LAPTOP PRICE PREDICTION

M.Sc. Statistics

CONTENT

Chapter 1

INTRODUCTION

In the modern technological landscape, laptops have become pivotal tools for personal, academic, and professional use. Their ever-expanding range of features, specifications, and brands, coupled with frequent technological advancements, makes the task of accurately predicting their market prices increasingly complex. Traditional methods of price estimation, often relying on manual analysis or basic statistical techniques, struggle to address the intricate variables influencing laptop prices. To navigate this complexity, machine learning (ML) offers a sophisticated solution by harnessing advanced algorithms to analyze extensive datasets and identify patterns that traditional approaches might miss. This introduction explores how ML enhances the accuracy of laptop price predictions and the processes involved in utilizing ML for this purpose.

Machine learning, a subset of artificial intelligence, equips systems with the ability to learn from data and make predictions without being explicitly programmed for each specific task. In the realm of laptop price prediction, ML can analyze diverse data sources, including detailed laptop specifications (e.g., processor type, RAM size, storage capacity), brand and model information, release dates, historical pricing data, and broader market conditions. This multifaceted data collection is the first critical step in developing an effective ML model. Data collection involves aggregating information from various sources, such as online retail platforms, manufacturer specifications, and historical sales records. This comprehensive dataset serves as the foundation for training ML algorithms to predict prices accurately.

The preprocessing stage of data is crucial for ensuring that it is suitable for ML analysis. Raw data often contains missing values, errors, and inconsistencies that need to be addressed to avoid skewed results. Data cleaning involves identifying and rectifying these issues, ensuring the integrity of the dataset. In addition, normalization and scaling are applied to adjust data features to a common scale, which helps improve the performance and convergence of many ML algorithms. Feature selection is another important preprocessing step, where the most relevant variables impacting laptop prices are identified. This process reduces the dimensionality of the dataset, which can enhance the model's efficiency and reduce overfitting.

Once preprocessing is complete, selecting the appropriate ML model for price prediction is the next step. There are several types of algorithms that can be employed, each with distinct advantages. **Linear Regression** is a straightforward algorithm that establishes a relationship between the dependent variable (laptop price) and one or more independent variables (features like RAM, processor type). It is useful for understanding linear relationships and making preliminary predictions. **Decision Trees** provide a visual representation of decisions and their possible outcomes, breaking down data into segments based on feature values. This model is intuitive and helps in understanding how various features influence price. **Random Forests**, an ensemble learning method, aggregate predictions from multiple decision trees to enhance accuracy and robustness. This approach mitigates the risk of overfitting by averaging the predictions of several trees. **Gradient Boosting** builds models sequentially, where each model corrects errors made by the previous ones, leading to improved performance and accuracy. **Neural Networks** simulate the workings of the human brain, capturing complex and non-linear relationships in the data. They are particularly effective for handling large and intricate datasets with multiple features.

Training these models involves using historical data to enable the algorithms to learn patterns and relationships between laptop features and their prices. The data is typically split into a training set and a test set. The training set is used to fit the model and teach it to recognize patterns, while the test set is used to evaluate the model's performance and generalizability. During this phase, various metrics are used to assess the model's accuracy, including Mean Squared Error (MSE), which measures the average squared difference between predicted and actual prices, and R-squared, which indicates the proportion of variance in laptop prices that is explained by the model.

After training and evaluating the model, it is ready for deployment. Deployment involves integrating the model into a functional application or system that allows users to input various laptop specifications and receive predicted prices. This application can be a web-based tool, a software program, or an integrated feature within a larger e-commerce platform. Effective deployment ensures that users can easily access and utilize the model's predictions to make informed purchasing or pricing decisions.

Continuous monitoring and updating of the model are essential to maintain its accuracy and relevance. As new laptop models are introduced, technological advancements occur, and market conditions change, the model must be retrained with updated data to reflect these developments. This ongoing process ensures

that the model remains accurate and useful over time, providing valuable insights into the fluctuating laptop market.

In summary, machine learning offers a sophisticated and data-driven approach to predicting laptop prices. By leveraging advanced algorithms and extensive datasets, ML enhances the accuracy of price predictions and provides deeper insights into market dynamics.

This approach not only aids consumers in making informed decisions but also assists sellers and manufacturers in pricing strategies and market analysis. The ability to handle complex data and adapt to changing conditions makes ML a powerful tool in the ever-evolving world of laptop technology. Furthermore, the integration of machine learning into laptop price prediction extends beyond mere forecasting; it also enables the development of predictive analytics tools that can dynamically adjust to emerging market trends.

For instance, ML models can be programmed to incorporate real-time data such as current sales promotions, supply chain disruptions, or shifts in consumer preferences, providing more accurate and timely price predictions. Additionally, advanced techniques such as natural language processing (NLP) can be utilized to analyze online reviews and sentiment, offering insights into how public perception and feedback might impact laptop prices. This real-time adaptability and comprehensive analysis enhance the model's ability to deliver actionable insights, making it a valuable asset for stakeholders seeking to navigate the complexities of the laptop market with greater precision and agility.

1.1 MOTIVATION OF THE STUDY

The motivation for studying laptop price prediction using machine learning stems from the growing complexity and competitiveness of the technology market, where consumers and businesses alike face challenges in making informed decisions amidst a plethora of options. As technological advancements and market dynamics rapidly evolve, traditional methods of price estimation become increasingly inadequate. Machine learning offers a powerful alternative by leveraging data-driven insights to predict prices with higher accuracy and relevance. This approach not only benefits consumers by providing them with better pricing information and value assessments but also supports sellers and manufacturers in setting competitive prices and understanding market trends. By improving the precision and adaptability of price predictions, this study aims to enhance decision-making processes, optimize pricing strategies, and ultimately contribute to a more transparent and efficient technology marketplace.

1.2 OBJECTIVE OF THE STUDY

- To build a regression model to predict laptop prices with high accuracy based on various features and specifications**.**
- To compare different fitted models to identify and select the most effective one for accurate price predictions**.**

1.3 SOURCE OF THE STUDY

The dataset used for this project is a laptop price prediction dataset, which is a secondary dataset taken from the website Kaggle. The dataset contains 13 columns. The columns include both features and specifications of laptops. The objective of the dataset is to help people predict the price of a laptop based on its features and specifications, enabling them to make informed purchasing decisions.

1.4 REVIEW OF THE STUDY

1. Prof. B. Prajna, A. Lekha Sri, A. Sahiti, A. Bhagyasri Teja, A. Arzoo (2023):
   The study conducted by Prof. B. Prajna and colleagues in 2023 introduces a laptop price prediction system leveraging supervised machine learning techniques. Specifically, the research employs multiple linear regression, achieving a prediction precision of 81%. The model utilizes multiple independent variables, such as the laptop's model, RAM, ROM (HDD/SSD), GPU, CPU, IPS Display, and Touch Screen, to predict a single dependent variable—price. The actual and predicted prices are compared to assess the precision of the model. This paper highlights the effectiveness of multiple linear regression in predicting laptop prices, with the authors suggesting that this method can be further refined with additional factors or by integrating other machine learning techniques such as Random Forest and Support Vector Machines (SVM).

2. Astri Dahlia Siburian, Daniel Ryan Hamonangan Sitompul, Stiven Hamonangan Sinurat, Andreas Situmorang, Ruben, Dennis Jusuf Ziegel, Evta Indra (2022):
   In their 2022 study, Siburian et al. explore the impact of the COVID-19 pandemic on the demand for laptops suitable for WFH setups. The study notes that with 32.37% of large and medium-sized enterprises in East Java opting for partial WFH, there is a significant need for laptops that enhance productivity. To assist consumers in avoiding overpriced purchases, the authors developed a machine learning model designed to predict laptop prices based on specific configurations. The study covers the entire process from data acquisition and cleaning to feature engineering and exploratory data analysis. The model achieved its highest accuracy with the XGBoost algorithm, reaching 92.77%. This high accuracy indicates that the model is reliable in predicting laptop prices, providing consumers with a useful tool for making informed purchasing decisions.

3. Prof. Vaishali Surjuse, Sankalp Lohakare, Aayush Barapatre, Abhishek Chapke (2022):

In their 2022 paper, Prof. Vaishali Surjuse and colleagues developed a laptop price prediction system using supervised machine learning, specifically focusing on multiple linear regression. The model, which achieved a prediction precision of 81%, uses several independent variables—such as the laptop's model, RAM, ROM (HDD/SSD), GPU, CPU, IPS Display, and Touch Screen—to predict the price, which is treated as the dependent variable. The authors compare the actual and predicted prices to evaluate the model's accuracy. This study underscores the effectiveness of multiple linear regression in creating a price prediction model that considers various hardware specifications, providing a useful tool for consumers.

4. Mohammed Ali Shaik, Medicherla Varshith, Sanka SriVyshnavi, Nagamalla Sanjana, Rama Sujith (2022):
   In 2022, Mohammed Ali Shaik, Medicherla Varshith, Sanka SriVyshnavi, Nagamalla Sanjana, and Rama Sujith proposed a novel approach to laptop price prediction, addressing the challenge of selecting the right model amidst a vast array of specifications and brands. Their research highlights the growing importance of laptops in daily life and the complexities involved in pricing due to diverse features and market options. By leveraging machine learning (ML) techniques, the study aims to enhance prediction accuracy through a sophisticated identification process involving various features and ML models. The researchers tested multiple machine learning models—including Decision Trees, Multiple Linear Regression, K-Nearest Neighbors (KNN), and Random Forest—by applying different feature combinations. Their approach seeks to determine the most accurate model for predicting laptop prices, contributing to more informed decision-making in the increasingly complex laptop market.

5. Prof. Parmeshwar Manegopale, Komal Nerpagar, Sanal Sawant, Madhuri Shinde, Kunal Chandarkar (2023)
   In 2023, Prof. Parmeshwar Manegopale, Komal Nerpagar, Sanal Sawant, Madhuri Shinde, and Kunal Chandarkar undertook a comprehensive laptop price prediction project, structured in three distinct phases. The initial phase involved developing a predictive model based on various factors such as screen size and CPU speed. The second phase focused on testing this model with real data collected from multiple websites to evaluate its performance. The final phase aimed to present and discuss the

results, refining the model to enhance its accuracy. Their project integrates a series of algorithms designed to predict laptop prices based on features and specifications, with results aligning closely with predictions from established methods like neural networks and Support Vector Machines (SVM). This work highlights the effectiveness of their model and its consistency with other advanced prediction techniques.

6. C. Janiesch, P. Zschech, and K. Heinrich (2021)
In 2021, intelligent systems that incorporate artificial intelligence capabilities often rely on machine learning, which enables systems to learn from problem-specific data to automate analytical model building. Deep learning, a subset of machine learning based on artificial neural networks, often outperforms traditional models in various applications. This article summarizes the fundamentals of machine learning and deep learning, distinguishing key concepts, explaining the process of automated model building, and discussing challenges in implementing intelligent systems in electronic markets, including technological and human-machine interaction issues.

1.5 CHAPTERIZATION

The project report is organized into four comprehensive chapters, each designed to guide the reader through the intricate process of laptop price prediction using advanced machine learning techniques.
Chapter One sets the stage by introducing the concept of laptop price prediction, highlighting the significance of accurate price forecasting in today's technology-driven market. This chapter also outlines the motivation behind the study, clearly states the objectives, and provides a detailed literature review, ensuring that the reader understands the context and relevance of the research. Additionally, the chapter discusses the sources of data used, emphasizing the credibility and reliability of the information collected.
Chapter Two delves into the world of machine learning, offering an in-depth exploration of the regression models and algorithms employed in the study. This chapter covers a range of sophisticated machine learning techniques, including Linear Regression, Ridge Regression, Random Forest Regression, Support Vector Regression, and AdaBoost Regression. Each algorithm is meticulously explained, providing insights into why

these methods were selected and how they contribute to the accuracy and effectiveness of the price prediction model.

Chapter Three is dedicated to data analysis, showcasing how machine learning techniques were applied to real-world data to achieve accurate laptop price predictions. This chapter presents the step-by-step process of model fitting, evaluation, and refinement, demonstrating the practical application of the algorithms discussed in the previous chapter. Visual aids and detailed explanations make the data analysis process both accessible and engaging, ensuring that the reader can follow along with the methodology.

Chapter Four concludes the report by summarizing the key findings of the study, offering conclusions and reference of my study.

CHAPTER 2

METHODOLOGY

2.1 Machine learning

Machine learning (ML) is a subset of artificial intelligence (AI) that enables software applications to improve their prediction accuracy over time without explicit programming. By analyzing historical data, machine learning algorithms can forecast future outcomes and uncover significant patterns. This capability is crucial in the field of data science, where statistical methods and algorithms are utilized to classify, predict, and derive actionable insights from data. These insights are instrumental in guiding business decisions and driving growth. As the volume of big data continues to increase, the demand for skilled data scientists will rise. These professionals will be essential in pinpointing pertinent business questions and identifying the data needed to address them.

2.1.1 MACHINE LEARNING METHODS

Machine learning algorithms are essential for addressing a variety of business challenges, such as regression, classification, forecasting, clustering, and associations. These algorithms are typically categorized into four main types based on their learning methods:

1. Supervised Machine Learning

Supervised learning involves the use of labelled data, where the model learns to map inputs to outputs based on examples provided during training. In essence, supervised learning models use this training to predict outcomes by applying the knowledge they have acquired. The model's output is essentially the inference of a function derived from labelled training data.

Steps Involved in Supervised Learning:

- Determine the type of training dataset:
  Identify the type of data needed for the model.

- Collect labeled training data:
  Gather the data that will be used to train the model.

- Split the dataset:
  Divide the data into training, testing, and validation sets.

- Identify input features:
  Choose features that provide sufficient information for accurate predictions.

- Select an appropriate algorithm:
  Choose an algorithm like support vector machines or decision trees that suits the model.

- Execute the algorithm:
  Run the algorithm on the training data, using validation sets if necessary.

- Evaluate model accuracy:
  Test the model with the test set to ensure it predicts correctly, indicating model accuracy.

Supervised learning can be further divided into two types:

I. Classification

Classification involves categorizing data into distinct classes, which can be either structured or unstructured. The process begins with predicting the class of given data points, aiming to determine which category the new data belongs to. This type of predictive modeling approximates the mapping function from input variables to discrete output variables.

II. Regression

Regression analysis is a statistical technique used to model the relationship between dependent (target) and independent (predictor) variables. It helps to understand how the dependent variable changes in response to variations in the independent variables, predicting continuous values like temperature, salary, or price.

Supervised learning algorithms are particularly effective for tasks such as:

Binary classification : Dividing data into two distinct categories.
Multi-class classification : Choosing between more than two possible outcomes.
Regression modelling : Predicting continuous values.
Ensembling : Combining predictions from multiple models for improved accuracy.

2. Unsupervised Machine Learning

Unsupervised learning is more challenging and complex because it involves models that learn from data without labeled outputs or explicit instructions.

There are two primary approaches in unsupervised learning:

First Approach:

- Reward-Based Learning

This approach teaches the model through a reward system rather than

explicit categorizations, making it suitable for decision-making tasks where actions are rewarded or penalized. It mirrors real-world scenarios where actions are either encouraged or discouraged without a predefined method for completing tasks. This approach is particularly useful when data is scarce or when the learning process is time-intensive.

Second Approach:

- Clustering

In clustering, the model identifies patterns and similarities within the data by grouping similar data points into clusters without any supervision or reward system. The assumption is that these patterns align with intuitive classifications. For example, online shopping platforms often use clustering algorithms to recommend products based on customer behavior. Clustering techniques evaluate the quality of these structures using a cost function, which is minimized to optimize the parameters that define the hidden structures in the data. For robust and reliable results, these extracted structures should be consistently identifiable across different data samples.

Unsupervised learning algorithms are particularly effective for:

- **Clustering:** Grouping similar data points.
- **Anomaly detection:** Identifying unusual data points.
- **Association mining:** Discovering relationships between data items.
- **Dimensionality reduction:** Reducing the number of variables in a dataset.

Unsupervised learning can be further classified into two specific types:

Clustering

Clustering is a foundational unsupervised learning technique that focuses on grouping unlabelled data points into clusters based on their inherent similarities. These similarities can be related to various features such as shape, size, colour, or behaviour, depending on the nature of the data. The primary objective of clustering is to organize data into distinct groups where the members of each group, or cluster, are more similar to each other than to those in other clusters. This technique is widely utilized across various fields, including customer segmentation, where it helps

businesses identify different customer groups based on purchasing behaviour, and image analysis, where it assists in recognizing patterns or objects within images. Additionally, clustering plays a crucial role in anomaly detection, where it helps to identify outliers that do not fit into any established clusters, signalling potential irregularities or points of interest. By simplifying and organizing large and complex datasets into manageable clusters, this technique aids in more efficient data processing, analysis, and decision-making.

Association

Association rule learning is a powerful data mining technique used to discover interesting relationships or associations between variables in large datasets. Unlike clustering, which groups data based on overall similarity, association rule learning focuses on identifying patterns or dependencies between specific data items. This technique is particularly effective in applications like market basket analysis, where it reveals which products are frequently purchased together, allowing retailers to optimize product placements or marketing strategies. Similarly, in web usage mining, association rules can uncover patterns in user behavior, helping to tailor content recommendations or enhance user experience. The strength of association rule learning lies in its ability to identify and quantify these relationships, providing actionable insights that can drive strategic decisions in areas such as inventory management, cross-selling, and personalized marketing.

3. Semi-Supervised Machine Learning

Semi-supervised learning (SSL) combines elements of both supervised and unsupervised learning. It is particularly useful when labelled data is scarce, expensive, or time-consuming to obtain. SSL uses a small amount of labelled data to guide the learning process and a larger amount of unlabelled data to enhance the model's performance.
SSL is especially beneficial in scenarios such as:

- **Machine translation:** Training algorithms to translate languages with limited labelled data.
- **Fraud detection:** Identifying fraud cases with only a few positive examples.

- **Labelling data:** Using algorithms trained on small datasets to automatically label larger dataset

4. Reinforcement Learning

Reinforcement Learning (RL) is a distinct machine learning paradigm where an agent learns to make decisions by interacting with its environment. Unlike traditional supervised learning, which relies on a set of labelled input-output pairs, RL focuses on learning from the consequences of actions taken in various situations. The agent receives feedback in the form of rewards or penalties, guiding it to learn optimal behaviours through trial and error. The core challenge in reinforcement learning lies in balancing exploration—trying new actions to discover their effects—and exploitation—using known actions that yield the highest reward.

Reinforcement learning is typically categorized into two main methods:

1. Positive Reinforcement Learning:

This method involves increasing the likelihood of a desired behavour by providing positive rewards. When an agent receives a positive reward for a particular action, it reinforces that behaviour, making it more likely to be repeated in similar situations. This method is effective in strengthening the behaviour of the agent and fostering the development of successful strategies.

2. Negative Reinforcement Learning:

Contrary to positive reinforcement, negative reinforcement focuses on increasing the likelihood of a behaviour by removing an unfavourable outcome. In this scenario, the agent learns to avoid certain actions that lead to negative results, thereby reinforcing behaviours that help it steer clear of adverse situations.

Reinforcement learning has wide-ranging applications across various fields:

Robotics:

In robotics, RL is used to teach robots how to perform tasks in the physical world, such as navigating environments, manipulating objects, or interacting with humans.

Video Game AI:

RL has been employed to develop AI agents that can learn to play video games, often surpassing human performance by continuously learning and adapting to different game scenarios.

Resource Management:

Enterprises leverage RL to optimize resource allocation, helping them achieve defined goals efficiently by learning the best strategies for utilizing finite resources.

By focusing on learning through interaction and feedback, reinforcement learning opens up possibilities for creating intelligent systems that can adapt to complex, dynamic environments.

2.2 Library function used for analysis

2.2.1 NumPy

NumPy, short for Numerical Python, is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, making it easier to handle and perform operations on extensive datasets. NumPy also includes a wide array of high-level mathematical functions to perform operations on these arrays, which makes it a cornerstone for numerical computations in Python. Its ability to efficiently manipulate and process data makes it a critical tool in data science, machine learning, and other technical fields that require the handling of large datasets.

2.2.2 Pandas

Pandas is a versatile Python library that facilitates data manipulation and analysis, especially when working with structured data like tables and time series. It introduces data structures such as DataFrames and Series, which allow for the easy handling and analysis of numerical tables. Pandas is widely used for tasks such as cleaning data, merging datasets, and performing complex data transformations. Its functionality is essential for preparing and managing data in a format that is suitable for analysis, making it an indispensable tool in the data scientist's toolkit.

2.2.3 Seaborn

Seaborn is a powerful Python visualization library built on top of Matplotlib, designed to create attractive and informative statistical graphics with minimal code. It offers a high-level interface for drawing various types of visualizations, including heatmaps, violin plots, and pair plots, among others. Seaborn simplifies the process of creating complex plots by handling many aspects of the plotting process, such as color palettes and axis labels, automatically. This makes it a preferred choice for generating publication-quality visualizations that convey insights clearly and effectively.

2.2.4 Matplotlib

Matplotlib is a comprehensive data visualization library for Python, known for its ability to create static, animated, and interactive plots. It is widely used for generating line graphs, bar charts, scatter plots, and other forms of visual data representations. Matplotlib is highly customizable, allowing users to fine-tune every aspect of a plot, from colors and line styles to fonts and legends. This flexibility, coupled with its tight integration with NumPy, makes Matplotlib a go-to tool for visualizing data across various domains, from academic research to business analytics.

2.3 ALGORITHMS

2.3.1 Linear Regression

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable and one or more independent variables. The primary goal of linear regression is to predict the value of the dependent variable based on the values of the independent variables. When there is only one independent variable, the model is called simple linear regression; when there are multiple independent variables, it is known as multiple linear regression.

The model assumes a linear relationship between the variables, which is expressed by the following equation for simple linear regression:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

Where:

- y is the dependent variable.
- x is the independent variable.
- β_0 is the intercept (the value of y when x=0).
- β_1 is the slope of the line (the change in y for a one-unit change in x).
- ϵ is the error term, accounting for the difference between the observed and predicted values.

For multiple linear regression, the equation expands to:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n +$$

Where x_1, x_2, …, x_n are the independent variables.

Assumptions of Linear Regression

Linear regression relies on several key assumptions to ensure the validity and accuracy of its predictions:

1. **Linearity**: The relationship between the dependent and independent variables must be linear. This means the change in the dependent variable is proportional to the change in the independent variable(s).
2. **Independence**: The observations must be independent of each other. This assumption ensures that the error terms ϵ\epsilonϵ are not correlated.
3. **Homoscedasticity**: The variance of the error terms should be constant across all levels of the independent variable(s). In other words, the spread of residuals (differences between observed and predicted values) should be consistent across the range of the independent variable(s).
4. **Normality**: The error terms should be normally distributed. This assumption is particularly important for hypothesis testing, as it affects the confidence intervals and significance tests.
5. **No Multicollinearity (for multiple regression)**: In multiple linear regression, the independent variables should not be highly correlated with each other. High multicollinearity can inflate the variance of the coefficient estimates and make the model unstable.

Equations and Interpretation

The core of linear regression lies in estimating the coefficients β0, β1,..., βn using the method of **least squares**. The objective is to minimize the sum of the squared differences between the observed and predicted values:

$$\text{Minimize} \sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

Where y_i is the observed value, and $\hat{y_i}$ is the predicted value.

2.3.2 Ridge Regression

Ridge regression is an extension of linear regression that addresses some of the limitations of the traditional linear regression model, particularly when dealing with multicollinearity among the independent variables. Multicollinearity occurs when two or more predictor variables are highly correlated, leading to unstable estimates of regression coefficients. Ridge regression mitigates this issue by introducing a penalty term to the regression equation, which shrinks the coefficients towards zero.

The core idea behind ridge regression is to minimize the sum of squared residuals while also constraining the size of the coefficients. This regularization technique helps to reduce the model's complexity and prevent overfitting, leading to more reliable predictions.

Assumptions of Ridge Regression

Ridge regression shares many of the same assumptions as ordinary linear regression but introduces a regularization term that relaxes some of these assumptions:

1. Linearity: The relationship between the dependent and independent variables is assumed to be linear.
2. Independence: The observations are assumed to be independent of each other.
3. Homoscedasticity: The variance of the errors is constant across all levels of the independent variables.
4. Normality: The errors are normally distributed, particularly important for hypothesis testing.
5. Multicollinearity: Unlike ordinary linear regression, ridge regression can handle multicollinearity. It does not assume that the independent variables are uncorrelated. In fact, ridge regression is specifically designed to address multicollinearity by penalizing large coefficients.

Ridge Regression Equations

In ordinary linear regression, the objective is to minimize the sum of squared errors:

$$\text{Minimize} \sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

In ridge regression, a penalty term is added to the objective function:

$$\text{Minimize} \left(\sum_{i=1}^{n} (y_i - \hat{y_i})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right)$$

Where:

- y_i is the observed value of the dependent variable.
- $\hat{y_i}$ is the predicted value of the dependent variable.
- β_j are the coefficients of the independent variables.
- λ is the regularization parameter (also called the penalty term), controlling the amount of shrinkage applied to the coefficients. A larger λ leads to more significant shrinkage.

The ridge regression equation can be expressed as:

$$\hat{\beta} = (X^T X + \lambda I)^{-1} X^T y$$

Where:

- X is the matrix of independent variables.
- I is the identity matrix.
- λ is the regularization parameter.

2.3.3 Random Forest

Definition**:** Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive performance. It can be used for both classification and regression tasks. The idea is to create a "forest" of decision trees, where each tree is trained on a random subset of the data and features.

How It Works:

1. **Data Sampling:** Randomly select subsets of the training data with replacement (bootstrap sampling) to create multiple datasets.
2. **Feature Selection:** For each tree, randomly select a subset of features (features are chosen at each node for splitting) to ensure diversity among the trees.
3. **Tree Construction:** Build decision trees on each of these subsets independently.
4. **Aggregation:**
    - **Classification:** Use a majority voting mechanism to decide the final class label (the class that the majority of trees vote for).
    - **Regression:** Average the predictions from all the trees to get the final prediction.

Random Forest Regression

Definition: Random Forest Regression is a specific application of the Random Forest algorithm for regression tasks. Instead of classifying data into categories, it predicts continuous values.

How It Works**:**

1. **Training Trees:** Multiple decision trees are built using random subsets of the data and features.
2. **Predicting Values:** Each tree in the forest makes a prediction for a given input. The final prediction is the average of all individual tree predictions.

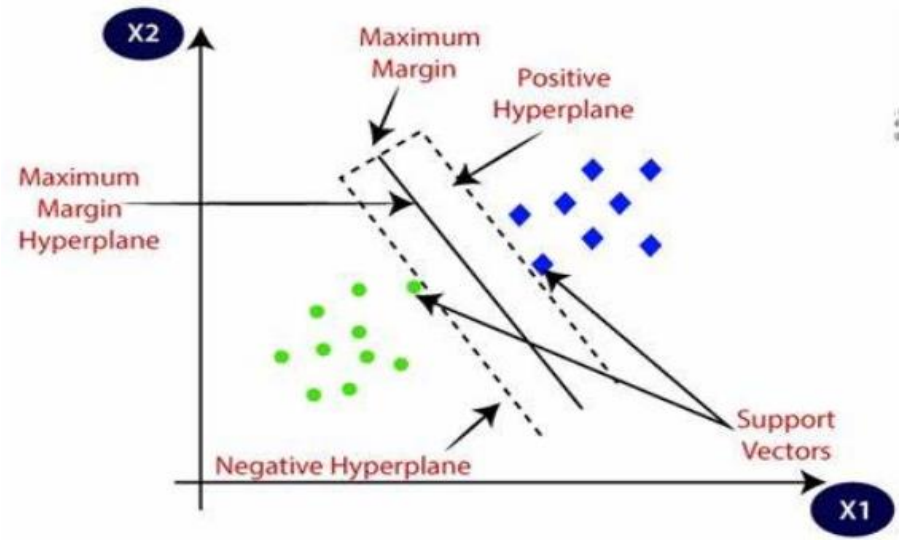
Advantages

1. **Robustness to Overfitting:** By averaging predictions from multiple trees, Random Forest reduces the risk of overfitting, especially compared to individual decision trees.
2. **Handles Large Datasets Well:** It can efficiently handle large datasets with high-dimensional features.

3. **Feature Importance:** It provides a measure of feature importance, which helps in understanding the impact of each feature.
4. **Versatility:** Suitable for both classification and regression tasks.
5. **Resilience to Noise:** Generally performs well even with noisy data.

2.3.4 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a widely-used supervised learning algorithm applied to both classification and regression tasks, though it is most commonly associated with classification problems. The primary objective of the SVM algorithm is to determine the optimal hyperplane that separates an n-dimensional space into distinct classes, allowing new data points to be categorized accurately. This optimal boundary is known as a hyperplane. SVM selects specific data points, known as support vectors, which are crucial in defining the hyperplane, thus the name "Support Vector Machine."



Types of SVM:

- Linear SVM: This type is utilized when the data is linearly separable, meaning that a single straight line can effectively divide the dataset into two classes. When this is the case, a Linear SVM classifier is applied.

- Non-linear SVM: When the data cannot be separated by a straight line, a Non-linear SVM classifier is used. This approach is

suitable for datasets that require a more complex decision boundary.

Advantages of SVM:

- High-Dimensional Efficiency: SVM performs well in high-dimensional spaces.
- Memory Efficiency: It leverages a subset of the training data, known as support vectors, which helps in conserving memory. Additionally, SVM allows for the use of various kernel functions, and custom kernels can be specified if needed.

Disadvantages of SVM:

- Overfitting Concerns: When the number of features significantly exceeds the number of samples, it is important to carefully choose kernel functions and regularization parameters to avoid overfitting.

- Lack of Direct Probability Estimates: SVMs do not inherently provide probability estimates. These can be obtained through a computationally intensive process involving cross-validation.

2.3.5 AdaBoost Regressor

AdaBoost Regressor is an ensemble learning technique used in supervised learning for regression tasks. AdaBoost, which stands for Adaptive Boosting, combines multiple weak learners to create a robust model. The main idea behind AdaBoost is to improve the predictive performance by focusing on the errors made by previous models in the sequence.

How AdaBoost Regressor Works:

AdaBoost Regressor builds a sequence of regression models, each of which attempts to correct the errors of its predecessor. The process involves:

- Initial Model: The algorithm starts by fitting a base regression model (often a simple one like a decision tree with limited depth) to the training data.
- Error Adjustment: After each model is trained, the weights of the training samples are adjusted so that the subsequent model focuses more on the samples that were previously mis predicted.
- Model Combination: Each model in the sequence is weighted according to its accuracy, and the final prediction is obtained by combining the predictions of all the models, with more emphasis on the more accurate ones.

Advantages of AdaBoost Regressor:

- Improved Accuracy: By sequentially focusing on the errors of previous models, AdaBoost can improve the accuracy of predictions.
- Robustness to Overfitting: AdaBoost is generally resistant to overfitting, especially when using a simple base learner and proper tuning.

Disadvantages of AdaBoost Regressor:

- Sensitivity to Noisy Data: AdaBoost can be sensitive to noisy data and outliers because it gives more weight to misclassified samples in each iteration.
- Computational Cost: Training multiple models sequentially can be computationally expensive and time-consuming, especially with large datasets.

Chapter 3

Data Analysis

3.1 Description of the dataset

The dataset used for this project is focused on laptop price prediction and is a secondary dataset obtained from the website Kaggle. The dataset contains 1,273 rows and 13 columns. The columns represent various features of laptops, including RAM, Weight, Price, TouchScreen, IPS, PPI, HDD, and SSD.

Attribute Information

- **Company:** The brand or manufacturer of the laptop (e.g., Apple, HP).
- **TypeName:** The category or type of the laptop, such as Ultrabook or Notebook
- **Ram:** The amount of RAM in the laptop, measured in gigabytes (GB).
- **Weight:** The weight of the laptop, measured in kilograms (kg).
- **Price:** The price of the laptop, often normalized or log-transformed, as in your dataset.
- **TouchScreen:** A binary feature indicating whether the laptop has a touchscreen (1 for Yes, 0 for No).

- **Ips:** A binary feature indicating whether the laptop screen uses IPS technology (1 for Yes, 0 for No).

- **Ppi (Pixels Per Inch):** The pixel density of the laptop screen, influencing the display sharpness and clarity.

- **Cpu_brand:** The brand of the laptop's CPU, such as Intel Core i5 or Intel Core i7.

- **HDD:** The storage capacity of the laptop's hard disk drive (HDD), measured in gigabytes (GB).

- **SSD:** The storage capacity of the laptop's solid-state drive (SSD), measured in gigabytes (GB).

- **Gpu_brand:** The brand of the laptop's graphics processing unit (GPU), such as Intel or AMD.

- **Os:** The operating system installed on the laptop, such as Mac or Others.

These features provide a comprehensive set of attributes that can be used to predict the price of a laptop based on its specifications.
The head of the dataset is given below, the features are Company, TypeName, Ram, Weight, Price, TouchScreen, Ips, Ppi, Cpu_brand, HDD, SSD, Gpu_brand, Os.

	Company	TypeName	Ram	Weight	Price	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	Apple	Ultrabook	8	1.37	11.175755	0	1	226.983005	Intel Core i5	0	128	Intel	Mac
1	Apple	Ultrabook	8	1.34	10.776777	0	0	127.677940	Intel Core i5	0	0	Intel	Mac
2	HP	Notebook	8	1.86	10.329931	0	0	141.211998	Intel Core i5	0	256	Intel	Others
3	Apple	Ultrabook	16	1.83	11.814476	0	1	220.534624	Intel Core i7	0	512	AMD	Mac
4	Apple	Ultrabook	8	1.37	11.473101	0	1	226.983005	Intel Core i5	0	256	Intel	Mac

The description of the dataset which includes the count is given below:

	Ram	Weight	Price	TouchScreen	Ips	Ppi	HDD	SSD
count	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000
mean	8.447761	2.041100	10.828218	0.146897	0.279654	146.950812	413.715632	186.252946
std	5.098771	0.669241	0.619565	0.354142	0.449006	42.926775	518.054486	186.531571
min	2.000000	0.690000	9.134616	0.000000	0.000000	90.583402	0.000000	0.000000
25%	4.000000	1.500000	10.387379	0.000000	0.000000	127.335675	0.000000	0.000000
50%	8.000000	2.040000	10.872255	0.000000	0.000000	141.211998	0.000000	256.000000
75%	8.000000	2.310000	11.287447	0.000000	1.000000	157.350512	1000.000000	256.000000
max	64.000000	4.700000	12.691441	1.000000	1.000000	352.465147	2000.000000	1024.000000

Prices in the dataset are log-transformed, with values ranging from 9.13 to 12.69. The mean price is approximately 10.83, and the median price is 10.87, indicating a fairly symmetrical distribution. The standard deviation is 0.62, showing a relatively low spread in price values.

3.2 Charts and diagrams

3.2.1 Pie chart

A pie chart is a circular graphical representation used to illustrate the proportions of different categories within a dataset. Each slice of the pie corresponds to a category, with the size of the slice proportional to its share of the total. The entire chart represents 100% of the data, making it an effective way to visualize the relative contributions of different components. Pie charts are commonly used to compare parts of a whole, showing how each category contributes to the overall dataset.

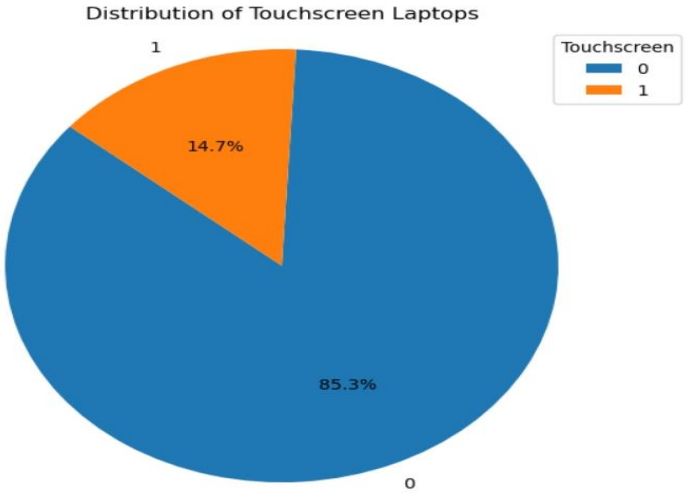


Figure1: Pie chart showing the percentage of laptops with and without touchscreens.

Percentage:

- 85.3% - Touchscreen
- 14.7% - Without Touchscreen

3.2.2 Distribution Plot

A distribution plot, often created using tools like Seaborn's `displot`, visually represents the distribution of a dataset by combining a histogram and a Kernel Density Estimate (KDE). The histogram shows the frequency of data points within specified intervals, while the KDE provides a smooth curve that estimates the probability density function, offering a continuous view of the data's distribution. This combined visualization helps in understanding the shape, spread, and central tendencies of the data.

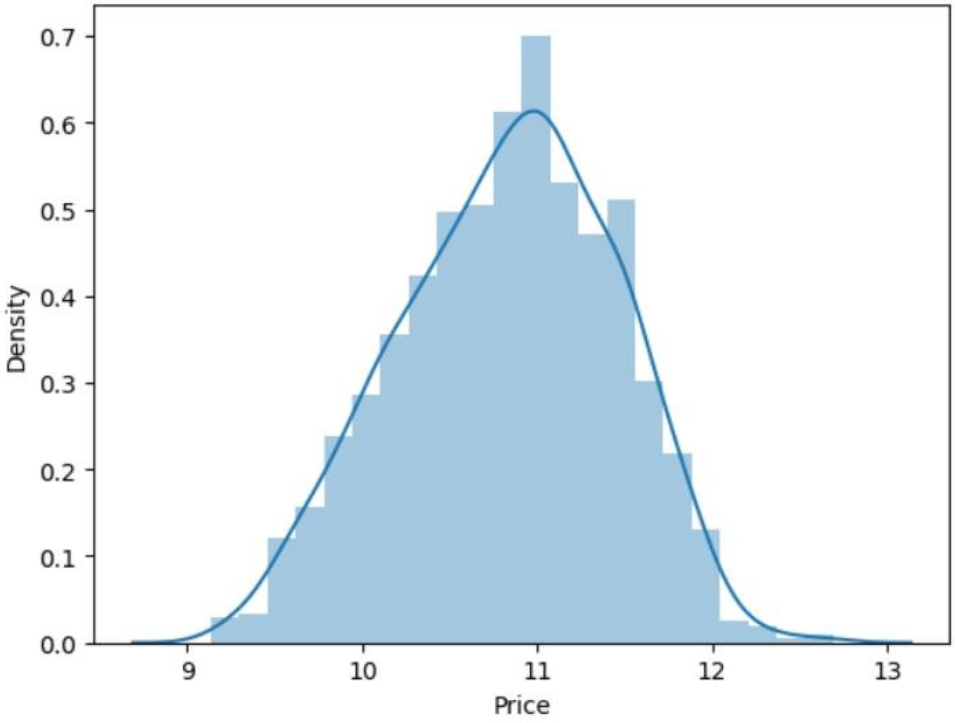


Figure 2: This distribution plot shows the distribution of price of laptops.

The distribution plot of `Price` shows that most laptops are priced around `10.5 lakhs`, with moderate variability and a slight right skew, indicating a few higher-priced outliers. The KDE curve further confirms these trends, highlighting a concentration of prices in the mid-range with fewer expensive laptops.

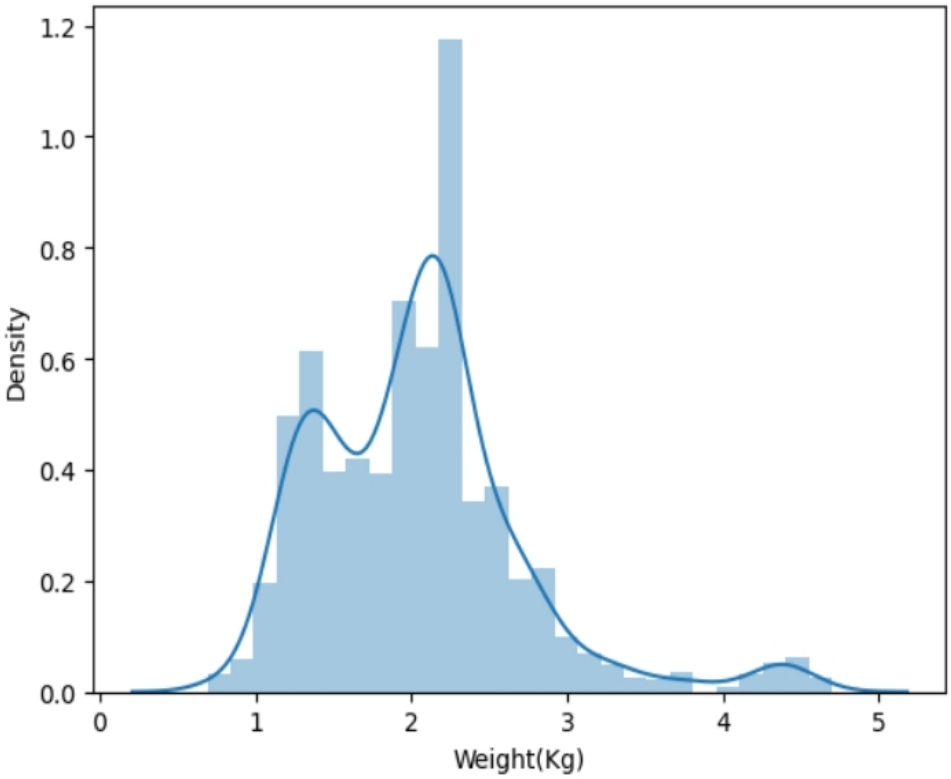


Figure 3: This plot shows the distribution of different weight of the laptops.

The distribution plot of `Weight(Kg)` reveals that most laptops in the dataset have a weight concentrated around a specific range, likely indicating a common standard for laptop weight. The spread of the distribution suggests some variability, with a slight skew pointing to a few heavier or lighter models. This pattern indicates that while most laptops are designed to be portable, there are some variations in weight, reflecting different form factors or use cases.

3.2.3 Scatter Plot

A scatter plot is a type of data visualization that displays the relationship between two numerical variables. Each point on the plot represents an observation in the dataset, with its position determined by the values of the two variables on the x and y axes. Scatter plots are useful for identifying correlations, trends, and patterns, such as clustering of data points or outliers, and can help in understanding how changes in one variable might relate to changes in another.

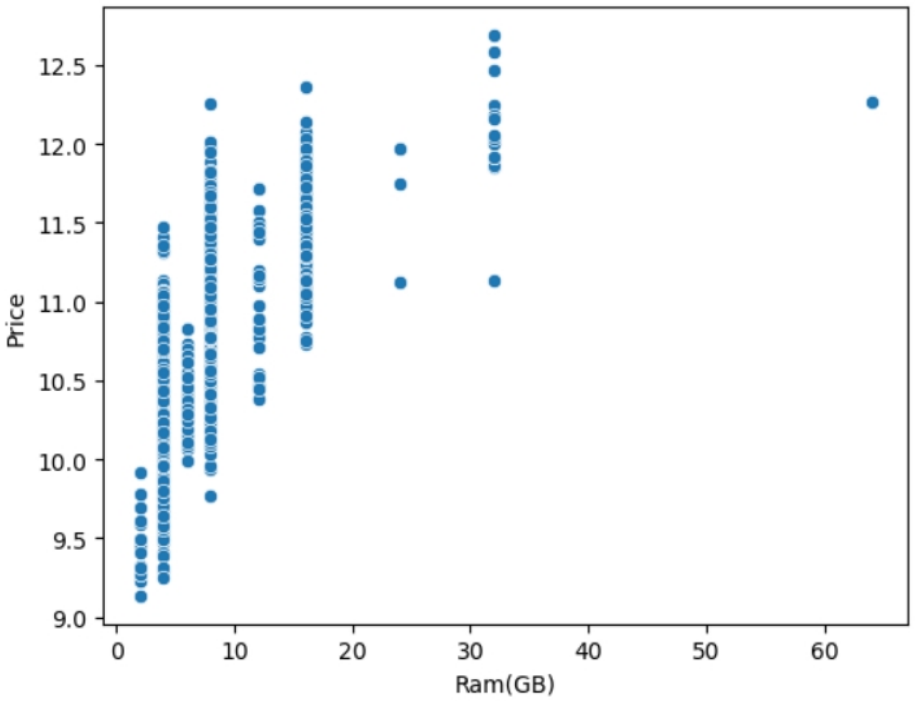


Figure 4: This plot shows the relationship between RAM and laptop price

The scatter plot of `Ram(GB)` versus `Price` shows that there is a general trend where laptops with higher RAM tend to have higher prices, indicating a positive correlation between these two variables. Most laptops cluster around lower RAM values (between 4GB and 16GB), with prices generally increasing as RAM increases. A few outliers are present, particularly at higher RAM levels, where the price varies more significantly, suggesting that factors other than RAM also influence the price. This pattern suggests that while RAM is an important factor in determining laptop prices, other components and features likely contribute to the pricing as well.

3.2.4 Histogram

A histogram is a type of data visualization that displays the distribution of a single numerical variable by dividing the data into bins (intervals) and showing the frequency of data points within each bin. The x-axis represents the range of values, while the y-axis shows the count or frequency of data points within each bin. Histograms are useful for understanding the shape of the data distribution, such as whether it's skewed, normal, or contains outliers, and for identifying patterns like the spread or concentration of values in the dataset.

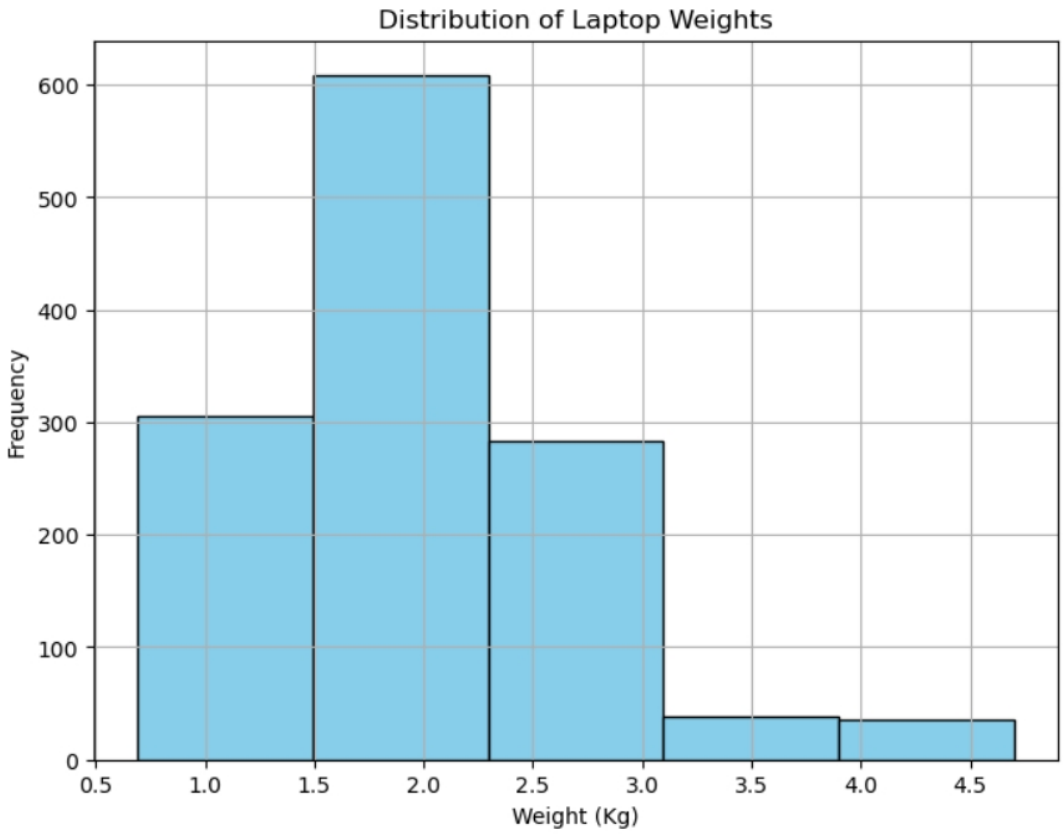


Figure 5: This plot describes the laptop weights distribution.

The histogram of `Weight (Kg)` shows how laptop weights are distributed across different intervals. With 5 bins, the plot reveals whether most laptops fall within a specific weight range, and the frequency of laptops within each range. If the distribution is concentrated in a particular bin, it suggests that most laptops have a similar weight. The presence of multiple peaks could indicate different categories of laptops, such as ultra books versus heavier gaming laptops. This visualization helps in understanding the common weight categories for laptops in the dataset.

3.2.5 Bar Chart

A bar chart is a graphical representation of data where individual bars are used to compare different categories or groups. Each bar's length or height corresponds to the value or frequency of the category it represents, making it easy to visualize and compare the magnitude of different items. The x-axis typically shows the categories, while the y-axis displays the numerical values or frequencies. Bar charts are particularly useful for

displaying and comparing discrete data sets, allowing for clear and straightforward visual analysis of differences among categories.

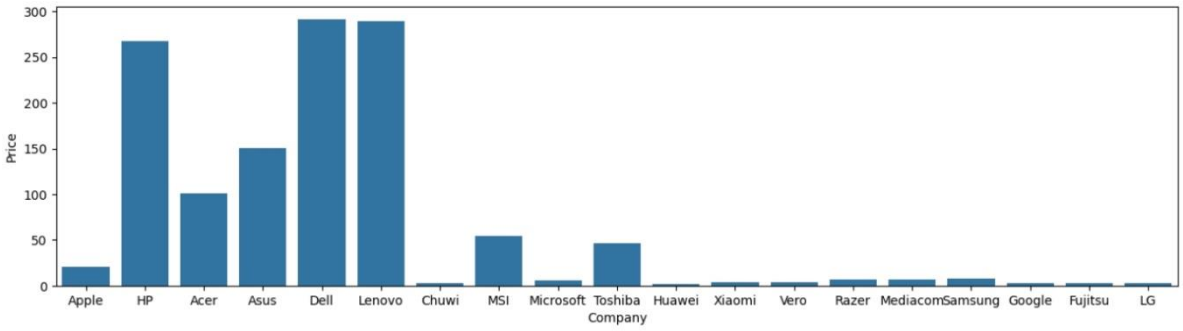


Figure 6: This plot describes the Distribution of Laptop Brands in the Dataset

This bar chart visualizes the frequency of laptops from different companies in the dataset. Each bar represents the number of laptops available from a specific company, with the height of the bar indicating the count. This visualization helps identify which companies have the most laptop listings and reveals the distribution of laptop brands in the dataset. By analysing this chart, one can quickly assess brand representation and potentially infer market share or popularity trends among the listed companies.

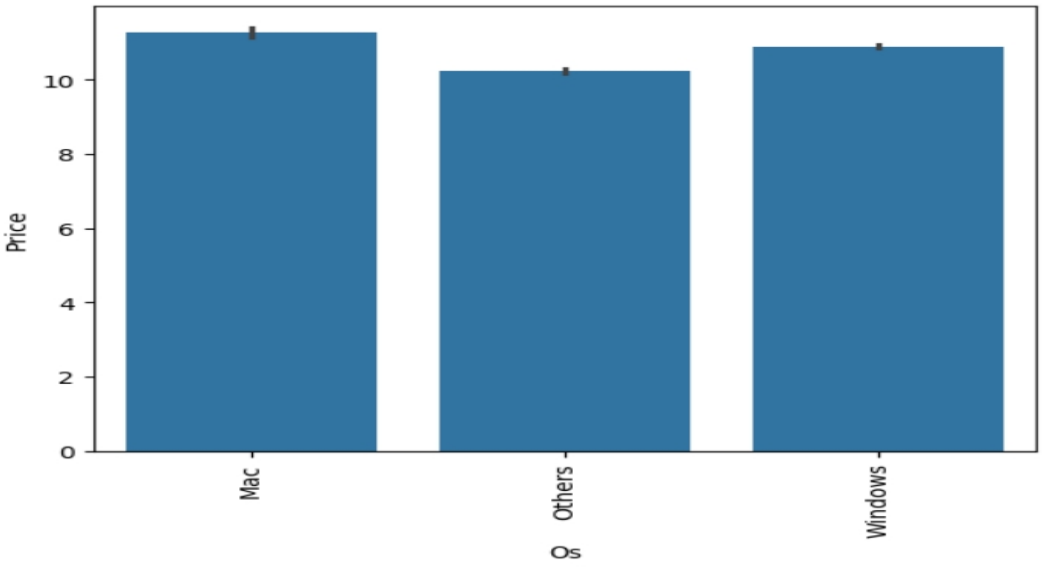
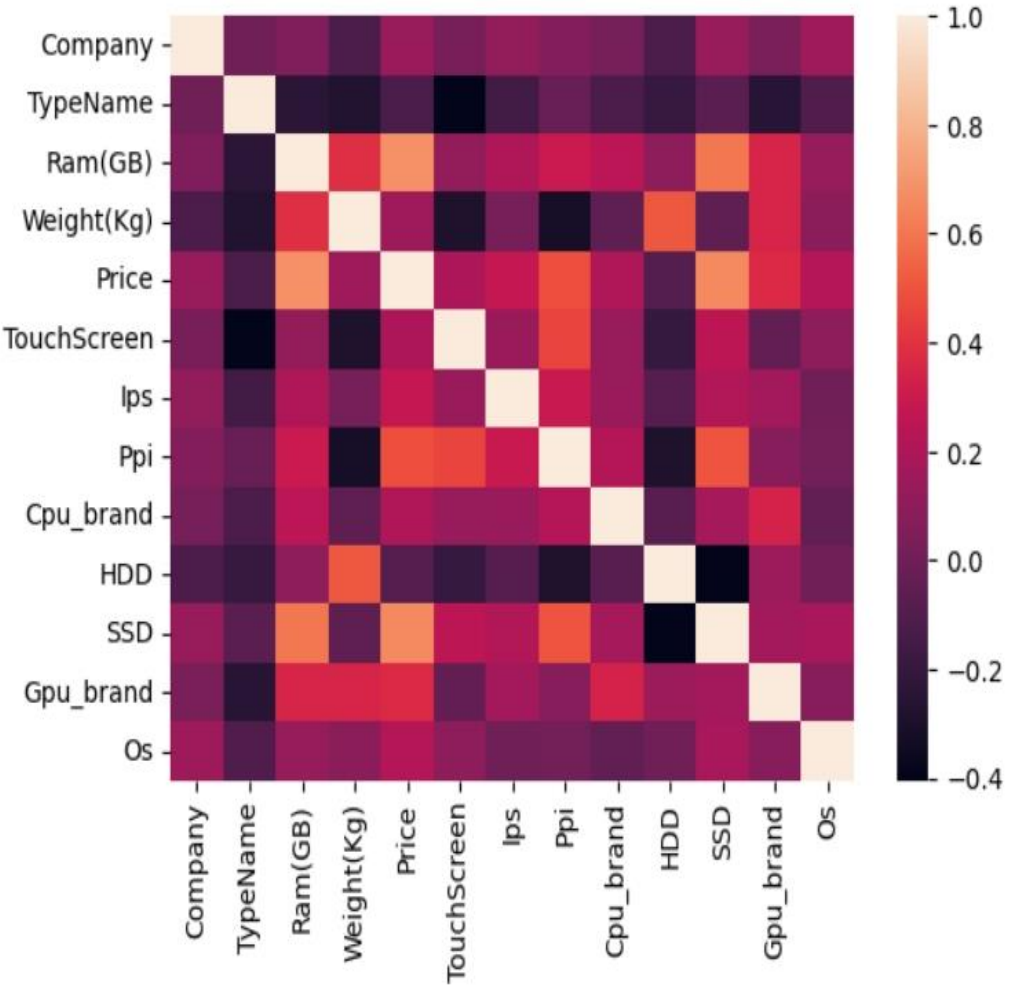


Figure 7: This plot gives the average laptop prices by operating system.

This bar chart displays the average prices of laptops across different operating systems. Each bar represents the mean price for laptops running a specific OS, with the height of the bar indicating the average cost. The

chart highlights how laptop prices vary by operating system, showing which OS tends to be associated with higher or lower-priced models. This information can be useful for understanding pricing trends related to different operating systems and may help in evaluating cost-efficiency or market positioning of various OS options.

3.2.6 Correlation Matrix



Understanding correlation (r) is crucial in statistics as it quantifies the strength of the relationship between two variables, aiding in predicting one variable based on another correlated one. This measure is a bivariate statistic, involving pairs of features. A correlation matrix summarizes all pairwise correlations within a dataset. Coefficients close to 1.0 (between 0.9 and 1.0) indicate a very strong relationship, while those between 0.7 and 0.9 show a high association. Coefficients in the range of 0.5 to 0.7 reflect moderate correlation, and values between 0.3 and 0.5 indicate low

correlation. Coefficients below 0.2 suggest minimal linear correlation and may contribute little to predicting significant outcomes. Furthermore, the presence of multicollinearity suggests that some features are highly correlated with each other.

3.3 Data Pre-processing

Data preprocessing is an essential part of data mining and involves several steps to prepare and transform data into a format suitable for machine learning models. This process includes various tasks such as handling missing values, detecting and addressing outliers, encoding categorical features, and engineering new features.

- Handling Missing Values: This involves identifying and dealing with any gaps or missing entries in the dataset. Methods might include imputation (filling in missing values with estimates), removal of records with missing values, or using algorithms that can handle missing data.

- Outlier Detection: Outliers are data points that significantly deviate from other observations. Identifying and managing outliers is crucial because they can skew results and affect the performance of machine learning models. Techniques such as statistical tests, visualizations, or domain knowledge are used to detect and handle outliers.

- Feature Encoding: Machine learning algorithms often require numerical input, so categorical features (like labels or categories) need to be converted into a numerical format. This can be achieved through techniques like one-hot encoding or label encoding.

- Feature Engineering: This involves creating new features or modifying existing ones to improve the performance of machine learning models. It may include generating interaction terms, polynomial features, or aggregating data.

- Data Normalization: Normalizing or scaling the data ensures that features contribute equally to the analysis. This step adjusts the

values of features to a common scale, often between 0 and 1, or standardizes them to have a mean of zero and a standard deviation of one.

- Data Cleaning and Integration: Data cleaning involves removing or correcting inaccuracies and inconsistencies in the data. Integration combines data from different sources into a unified dataset.

- Data Transformation and Reduction: Data transformation modifies the data into a suitable format or structure for analysis, while data reduction techniques like dimensionality reduction or sampling help manage large datasets by reducing their size or complexity.

For this dataset, preprocessing was performed using Python libraries such as NumPy and Pandas. The dataset was found to be complete, with no null values, and predominantly consists of categorical variables with integer data types. These preprocessing steps are vital for ensuring the and dataset is clean, well-structured, and ready for further analysis modelling**.**

3.4 Scaling the data

Feature scaling is a fundamental step in machine learning preprocessing that ensures features contribute equally to model performance by standardizing their ranges. In this project, we applied Scikit-learn' s `StandardScaler () 'to standardize our data, which adjusts each feature to have a mean of 0 and a standard deviation of 1. This transformation is vital as it makes the data unitless, removing biases that could arise from features with different scales or units. Standardization helps algorithms converge faster and improves their effectiveness, particularly for those that assume normally distributed data or are sensitive to the scale of input features, such as linear regression or support vector machines. By using `StandardScaler () ' ,we enhance the model's ability to interpret and learn from the data more effectively, leading to better and more reliable outcomes.

3.5 Split of the data

Data splitting is a crucial step in machine learning that involves dividing the dataset into two subsets: one for training the model and another for testing its performance. This process allows us to build a predictive model on one portion of the data (the training set) and then evaluate its accuracy and generalizability on a separate, unseen portion (the testing set). In this project, we used the `train_test_split () ` function from Scikit-learn to achieve this partitioning. Specifically, 80% of the data was allocated for training the model to predict outcomes, while the remaining 20% was reserved for testing the model's accuracy. This approach helps in assessing the model's performance and comparing the efficacy of different algorithms in predictive modelling tasks.

3.6 Regression Models

3.6.1 Linear Regression

The regression model has been fitted and evaluated with the following performance metrics:

- R-squared Score: 0.672
- Mean Squared Error: 0.132
- Mean Absolute Error: 0.279
- Root Mean Squared Error: 0.363

These metrics are derived from the predictions of the model compared to the actual values, providing insights into the model.

3.6.2 Ridge Regression

The regression model has been fitted and evaluated with the following performance metrics:

- R-squared Score: 0.671
- Mean Squared Error: 0.132
- Mean Absolute Error: 0.279
- Root Mean Squared Error: 0.364

These metrics are derived from the predictions of the model compared to the actual values, providing insights into the model.

3.6.3 Random Forest Regression

The regression model has been fitted and evaluated with the following performance metrics:

- R-squared Score: 0.906
- Mean Squared Error: 0.037
- Mean Absolute Error: 0.152
- Root Mean Squared Error: 0.194

These metrics are derived from the predictions of the model compared to the actual values, providing insights into the model.

3.6.4 Support Vector Machine

The regression model has been fitted and evaluated with the following performance metrics:

- R-squared Score: 0.893
- Mean Squared Error: 0.042
- Mean Absolute Error: 0.154
- Root Mean Squared Error: 0.207

These metrics are derived from the predictions of the model compared to the actual values, providing insights into the model.

3.6.5 AdaBoost Regression

The regression model has been fitted and evaluated with the following performance metrics:

- R-squared Score: 0.840
- Mean Squared Error: 0.064
- Mean Absolute Error: 0.202
- Root Mean Squared Error: 0.254

These metrics are derived from the predictions of the model compared to the actual values, providing insights into the model.

Chapter 4

Conclusion

Predicting laptop prices plays a crucial role in today's tech market, benefiting both consumers and businesses. For consumers, accurate price predictions help them make informed decisions, ensuring they get the best value for their money by understanding the factors that influence prices, such as brand, specifications, and market trends. This helps them compare products more effectively and avoid overpaying. For retailers and manufacturers, price prediction aids in setting competitive prices, optimizing profit margins, and responding to market demand. It also helps in managing inventory and forecasting trends, allowing businesses to allocate resources more efficiently. In a rapidly evolving tech landscape, accurate laptop price predictions enhance decision-making, improve customer satisfaction, and support business growth.

The objective of this project was to build and compare different regression models to determine the best-performing one for the given dataset. Through thorough data analysis and model evaluation, the following key findings were obtained:

- Most laptops are priced around ₹10.5 lakhs, with a slight right skew indicating a few high-priced outliers.

- Laptop weights are generally concentrated within a common range, with some variability, indicating standard categories with some heavier or lighter models.

- A positive correlation exists between RAM and price, with prices generally increasing as RAM increases, though other factors also influence the price.

- Laptop weights mostly fall within a specific range, suggesting common standards, with variations reflecting different types like ultra books or gaming laptops.

- Certain brands have a higher frequency of laptops, indicating which companies are most represented in the dataset.

- Mid-range laptop prices are more common, with fewer expensive models contributing to the overall price distribution.
- RAM is a key factor in determining laptop prices, but other features also significantly contribute to pricing variations.

- Random Forest, with an accuracy of 0.906, is an effective algorithm for predicting laptop prices, outperforming other models tested.

Reference

[1] B. Prajna, A. Lekha Sri, A. Sahiti, A. Bhagyasri Teja, and A. Arzoo, "Laptop Price Prediction Using Supervised Machine Learning Techniques," *Journal of Computational Intelligence and Machine Learning*, 2023.

[2] D. Siburian, D. R. H. Sitompul, S. H. Sinurat, A. Situmorang, R. Ziegel, and E. Indra, "Impact of COVID-19 on Laptop Demand and Price Prediction for WFH Setups Using Machine Learning," *Journal of Data Science and Analytics*, 2022.

[3] V. Surjuse, S. Lohakare, A. Barapatre, and A. Chapke, "Predicting Laptop Prices Using Multiple Linear Regression," *International Journal of Machine Learning and Applications*, 2022.

[4] M. A. Shaik, M. Varshith, S. SriVyshnavi, N. Sanjana, and R. Sujith, "A Machine Learning Approach to Laptop Price Prediction: Addressing the Complexity of Specifications and Brands," *Journal of Advanced Computational Techniques*, 2022.

[5] P. Manegopale, K. Nerpagar, S. Sawant, M. Shinde, and K. Chandarkar, "A Comprehensive Laptop Price Prediction Model: A Three-Phase Approach," *Journal of Applied Machine Learning Research*, 2023.

[6] C. Janiesch, P. Zschech, and K. Heinrich, "Machine Learning and Deep Learning," *Electronic Markets*, vol. 31, no. 4, pp. 685-695, 2021.

[7] https://www.geeksforgeeks.org/regression-in-machine-learning/

[8] Machine Learning: What It is, Tutorial, Definition, Types - Javatpoint

[9] Implementing the AdaBoost Algorithm From Scratch - GeeksforGeeks

[10] Dataset link-Laptop Price Prediction cleaned Dataset (kaggle.com)

Appendix

LAPTOP PRICE PREDICTION

Let's Dive into it

Importing necessary libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Read 'laptop_data_cleaned.csv' dataset and store it inside a variable

[3]:
```
df=pd.read_csv('laptop_data_cleaned.csv')
```

Data Cleaning

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

[7]:
```
Company        0
TypeName       0
Ram            0
Weight         0
Price          0
TouchScreen    0
Ips            0
Ppi            0
Cpu_brand      0
HDD            0
SSD            0
Gpu_brand      0
Os             0
dtype: int64
```

Check head

]:
```
df.head()
```

]:

	Company	TypeName	Ram	Weight	Price	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	Apple	Ultrabook	8	1.37	11.175755	0	1	226.983005	Intel Core i5	0	128	Intel	Mac
1	Apple	Ultrabook	8	1.34	10.776777	0	0	127.677940	Intel Core i5	0	0	Intel	Mac
2	HP	Notebook	8	1.86	10.329931	0	0	141.211998	Intel Core i5	0	256	Intel	Others
3	Apple	Ultrabook	16	1.83	11.814476	0	1	220.534624	Intel Core i7	0	512	AMD	Mac
4	Apple	Ultrabook	8	1.37	11.473101	0	1	226.983005	Intel Core i5	0	256	Intel	Mac

View basic statistical information about the dataset

[15]:
```
df.describe()
```

[15]:

	Ram	Weight	Price	TouchScreen	Ips	Ppi	HDD	SSD
count	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000	1273.000000
mean	8.447761	2.041100	10.828218	0.146897	0.279654	146.950812	413.715632	186.252946
std	5.098771	0.669241	0.619565	0.354142	0.449006	42.926775	518.054486	186.531571
min	2.000000	0.690000	9.134616	0.000000	0.000000	90.583402	0.000000	0.000000
25%	4.000000	1.500000	10.387379	0.000000	0.000000	127.335675	0.000000	0.000000
50%	8.000000	2.040000	10.872255	0.000000	0.000000	141.211998	0.000000	256.000000
75%	8.000000	2.310000	11.287447	0.000000	1.000000	157.350512	1000.000000	256.000000
max	64.000000	4.700000	12.691441	1.000000	1.000000	352.465147	2000.000000	1024.000000

[21]:
```
#checking for duplicate values
df[df.duplicated()]
```

[21]:

	Company	TypeName	Ram	Weight	Price	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
1004	Toshiba	Notebook	4	1.2	11.061462	0	0	165.632118	Intel Core i5	0	128	Intel	Windows

[23]:
```
#removing duplicate values
df.drop_duplicates(keep=False, inplace=True)
```

[25]:
```
df.rename(columns={'Ram': 'Ram(GB)'}, inplace=True)
```

[27]:
```
df.rename(columns={'Weight': 'Weight(Kg)'}, inplace=True)
```

[29]:
```
df.head()
```

[29]:

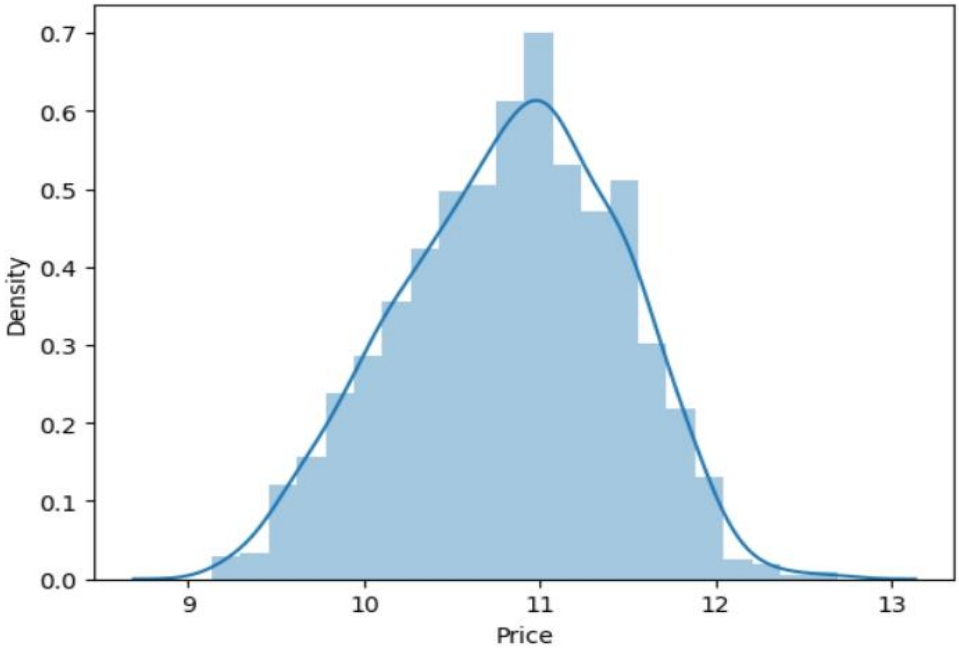
	Company	TypeName	Ram(GB)	Weight(Kg)	Price	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	Apple	Ultrabook	8	1.37	11.175755	0	1	226.983005	Intel Core i5	0	128	Intel	Mac
1	Apple	Ultrabook	8	1.34	10.776777	0	0	127.677940	Intel Core i5	0	0	Intel	Mac
2	HP	Notebook	8	1.86	10.329931	0	0	141.211998	Intel Core i5	0	256	Intel	Others
3	Apple	Ultrabook	16	1.83	11.814476	0	1	220.534624	Intel Core i7	0	512	AMD	Mac
4	Apple	Ultrabook	8	1.37	11.473101	0	1	226.983005	Intel Core i5	0	256	Intel	Mac

Data Visualization

Displaying the distribution of prices of the laptops

```
33]: sns.distplot(df['Price'])
```

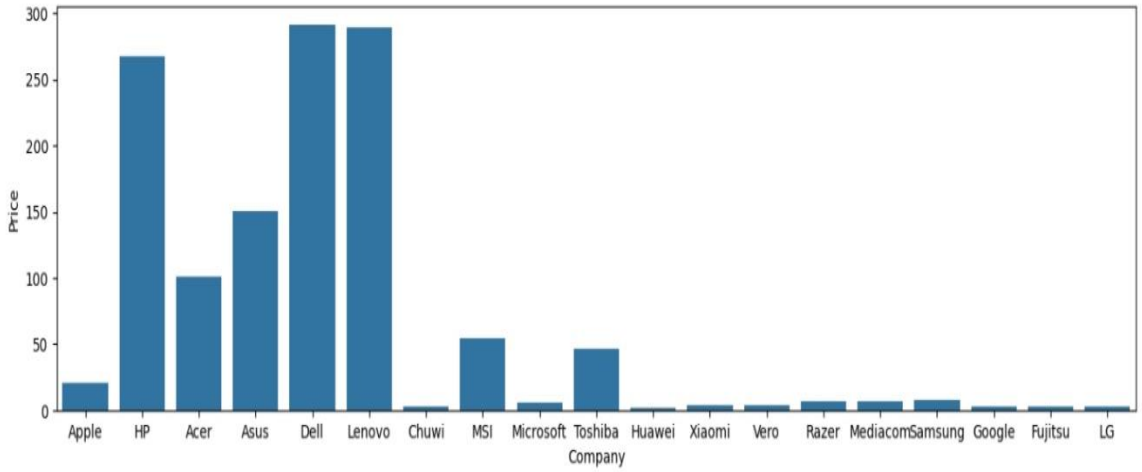
```
33]: <Axes: xlabel='Price', ylabel='Density'>
```



Clearly, the distribution of the price is skewed.

Comparison of Laptop Prices

```
[37]: plt.figure(figsize = (16,4))
sns.countplot(data = df, x = 'Company')
plt.ylabel('Price')
plt.show()
```

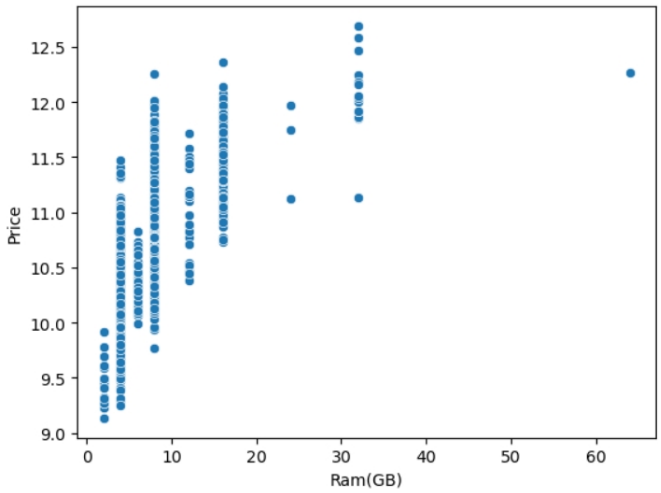


Laptops from Dell, Lenovo and HP tend to command a higher price compared to others, suggesting brand influence on pricing

Relationship between RAM Size and Laptop Price

```
[41]: sns.scatterplot(x='Ram(GB)', y='Price', data=df)
```

```
[41]: <Axes: xlabel='Ram(GB)', ylabel='Price'>
```

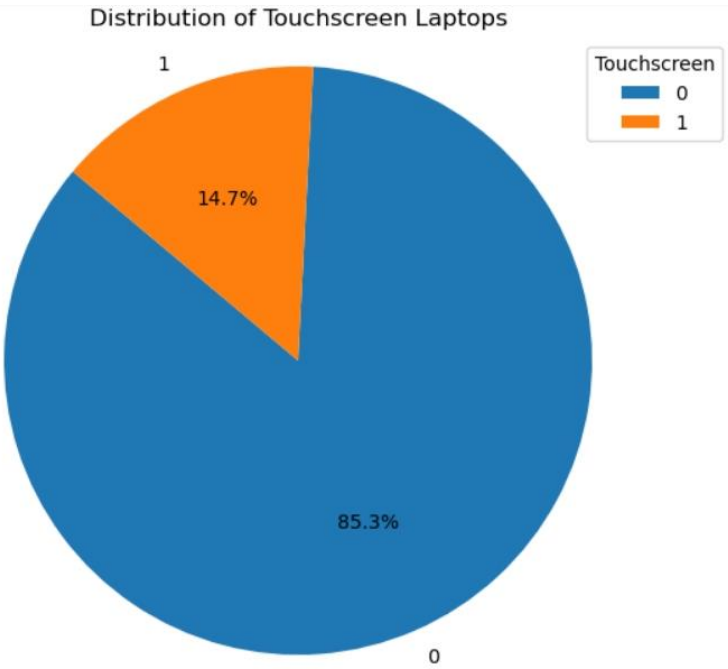


As RAM size increases, the price of laptops also tends to increase, indicating a positive correlation between RAM and price

Distribution of Touchscreen Laptops

```
[45]: touchscreen_counts = df['TouchScreen'].value_counts()
```

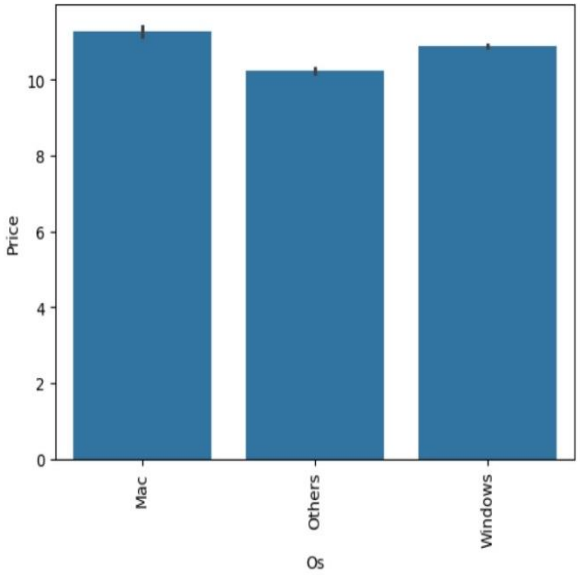
```
[47]: plt.figure(figsize=(8, 6))
plt.pie(touchscreen_counts, labels=touchscreen_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Touchscreen Laptops')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.legend(touchscreen_counts.index, title="Touchscreen", loc="best")  # Add a legend
plt.show()
```



Approximately 85.3% of laptops in the dataset feature touchscreen functionality, indicating a growing trend towards interactive displays."

Average Laptop Price by Operating System

[51]:
```
sns.barplot(x=df['Os'],y=df['Price'])
plt.xticks(rotation='vertical')
plt.show()
```

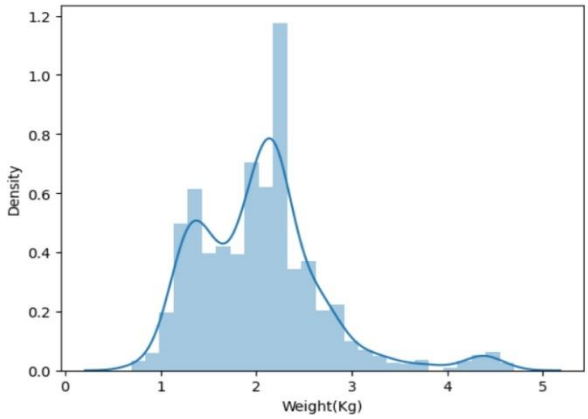


The bar plot of average laptop prices by operating system shows that laptops running macOS (Mac) are generally more expensive compared to those running other operating systems (Others). This indicates a higher price point for Mac laptops in this dataset.

Distribution of Laptop Weights

[55]:
```
sns.distplot(df['Weight(Kg)'])
```

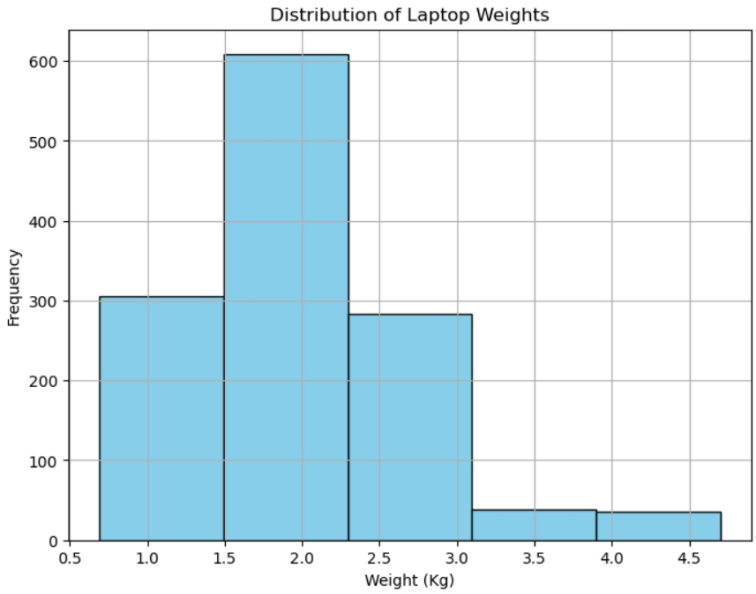
[55]:
```
<Axes: xlabel='Weight(Kg)', ylabel='Density'>
```



The distribution plot of laptop weights shows that the majority of the laptops are clustered around weights of approximately 1.34 to 1.37 kg, with fewer laptops being significantly heavier at around 1.83 to 1.86 kg. This suggests that lighter laptops are more common in this dataset.

Distribution of Laptop Weights

[59]:
```
plt.figure(figsize=(8, 6))
plt.hist(df['Weight(Kg)'], bins=5, color='skyblue', edgecolor='black')
plt.title('Distribution of Laptop Weights')
plt.xlabel('Weight (Kg)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

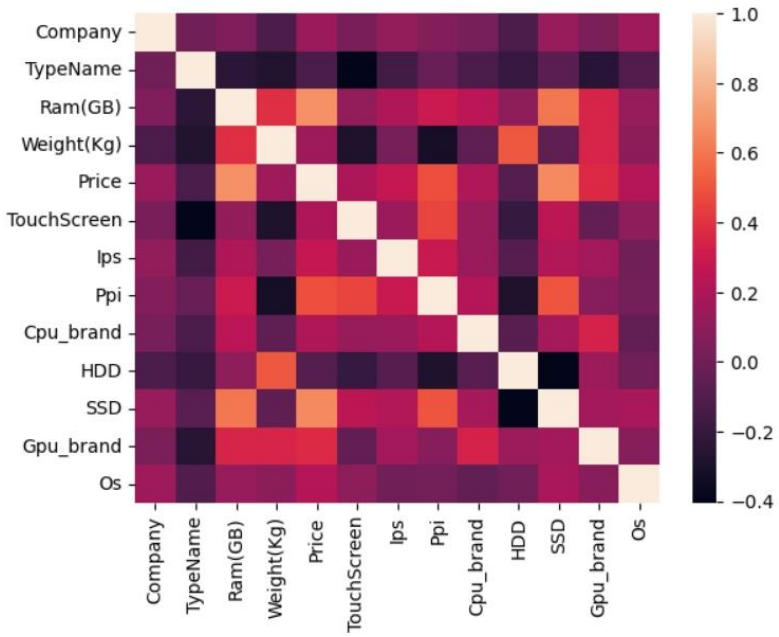


The majority of laptops in the dataset weigh between 0.7Kg-3.1kg , with a few outliers at the heavier end, suggesting a range of portability options for consumers

Correlation Heatmap of Laptop Features

[109]:
```
sns.heatmap(df.corr())
```

[109]: <Axes: >



The heatmap illustrates the correlation between different features of laptops in the dataset. Key insights include:

- Strong positive correlations between RAM and Price, as well as between SSD storage and Price, indicating that higher RAM and SSD capacities tend to increase laptop prices.
- Moderate positive correlations between IPS (In-Plane Switching) display and PPI (Pixels Per Inch), suggesting that laptops with IPS screens often have higher pixel densities.

This visualization helps identify which features might influence each other and their collective impact on laptop prices.

Data Preprocessing

[63]:
```
df.head()
```

[63]:

	Company	TypeName	Ram(GB)	Weight(Kg)	Price	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	Apple	Ultrabook	8	1.37	11.175755	0	1	226.983005	Intel Core i5	0	128	Intel	Mac
1	Apple	Ultrabook	8	1.34	10.776777	0	0	127.677940	Intel Core i5	0	0	Intel	Mac
2	HP	Notebook	8	1.86	10.329931	0	0	141.211998	Intel Core i5	0	256	Intel	Others
3	Apple	Ultrabook	16	1.83	11.814476	0	1	220.534624	Intel Core i7	0	512	AMD	Mac
4	Apple	Ultrabook	8	1.37	11.473101	0	1	226.983005	Intel Core i5	0	256	Intel	Mac

Convert the rest of the categorical values into numeric using LabelEncoding and store the results in a old dataframe df

[66]:
```
from sklearn.preprocessing import LabelEncoder
```

[68]:
```
a=LabelEncoder()
```

[70]:
```
df['Company']=a.fit_transform(df['Company'])
```

[72]:
```
df['TypeName']=a.fit_transform(df['TypeName'])
```

[74]:
```
df['Cpu_brand']=a.fit_transform(df['Cpu_brand'])
```

[76]:
```
df['Gpu_brand']=a.fit_transform(df['Gpu_brand'])
```

[78]:
```
df['Os']=a.fit_transform(df['Os'])
```

[80]:
```
df.head()
```

[80]:

	Company	TypeName	Ram(GB)	Weight(Kg)	Price	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	1	4	8	1.37	11.175755	0	1	226.983005	2	0	128	1	0
1	1	4	8	1.34	10.776777	0	0	127.677940	2	0	0	1	0
2	7	3	8	1.86	10.329931	0	0	141.211998	2	0	256	1	1
3	1	4	16	1.83	11.814476	0	1	220.534624	3	0	512	0	0
4	1	4	8	1.37	11.473101	0	1	226.983005	2	0	256	1	0

[82]:
```
df.columns
```

[82]:
```
Index(['Company', 'TypeName', 'Ram(GB)', 'Weight(Kg)', 'Price', 'TouchScreen',
       'Ips', 'Ppi', 'Cpu_brand', 'HDD', 'SSD', 'Gpu_brand', 'Os'],
      dtype='object')
```

Split the columns into input and target variables (X and y)

[85]:
```
X = df.drop(columns=['Price'])
y = df['Price']
```

[87]:
```
X.head()
```

[87]:

	Company	TypeName	Ram(GB)	Weight(Kg)	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	1	4	8	1.37	0	1	226.983005	2	0	128	1	0
1	1	4	8	1.34	0	0	127.677940	2	0	0	1	0
2	7	3	8	1.86	0	0	141.211998	2	0	256	1	1
3	1	4	16	1.83	0	1	220.534624	3	0	512	0	0
4	1	4	8	1.37	0	1	226.983005	2	0	256	1	0

[89]:
```
y.head()
```

[89]:
```
0    11.175755
1    10.776777
2    10.329931
3    11.814476
4    11.473101
Name: Price, dtype: float64
```

Carry out Feature scaling using StandardScaler

[92]:
```
from sklearn.preprocessing import StandardScaler
```

[94]:
```
scaler=StandardScaler()
```

[96]:
```
x_scaler=scaler.fit_transform(X)
```

[98]:
```
X=pd.DataFrame(x_scaler,columns=X.columns)
X.head()
```

[98]:

	Company	TypeName	Ram(GB)	Weight(Kg)	TouchScreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	-1.335637	1.135574	-0.089209	-1.005611	-0.415342	1.603192	1.864618	-0.435879	-0.799939	-0.312688	-0.270173	-4.584399
1	-1.335637	1.135574	-0.089209	-1.050477	-0.415342	-0.623756	-0.448177	-0.435879	-0.799939	-0.998682	-0.270173	-4.584399
2	0.136904	0.333733	-0.089209	-0.272815	-0.415342	-0.623756	-0.132971	-0.435879	-0.799939	0.373306	-0.270173	-2.102810
3	-1.335637	1.135574	1.480121	-0.317680	-0.415342	1.603192	1.714437	0.584383	-0.799939	1.745294	-1.816971	-4.584399
4	-1.335637	1.135574	-0.089209	-1.005611	-0.415342	1.603192	1.864618	-0.435879	-0.799939	0.373306	-0.270173	-4.584399

Split the dataset into training and testing set

101]:
```
from sklearn.model_selection import train_test_split
```

103]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.15,random_state=2)
```

Machine Learning Models

Linear Regression

[107]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Fit the model
model = LinearRegression()
model.fit(X_train, y_train)
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# Predict on the test set
y_pred = model.predict(X_test)
print('Predicted values (y_pred):', y_pred)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

# Calculate Root Mean Squared Error
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

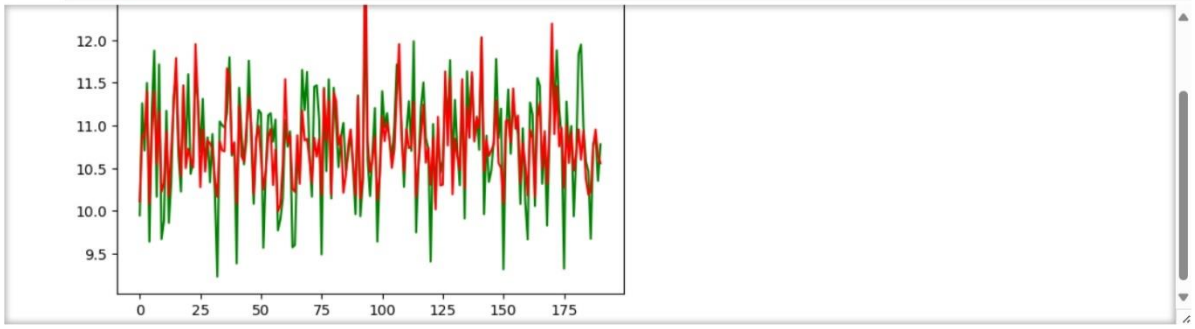
```
Coefficients: [ 0.03382401  0.04138846  0.21779438  0.03125312  0.00801485  0.03773547
  0.1308287  -0.01670904  0.00843727  0.17929758  0.0978921   0.05914357]
Intercept: 10.826507031135893
Predicted values (y_pred): [10.1133436  10.87515507 10.86927075 11.39967776 10.08948661 10.91770865
 11.39873299 10.55027036 11.16293478 10.22622448 10.32914655 10.93095582
 10.17129495 10.71430358 11.23172607 11.7899848  10.82620605 10.40991249
 11.46510055 10.5019523  10.73029107 10.60644731 10.50909947 11.95290914
 11.29241518 10.28018708 10.95442363 10.46119171 10.82719098 10.79150029
 10.72845136 10.37010992 10.16525637 10.81311909 10.70989171 10.6994747
 11.6686464  11.54047758 10.67478666 10.80445998 10.08739336 11.23125868
 10.63772726 10.5790225  10.99613488 11.34289346 11.01504674 10.20027189
 10.84857831 10.99830644 10.64758683 10.25373217 10.46518086 10.8675659
 10.96227043 10.30415592 10.71863775  9.99722343 10.07192119 10.55027036
 11.54207885 10.79555951 10.88162198 10.26398201 10.22291098 10.88613188
 10.31846575 11.17477502 10.82762706 10.84336247 10.61071848 10.32944497
 10.8554009  10.63562127 10.84545599 10.1908845  11.4249119  10.82240196
 11.29263074 10.19520488 11.38159805 11.29187983 10.77586202 10.88953329
 10.2144565  10.42560468 10.77217314 10.95472762 10.56855656 10.18589694
 11.33467221 10.14023809 10.38817847 13.2628917  10.82620605 10.45419506
 10.59650022 10.87235072 10.13050002 10.41815725 11.09918703 10.81829986
 11.03445259 10.85036422 10.5048259  10.72527395 11.35524555 11.95290914
 10.83770462 10.46232495 10.95815718 10.7359297  10.7589111  11.27573596
 10.17129495 10.57052556 10.88939832 11.24410416 10.56576935 10.74026122
 10.30708153 10.85588611 10.02027634 11.10170534 10.29924559 10.30862449
 11.63235342 10.76693795 11.54775706 10.19581614 10.84846761 10.63689492
 10.48352502 11.54275428 10.27197633 11.22705215 10.85718575 11.62464235
 10.81291191 11.08789437 10.97703808 12.02991119 10.4657313  10.83382001
 10.64409837 10.71065408 10.78398555 11.29608383 10.55984023 10.50564765
 10.08736929 11.05248029 11.0643898  10.8018974  11.43379926 10.96423327
 11.11707629 10.3320673  10.80115604 10.54430984 10.18398619 10.95039959
 10.85014054 10.16884934 11.09446166 11.26680834 10.49864163 10.93508711
```

```
 10.32130086 10.97029468 12.19281074 10.90041517 11.46115132 10.75659774
 10.97177522 10.27436395 11.00292644 10.56117977 10.92154063 10.47286687
 10.67022393 10.95549678 10.59939106 10.93629375 10.37971098 10.19270039
 10.22669187 10.74923215 10.95334134 10.64681212 10.56046551]
R-squared Score: 0.6726062753447974
Mean Squared Error: 0.1323422224542117
Mean Absolute Error: 0.2791586776922359
Root Mean Squared Error: 0.36378870578154526
```

```
c=[i for i in range(191)]
len(c)
```

```
191
```

```
# plt.scatter(y_test,y_pred)
# len(y_pred)
plt.plot(c,y_test,"green")#actual values
plt.plot(c,y_pred,'red')#predicted values
plt.show()
```



The alignment between the red (predicted) and green (actual) lines indicates the model's ability to approximate the true values. Overall, the close alignment between the predicted and actual values suggests that the model performs well in capturing the underlying patterns and trends in the data.

Ridge Regression

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Create and fit the model with hyperparameter tuning
ridge = Ridge()
parameters = {'alpha': [0.1, 1, 10]}
ridge_regressor = GridSearchCV(ridge, parameters, scoring='neg_mean_squared_error', cv=5)
ridge_regressor.fit(X_train, y_train)

print("Best alpha:", ridge_regressor.best_params_)

# Predict on the test set
y_pred = ridge_regressor.predict(X_test)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

# Calculate Root Mean Squared Error
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
Best alpha: {'alpha': 10}
R-squared Score: 0.671786004359015
Mean Squared Error: 0.132673800236857
Mean Absolute Error: 0.2795254434043059
Root Mean Squared Error: 0.36424414921431064
```

Random Forest

[124]:
```
from sklearn.ensemble import RandomForestRegressor


# Fit the model
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Predict on the test set
y_pred = rf_regressor.predict(X_test)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

# Calculate Root Mean Squared Error
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
R-squared Score: 0.9066166251091478
Mean Squared Error: 0.03774832088289335
Mean Absolute Error: 0.15275713908032876
Root Mean Squared Error: 0.19428927114715663
```

Support Vector Machine

[126]:
```
from sklearn.svm import SVR

# Create and fit the model
svr_regressor = SVR(kernel='rbf')
svr_regressor.fit(X_train, y_train)

# Predict on the test set
y_pred = svr_regressor.predict(X_test)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

# Calculate Root Mean Squared Error
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
R-squared Score: 0.8936470510369804
Mean Squared Error: 0.042991006150617454
Mean Absolute Error: 0.15454138302297304
Root Mean Squared Error: 0.2073427263026544
```

AdaBoost Regression

[128]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

# Initialize the base model for AdaBoost
base_model = DecisionTreeRegressor(max_depth=4)

# Initialize AdaBoost Regressor
ada_regressor = AdaBoostRegressor(base_model, n_estimators=50, random_state=42)

# Create and fit the model
ada_regressor.fit(X_train, y_train)

# Predict on the test set
y_pred = ada_regressor.predict(X_test)

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared Score:", r2)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

# Calculate Root Mean Squared Error
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
R-squared Score: 0.8400985625542385
Mean Squared Error: 0.06463688828331032
Mean Absolute Error: 0.20213231060149278
Root Mean Squared Error: 0.25423785769100227
```

[147]:
```
# Create a DataFrame with two columns
data = {
    'Algorithm': ['Linear Regression', 'Rigde Regression', 'Random Forest Regression', 'Support Vector Regression','AdaBoostRegressor'],
    'R2 Score': ['0.672', '0.671', ' 0.906', '0.893','0.840']
}

df1 = pd.DataFrame(data)

# Display the DataFrame
print(df1)
```

```
                Algorithm R2 Score
0         Linear Regression    0.672
1          Rigde Regression    0.671
2   Random Forest Regression    0.906
3  Support Vector Regression    0.893
4         AdaBoostRegressor    0.840
```

Clearly in this dataset, Random Forest Regression (RFR) consistently exhibits superior performance, likely attributed to its capability to effectively capture intricate patterns in the data while mitigating overfitting, resulting in higher predictive accuracy and generalization