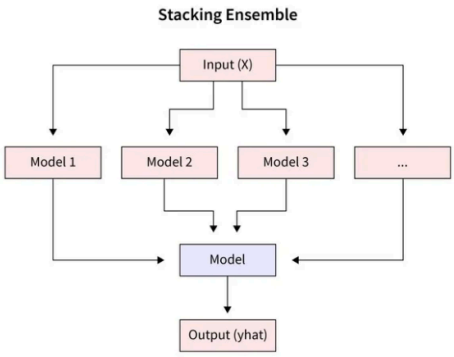

Figure 12 . Ensemble Learning (Stacking)

- **Ensemble Learning:** Stacking is a form of **Ensemble Learning**, where multiple base models are used to generate predictions, and a meta-model is used to combine those predictions into a final result. In this case, the base models include K-Nearest Neighbors (KNN) and Decision Trees (DT). The meta-model in the stacking approach is a Linear Regression model that combines the outputs of the base learners to make the final prediction.
- **Implementation:**
  - The base models used in this implementation are:
    - **K-Nearest Neighbors (KNN)**: The KNN model is tuned using `GridSearchCV`, focusing on parameters such as the number of neighbors (`n_neighbors`), the weight function (`weights`), and the distance metric (`p`).
    - **Decision Tree Regressor (DT)**: The DT model is also tuned using `GridSearchCV`, with key parameters such as the maximum depth of the tree (`max_depth`), the minimum samples required to split a node (`min_samples_split`), and the minimum number of samples required to be at a leaf node (`min_samples_leaf`).
  - **Stacking Regressor:** After tuning the KNN and DT models, the best versions of these models are used as base learners in the **Stacking Regressor**. A `LinearRegression` model is used as the meta-model (final estimator) that combines the predictions of the base learners (KNN and DT).
  - The dataset is first preprocessed by scaling the features using `StandardScaler`. The data is then split into training and testing sets using `train_test_split` (80% for training and 20% for testing).
  - `GridSearchCV` is used for hyperparameter optimization for both KNN and DT models, with 5-fold cross-validation for each grid search. The hyperparameter grids for both models are specified in the code.
  - The performance of the stacking model is evaluated on both training and test data using Mean Squared Error (MSE) and R-squared (R^2) metrics.

- **Advantages:**
  - **Improved Predictive Performance:** By combining the strengths of different base models, Stacking can enhance predictive performance, particularly for complex datasets.
  - **Flexibility:** Stacking can use a variety of base models, such as KNN, Decision Trees, or even other regressors, allowing for flexibility in model selection.
  - **Meta-Model Integration:** The use of a meta-model to combine predictions allows for optimal integration of the base learners' outputs, improving overall accuracy.
- **Limitations:**
  - **Increased Complexity:** Stacking increases model complexity by requiring multiple base models and a meta-model, leading to higher computational costs and longer training times.
  - **Overfitting Risk:** While stacking generally improves performance, there is still a risk of overfitting, particularly if the base models are overly complex or if hyperparameters are not properly tuned.

3 Model Development Process

The development of the models followed a systematic process to ensure that each model was properly trained, optimized, and evaluated. The following steps were carried out:

1. **Data Splitting:**
   - The dataset was randomly split into two parts: a training set comprising 80% of the data and a testing set comprising 20% of the data. This split ensured that the models were trained on a sufficient amount of data while retaining an independent set for performance evaluation.
   - K-fold cross-validation, with $k = 5$ and $k = 10$, was used during model performance evaluation. Cross-validation helps to evaluate the model's performance across multiple subsets of the data, reducing the risk of overfitting and providing a more reliable estimate of model accuracy.
2. **Data Preprocessing:**
   - The numerical features were standardized using `StandardScaler`. This step ensures that the features are on the same scale, which is particularly important for distance-based algorithms (such as KNN) and gradient-based models (such as XGBoost and Gradient Boosting). Standardization helps to prevent any feature from dominating the others due to differences in scale.

- Categorical variables, such as the operating system type and screen type, were encoded using One-Hot Encoding. This method converts the categorical variables into a format that can be fed into machine learning models by creating binary columns for each category.

3. **Hyperparameter Tuning:**

   - Grid Search was utilized to find the best hyperparameters for each model. The primary hyperparameters tuned included:
     - `n_estimators`: The number of decision trees or boosting rounds used by the model.
     - `max_depth`: The maximum depth of the trees in Random Forest and Gradient Boosting, and the maximum number of levels in the base models of the Stacking Regressor.
     - `learning_rate`: The rate at which the model's predictions are adjusted, particularly important for boosting algorithms such as Gradient Boosting and XGBoost.
   - Grid Search was performed using `GridSearchCV`, with cross-validation to ensure optimal hyperparameters were chosen, improving model accuracy.

4. **Training and Evaluation:**

   - Each model was trained using the training set, and their performance was evaluated on the testing set. This process ensures that the model is not overfitting the training data and can generalize well to unseen data.
   - The models were assessed using two main evaluation metrics:
     - **Mean Squared Error (MSE)**: This metric is used to assess the average squared difference between the predicted and actual values. A lower MSE indicates better predictive performance.
     - **R-squared (R^2)**: This metric measures the proportion of variance in the target variable that is explained by the model. A higher R^2 indicates a better fit of the model to the data.

4 Model Evaluation Results

After training and tuning the models, the performance of each model was evaluated using the test data. The following table presents the average performance of the models based on the Mean Squared Error (MSE) and R-squared (R^2) metrics.

From the evaluation results, we can conclude the following:

- XGBoost performed the best with the lowest MSE (17,634.4) and highest R^2 (0.79), indicating that it explained the most variance in the target variable and had the least error.

Table 4 . Average Performance of Prediction Models

Model	MSE	R^2
Random Forest Regressor	23,226.4	0.75
Gradient Boosting	23,357.1	0.76
XGBoost Regressor	17,634.4	0.79
Stacking Regressor	23,066.5	0.76

- The Random Forest Regressor, Gradient Boosting, and Stacking Regressor all showed competitive performance, with R^2 values ranging from 0.75 to 0.76. However, their MSE values were higher than XGBoost, suggesting that their predictions were less accurate in comparison.
- While Stacking performed similarly to Gradient Boosting and Random Forest, it did not outperform the other models. This indicates that, in this case, combining models did not yield a significant advantage over individual models.

The performance of the models shows that XGBoost, with its advanced boosting mechanism and built-in regularization, outperforms the other models for this particular dataset, making it the most suitable choice for prediction tasks in this scenario.

Conclusion

The model development phase successfully applied machine learning algorithms to predict smartphone prices based on detailed technical specifications. The primary goal of this project was to develop accurate and reliable models that could predict the prices of smartphones given their features. After thorough experimentation with multiple models, XGBoost Regressor emerged as the most suitable model due to its superior performance and robustness. The project demonstrated the importance of effective hyperparameter tuning and proper data preprocessing techniques in achieving high prediction accuracy.

Part V
Results

1 Model Performance

In this project, several machine learning models were developed and evaluated to predict smartphone prices based on their technical specifications. Below are the key findings from the model evaluation:

1.1 XGBoost Regressor

The XGBoost Regressor model delivered the best performance among all models tested. Its key evaluation metrics are as follows:

- **MSE (Mean Squared Error): 17,634.4**, indicating that the model minimizes the squared error between predicted and actual prices effectively.
- **R^2 (Coefficient of Determination): 0.79**, suggesting that 79% of the variance in smartphone prices is explained by the model.

This demonstrates the model's high accuracy and strong predictive power, with predictions closely aligning with actual values.

1.2 Stacking Regressor

The Stacking Regressor model achieved second-best performance, showing a strong but slightly less precise prediction compared to XGBoost:

- **MSE: 23,066.5**.
- **R^2: 0.76**.

While it provided a solid performance, the Stacking Regressor was slightly less effective than XGBoost in accurately predicting prices.

1.3 Random Forest Regressor and Gradient Boosting Regressor

These models exhibited slightly lower performance but were still robust in predicting smartphone prices. They showed limitations when dealing with non-linear and complex data patterns, as well as noisy data. Their performance metrics are as follows:

- **Random Forest Regressor:** MSE: **23,226.4**, R^2: **0.75**.
- **Gradient Boosting Regressor:** MSE: **23,357.1**, R^2: **0.76**.

2 Visualization Analysis

To further evaluate model performance, the following visualizations were used:

2.1 Scatter Plot

A scatter plot of actual prices versus predicted prices for each model revealed the following:

- The XGBoost Regressor closely followed the diagonal line, indicating strong predictive accuracy. The points are tightly clustered around the line, confirming its precise predictions.
- Other models, such as the Stacking Regressor, also performed well but were slightly less precise, as their points showed more deviation from the ideal diagonal line.

2.2 Residual Plot

The residual plot for the XGBoost Regressor showed that residuals (errors) were evenly distributed around zero. This indicates that the model did not suffer from visible overfitting or underfitting, confirming its robustness and generalizability.

3 Comparison Across Brands

The analysis of predicted prices across different smartphone brands yielded the following insights:

3.1 Premium Brands

Premium brands, such as Apple and Samsung, exhibited higher average prices. This trend was accurately captured by all models, demonstrating their ability to handle high-end devices and predict their pricing with a high degree of accuracy.

3.2 Mid-Range Brands

Mid-range brands, such as Xiaomi and Realme, were primarily concentrated in the lower to mid-price segments. The XGBoost model effectively captured and predicted the value of these devices, with minimal deviation from actual prices, demonstrating its ability to handle a wide range of pricing tiers.

Part VI
Conclusion and Future Directions

1 Conclusion

- The project successfully achieved its goal of developing machine learning models to predict smartphone prices based on detailed technical specifications. The XGBoost Regressor model was identified as the optimal choice, as it delivered superior performance with an MSE of **17,634.4** and R^2 of **0.79**.
- Exploratory Data Analysis (EDA) revealed valuable insights into the factors that influence smartphone prices, such as RAM, internal storage, and brand reputation. These insights proved crucial for building effective prediction models.
- This project highlights the importance of machine learning in solving real-world problems in commerce and technology, providing valuable tools that can benefit both manufacturers and consumers by enabling more informed pricing decisions.

2 Project Limitations

- **Data Limitations:**
  - The dataset was primarily collected from online sources, which may not always be up-to-date or comprehensive.
  - Key factors such as production costs and manufacturers' pricing strategies were not included in the dataset, limiting the scope of the analysis.
- **Model Limitations:**
  - Hyperparameter tuning, especially for complex models like XGBoost, was computationally expensive and time-consuming.
  - Non-technical factors such as brand perception, market trends, and consumer demand are challenging to incorporate into the model effectively.

2.1 Future Directions

- **Data Expansion:**
  - Collect more diverse and comprehensive data from various sources to improve the dataset's richness, especially for underrepresented brands.
  - Implement real-time data updates and continuous model retraining to ensure that price predictions remain accurate over time.
- **Model Enhancements:**
  - Investigate deep learning techniques, such as Neural Networks, to capture more complex relationships among features and improve prediction accuracy.
  - Explore advanced ensemble methods and model stacking techniques to further boost the performance of the existing models.
- **Integration of Additional Features:**
  - Incorporate non-technical factors such as brand reputation, user reviews, and market demand trends to improve model predictions.
  - Utilize social media data or customer feedback to incorporate more consumer sentiment and preferences into the price prediction models.
- **Practical Applications:**
  - Develop a user-friendly web-based application or tool that allows consumers to input smartphone specifications and receive real-time price predictions.
  - Integrate the model into e-commerce platforms or retail systems to assist businesses in making competitive pricing decisions.

References

[1] GeeksforGeeks. Random forest regression in python, 2024. Accessed: 2024-12-15.

[2] GeeksforGeeks. Xgboost, 2024. Accessed: 2024-06-17.

[3] GSMArena Team. GSMArena: Mobile Phone Reviews, Specifications, and News, 2024. Accessed: 2024-12-15.

[4] Scikit-learn Developers. *Scikit-learn: Machine Learning in Python*, 2024. Accessed: 2024-06-17.

[5] Scrapy Developers. *Scrapy Documentation*, 2024. Accessed: 2024-12-15.