Predictive Analytics for Next-Day Stock Market Closing Price Using AI

MAJOR PROJECT REPORT

SEMESTER- 2

FOUR-YEAR UNDERGRADUATE PROGRAMME
(DESIGN YOUR DEGREE)

SUBMITTED TO

UNIVERSITY OF JAMMU, JAMMU

TEAM MEMBERS	ROLL NO
Avichal Badyal	DYD-23-02
Divya Verma	DYD-23-04
Gourav Sharma	DYD-23-06
Ishaan Uppal	DYD-23-08
Vidhita Arora	DYD-23-22

UNDER THE MENTORSHIP OF

Prof. KS CHARAK	Dr. JATINDER MANHAS	Dr. SANDEEP ARYA
Professor	Associate Professor	Assistant Professor
Department of Mathematics	Computer Science and IT	Department of Physics
University of Jammu, Jammu	University of Jammu, Jammu	University of Jammu, Jammu

Submitted on: ____ August, 2024

CERTIFICATE

The report titled "**Predictive Analytics for Next-Day Stock Market Closing Price Using AI**" was done by the Group including group members- Avichal Badyal, Divya Verma, Gourav Sharma, Ishaan Uppal and Vidhita Arora. The project served as a significant undertaking for Semester 2 of their academic program. Under the supervision and guidance of Prof. KS Charak, Dr. Jatinder Manhas and Dr. Sandeep Arya for the partial fulfillment of the Design Your Degree, Four Year Undergraduate Programme at the University of Jammu, Jammu and Kashmir. This project report is original and has not been submitted elsewhere for any academic recognition.

Signature of Students

Signature of Mentors:

a) Prof. KS Charak

b) Dr. Jatinder Manhas

c) Dr. Sandeep Arya

Prof. Alka Sharma

Director, SIIEDC, University of Jammu

ACKNOWLEDGEMENT

We can acknowledge all those who helped and guided us during our research work. First and foremost, we thank almighty god from the depth of our hearts for generating enthusiasm and granting us spiritual strength to pass through this challenge successfully.Words are too meagre to express our esteem indebtedness and wholehearted sense of gratitude towards our research guide and supervisors Prof. KS Charak, Dr. Jatinder Manhas and Dr. Sandeep Arya. It's our pleasure beyond words to express our deep sense of feelings for their inspiring guidance, generous encouragement, and well-versed advice. They provided us with undaunted encouragement and support despite his/her busy schedules. We feel extremely honoured for the opportunity conferred upon us to work under their perpetual motivation. We are highly thankful to all respected teachers for their immense help during the conduct of our research work.

We are extremely fortunate to have an opportunity to express our heartiest gratitude to Prof. Alka Sharma, Director, SIIEDC, University of Jammu, Jammu, and member of the advisory committee for her valuable advice and generous help and for providing us all the necessary facilities from the department.The words are small trophies to express our feelings of affection and indebtedness to our friends who helped us throughout the research, and whose excellent company, affection, and co-operation helped us in carrying out our research work with joy and happiness. We acknowledge all the people, mentioned or not mentioned here, who have silently wished and given fruitful suggestions and helped us achieve the present goal.

ABSTRACT

Stock market prediction is a significant challenge due to the dynamic and volatile nature of financial markets. Traditional prediction methods like fundamental and technical analysis often fail to capture the nonlinear patterns inherent in stock prices. This project addresses these limitations by utilizing advanced artificial intelligence (AI) and machine learning (ML) techniques to predict the next-day closing price of Axis Bank stock, one of India's major private sector banks. The project applies a range of machine learning algorithms, including Decision Trees, Random Forest, K-Nearest Neighbors (KNN), Gradient Boosting, and Linear Regression, to analyze historical stock data.

The data, sourced from Yahoo Finance, consists of three months of Axis Bank's stock prices, including key features such as 'Open', 'High', 'Low', 'Adj Close', and trading volume. The objective is to forecast short-term price fluctuations, with a focus on understanding which models offer the best predictive accuracy. Each algorithm is evaluated based on its performance metrics, such as Mean Squared Error (MSE) and R-squared scores, to compare their predictive capabilities.

The project aims to contribute to the growing field of AI-driven financial forecasting by offering a detailed analysis of how different algorithms perform on volatile stock market data. The findings could assist traders, investors, and financial analysts in making more informed decisions, ultimately reducing investment risks and improving returns. Through this project, we also highlight the importance of feature engineering, hyperparameter tuning, and model evaluation in developing robust stock price prediction models.

INDEX:

CHAPTER 1

INTRODUCTION

1.1 Background

The stock market is a complex and dynamic environment, influenced by a myriad of factors including economic indicators, political events, market sentiment, and global trends. These variables contribute to the inherent volatility of stock prices, making accurate prediction a challenging endeavor for investors, traders, and financial analysts alike. Traditionally, stock market predictions have relied on fundamental and technical analysis, where analysts study financial statements, economic indicators, and historical price movements to make informed decisions. However, these methods often struggle to capture the nonlinear and dynamic patterns present in stock market data.

With the advancement of artificial intelligence (AI) and machine learning (ML) technologies, new opportunities have emerged in financial forecasting. AI models, particularly those based on machine learning, can analyze vast amounts of historical data, identify hidden patterns, and make predictions that traditional methods might overlook. These models can continuously improve by learning from data, providing a more adaptive approach to stock market prediction.

The application of AI in finance has garnered significant interest, particularly in the area of stock market prediction. Machine learning algorithms offer the potential to model complex market behaviors and provide more accurate forecasts. This project explores the use of AI techniques to predict stock prices, focusing specifically on Axis Bank, a major financial institution in India.

1.2 Motivation

The ability to predict stock prices with a high degree of accuracy is a powerful tool for investors and traders. In financial markets, even small improvements in predictive accuracy can translate into significant gains or loss avoidance. Therefore, there is a strong motivation to develop and refine methods that can accurately forecast future stock prices.

The increasing availability of financial data, coupled with advances in AI, presents a unique opportunity to enhance predictive models. Machine learning algorithms have shown promise in capturing intricate patterns and relationships within large datasets, making them suitable candidates for stock market prediction.

Axis Bank, as one of the leading private sector banks in India, has a substantial impact on the Indian stock market. Understanding and predicting the stock price movements of Axis Bank can provide valuable insights for investors and analysts. This project aims to leverage AI techniques to forecast the short-term price movements of Axis Bank's stock, contributing to the growing body of research on AI-driven financial analysis.

1.3 Overview of Machine Learning and Regression

1.3.1 Machine Learning

A subfield of computer science and artificial intelligence called machine learning (ML) is concerned with using data and algorithms to help AI mimic human learning processes and progressively become more accurate.

1.3.2 Types of Machine Learning

a) Supervised Machine Learning

Supervised machine learning, or supervised learning, is characterized by the use of labelled datasets to train algorithms for precise outcome prediction or data classification. The model modifies its weights when input data is entered until a satisfactory fit is achieved. This happens during the cross-validation phase, which makes sure the model doesn't over fit or under fit. Sorting spam into a different folder from your email is just one example of the many real-world problems that supervised learning helps enterprises solve at scale. Neural networks, naïve bayes, logistic regression, random forest, linear regression, and support vector machines (SVM) are a few techniques used in supervised learning.

b) Unsupervised Machine Learning

Unsupervised learning also known as unsupervised machine learning, is the process of analysing and grouping unlabelled datasets (subsets known as clusters) using machine learning algorithms. These algorithms find hidden relationships or patterns in the data without requiring human assistance. This approach is perfect for consumer segmentation, cross-selling tactics, exploratory data analysis, and pattern and image recognition since it can identify patterns and similarities in data. It can also be applied to dimensionality reduction, which lowers the amount of features in a model. Two popular methods for this are singular value decomposition (SVD) and principal component analysis (PCA). Neural networks, probabilistic clustering techniques, and k-means clustering are among more algorithms utilized in unsupervised learning.

c) Semi- Supervised Machine Learning

A satisfying middle ground between supervised and unsupervised learning is provided by semi-supervised learning. It guides categorization and feature extraction from a larger, unlabelled data set during training by using a smaller, labelled data set. The issue of insufficient labelled data for a supervised learning system can be resolved through semi-supervised learning. It's also beneficial if labelling sufficient data would be too expensive [1].

1.3.3 Regression

Regression analysis is a sort of study that makes use of statistical techniques to try and explain and comprehend the links between a dependent variable (also called the outcome variable) and an independent variable (also called predictor variables or explanatory variables). Regression models typically take into account error terms as well, which quantify the margin of error in an analysis to identify discrepancies. Regression analyses are performed primarily for two purposes: either to find causal links or to make predictions. Regression analyses, including multiple regression, nonlinear regression, basic linear regression, and linear regression, can be used to accomplish these objectives [2].

1.4 Objective

The primary objective of this project is to predict the stock prices of Axis Bank using a variety of AI-driven machine learning algorithms. By analyzing historical price data, this study seeks to evaluate the performance of different models in forecasting short-term price movements.

1.5 Scope

This project focuses on the stock price data of Axis Bank over the past three months, sourced from Yahoo Finance. The study concentrates on short-term price prediction, aiming to provide insights into the performance of different AI models when applied to stock market data. By using a relatively short timeframe, the project attempts to capture the immediate fluctuations in stock prices, which can be crucial for day traders and short-term investors.

The five chosen algorithms represent a range of approaches to machine learning, from simple models like Decision Trees, Linear Regression to more complex ensemble methods like Random Forest and Gradient Boosting.

The scope of this project is limited to Axis Bank's stock, and the findings may not necessarily generalize to other stocks or sectors. However, the methodologies and insights gained can serve as a foundation for future research in stock market prediction.

1.6 Significance

This project contributes to the ongoing research and development of AI-driven financial forecasting models. By evaluating the performance of multiple machine learning algorithms, it provides valuable insights into their applicability in predicting stock prices. The comparative analysis of these models offers guidance on selecting the appropriate algorithm for different market conditions and data characteristics.

Moreover, the findings from this project can assist investors in making more informed decisions by identifying which models are more reliable for predicting stock price movements. AI-driven financial tools have the potential to reduce investment risks and improve returns, making them increasingly valuable in the fast-paced world of stock trading.

In summary, this project underscores the role of AI in transforming financial analysis and highlights the potential of machine learning models to enhance the accuracy of stock market predictions. By focusing on Axis Bank, a key player in the Indian financial sector, this study also provides a practical application of AI in a real-world context.

CHAPTER 2

LITERATURE SURVEY

2.1 Stock Market Prediction and Challenges

The stock market is a dynamic and complex system influenced by numerous factors, including economic conditions, market sentiment, political events, and even natural disasters. Traditional methods for predicting stock prices, such as technical analysis and fundamental analysis, have provided useful insights but often struggle to capture the nonlinear relationships and sudden shifts in the market. According to Fama (1970), the Efficient Market Hypothesis (EMH) suggests that stock prices fully reflect all available information, making it difficult to outperform the market consistently through traditional methods.

Despite the unpredictability of stock prices, researchers have increasingly turned to data-driven methods like machine learning (ML) to enhance prediction accuracy. ML techniques allow for the analysis of large datasets and can uncover hidden patterns that traditional methods might overlook. As financial markets have become more data-intensive, the need for sophisticated predictive models has increased [3].

2.2 Role of Data in Stock Market Prediction

One of the critical aspects of stock market prediction is the availability and quality of data. Stock market data typically includes various features such as opening price, closing price, volume, high and low prices, and technical indicators like moving averages. Studies have shown that the inclusion of additional features such as macroeconomic indicators, investor sentiment, and even social media data can improve prediction accuracy [4].

For instance, Tsai and Hsiao (2010) demonstrated that incorporating macroeconomic factors such as interest rates, GDP, and inflation rates along with stock market data enhances the model's predictive power [5]. Other studies, like Ding et al. (2015), have explored the integration of unstructured data, such as news articles or social media sentiment, into stock market models, showing that this can significantly improve short-term predictions [6].

2.3 AI and Stock Market Prediction

Artificial Intelligence (AI) has been increasingly adopted in financial markets due to its ability to process and analyze large volumes of data efficiently. Machine learning (ML) and deep

learning (DL) models have become popular in financial forecasting because they can model complex, nonlinear relationships between market features.

- **Machine Learning**: ML models, especially supervised learning techniques, have been widely used in predicting stock market trends. According to Patel et al. (2015), ML models like Support Vector Machines (SVM) and ensemble methods such as Random Forest outperform traditional regression-based approaches when it comes to predicting market movements [7].
- **Deep Learning**: Recently, deep learning models, especially Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, have gained traction in stock market prediction. These models are particularly suited for time-series data, like stock prices, as they can capture temporal dependencies. Fischer and Krauss (2018) found that LSTM networks significantly outperformed traditional models in predicting stock price movements for short-term trading [8].

2.4 Importance of Feature Engineering

Feature engineering plays a critical role in improving the performance of stock market prediction models. By carefully selecting and transforming input features, researchers can help models better capture the underlying market trends. According to Shah et al. (2019), feature engineering techniques such as moving averages, Bollinger bands, and relative strength index (RSI) can provide valuable signals for predicting stock price fluctuations [9].

In their study, Huang et al. (2005) emphasized the importance of selecting relevant features to avoid overfitting and enhance the generalization ability of predictive models. Moreover, techniques such as dimensionality reduction, including Principal Component Analysis (PCA), have been applied to reduce noise and focus on the most significant variables, which improves model efficiency and accuracy [10].

2.5 Hyperparameter Tuning and Model Optimization

Another key factor in building accurate stock market prediction models is hyperparameter tuning. Machine learning models often have several parameters that control the behavior of the learning algorithm, such as the number of trees in a Random Forest or the learning rate in Gradient Boosting. Optimizing these parameters through techniques such as GridSearchCV or RandomizedSearchCV can substantially improve the performance of the model [11].

Zhong and Enke (2017) demonstrated that hyperparameter optimization plays a crucial role in ensuring that models do not overfit or underperform. They highlighted that models like Gradient Boosting and Random Forest are sensitive to their parameter settings and perform best when optimized carefully [12].

2.6 Application of AI in Financial Forecasting

AI-based models have a broad range of applications in the financial sector, beyond stock price prediction. They are used for credit scoring, fraud detection, portfolio management, and algorithmic trading. One study by Bao et al. (2017) utilized LSTM networks for algorithmic trading and found that the model could successfully generate profitable trading signals based on stock price predictions [13]. Similarly, de Prado (2018) noted that machine learning models could improve portfolio optimization by identifying asset correlations and reducing investment risks [14].

2.7 Summary of Key Findings from the Literature

The literature highlights the growing importance of data-driven methods, particularly AI and machine learning models, in stock market prediction. While traditional methods have their merits, the ability of AI models to process large datasets, capture nonlinear relationships, and continuously improve through learning makes them superior tools for financial forecasting. Studies also emphasize the importance of high-quality data, feature engineering, and hyperparameter tuning in building robust and accurate predictive models. These insights form the foundation for the current study on stock market prediction using advanced AI techniques.

CHAPTER 3

METHODOLOGY

Predicting stock prices is an intricate process that requires the combination of historical data, advanced computational tools, and machine learning techniques. The complexity arises from the stock market's inherent volatility, driven by factors like economic conditions, investor sentiment, and global events. To address these challenges, machine learning models can identify patterns in historical data and make informed predictions about future movements. This chapter outlines the tools and technologies used to develop the stock price prediction model, focusing on the methods applied to extract, process, and analyze stock data.

3.1 Tools

The following tools were essential for the development of the stock market prediction models:

a) **Python:** The core programming language used for building and testing machine learning models. Python's extensive libraries enable efficient data manipulation, statistical analysis, and visualization**.**

b) **Pandas:** A powerful data manipulation tool used to load and clean the dataset. Pandas supports operations such as filtering, grouping, and aggregating data, making it invaluable for preprocessing the stock data.

c) **NumPy:** Employed for numerical operations, NumPy's array structures simplify complex mathematical computations, which are critical when handling large datasets like stock price histories.

d) **Matplotlib and Seaborn**: These libraries were used for visualizing the relationships between stock features and trends. Data visualization helped identify patterns in stock price movement and highlight potential outliers or anomalies.

e) **Scikit-learn:** A comprehensive machine learning library that provided the algorithms used for stock price prediction. Scikit-learn's functionality includes model training, evaluation, and tuning through built-in methods.

f) **Jupyter Notebook:** An interactive development environment where code could be written, tested, and refined in real-time. The notebook format allowed for a combination of code, visualizations, and annotations, providing an ideal platform for experimentation.

3.2 Technology

The core machine learning models utilized in this project are listed below, each offering unique strengths in predicting stock price movements based on historical data.

3.2.1 Machine Learning Algorithms

a) **K-Nearest Neighbors (KNN):** A non-parametric algorithm that bases predictions on the distance between data points. KNN is sensitive to the choice of 'k' and distance metric, but it works well for capturing local patterns in stock price movements.

b) **Decision Tree:** A simple but effective algorithm that uses feature-based splits to model stock price movements. Decision Trees work well on smaller datasets and provide interpretability, but they can overfit without regularization.

c) **Random Forest:** An ensemble learning method that improves upon Decision Trees by creating multiple trees and averaging their predictions. Random Forest is robust to overfitting and offers higher accuracy by reducing variance through bootstrapped samples.

d) **Gradient Boosting:** A sequential ensemble method that builds models iteratively, with each new model focusing on the errors made by the previous one. Gradient Boosting is highly accurate but requires careful tuning to avoid overfitting.

e) **Linear Regression:** A straightforward method for modeling the relationship between independent variables (features like 'Open', 'High', 'Low') and the dependent variable (closing price). Linear Regression serves as a baseline model, providing interpretable results with a simple linear equation.

3.2.2 Data Preprocessing and Feature Engineering

Data preprocessing included managing missing values and applying normalization techniques. Feature selection ensured that only relevant variables like 'Open', 'High', 'Low', 'Adj Close', and volume were used for model training. Techniques like correlation analysis and feature scaling were applied to improve model performance.

3.2.3 Hyperparameter Tuning

Each model was fine-tuned using hyperparameter optimization techniques, such as GridSearchCV. This ensured that the models performed optimally, especially when selecting the number of estimators in Random Forest or the learning rate in Gradient Boosting.

The combination of these tools and machine learning techniques provided the foundation for building a robust, predictive model that can offer valuable insights into the stock price movements of Axis Bank. Through careful evaluation and tuning, the project demonstrates how AI can enhance financial forecasting, improving decision-making in stock markets.

CHAPTER 4

DATA PREPARATION

1. Loading the Dataset

The dataset, consisting of historical stock data for Axis Bank, was imported from a CSV file. This data serves as the foundation for regression analysis using five machine learning algorithms: Linear Regression, Gradient Boosting, KNN, Decision Tree, and Random Forest Regression.

We used the Pandas library to load and manipulate the dataset efficiently. This allowed us to quickly examine the data's structure and perform various preprocessing steps.

- **import pandas as pd**

➢ data = pd.read_csv('AXISBANK.NS.csv')

2. Data Exploration

Exploratory data analysis (EDA) is critical in understanding the dataset and revealing any patterns, trends, or irregularities. The initial step was to check for missing values, data types, and the overall structure of the data:

 # Check basic information of the dataset

➢ data.info()

 # Display the first few rows of the dataset

➢ data.head()

Next, we used Matplotlib and Seaborn for data visualization to better understand the distribution of key features like stock prices and volume:

- import matplotlib.pyplot as plt
- import seaborn as sns

 # Plot the distribution of the closing price

➢ plt.figure(figsize=(10,6))

sns.histplot(data['Close'], kde=True)

plt.title('Distribution of Closing Price')

plt.xlabel('Closing Price')

plt.ylabel('Frequency')

plt.show()

 # Line plot for closing prices over time

- plt.figure(figsize=(12,6))

  plt.plot(data['Date'], data['Close'], label='Closing Price')

  plt.title('Closing Price Over Time')

  plt.xlabel('Date')

  plt.ylabel('Closing Price')

  plt.legend()

  plt.show()

These visualizations allowed us to identify potential trends in the stock data, such as upward or downward movements, and to observe any outliers.

3. Handling Missing Data

Missing data can lead to biased or incorrect model predictions, so it's important to handle these appropriately. We identified missing values in columns like [insert columns if relevant] and handled them using techniques such as:

- **Forward/Backward Fill**: Filling missing values based on nearby data.
- **Mean/Median Imputation**: For numerical columns like price and volume, missing values were filled with the column's mean or median values.

 # Filling missing values using forward fill

- data.fillna(method='ffill', inplace=True)

4. Feature Selection

Feature selection is a crucial step in ensuring that the model uses only the most relevant data. The dataset contained several important features, including:

- **Date**: The trading date.
- **Open**: The opening stock price.
- **Close**: The closing stock price (our target variable).

- **High/Low**: The highest and lowest prices during the day.
- **Volume**: The volume of stocks traded.

We removed irrelevant columns (such as unnecessary IDs or non-numerical data) to focus on numerical features.

 # Select important features for the regression task

features = ['Open', 'Close', 'High', 'Low', 'Volume']

X = data[features]

y = data['Close'] # Target variable

5. Data Transformation

To ensure consistency across features with different scales, we applied feature scaling using the StandardScaler from the scikit-learn library. This step was especially important for algorithms sensitive to feature scaling, such as KNN and Gradient Boosting.

- **from sklearn.preprocessing import StandardScaler**

  scaler = StandardScaler()

  X_scaled = scaler.fit_transform(X)

6. Train-Test Split

The dataset was divided into training and testing sets, typically in an 80:20 ratio. This step ensures that our models are trained on a portion of the data and evaluated on unseen data to measure performance accurately.

- **from sklearn.model_selection import train_test_split**

  X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

7. Visualization of Key Features

Visualizing key features helped to further understand the relationships between different attributes. For instance, plotting correlations between features helped to identify any multicollinearity that might affect the regression models:

 # Correlation heatmap

plt.figure(figsize=(10,6))

```
sns.heatmap(data[features].corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Heatmap')
plt.show()
```

This heatmap provided insights into how closely features like "Open," "Close," "High," and "Low" prices were correlated.

CHAPTER 5

EXPERIMENTATION

In this chapter, we present the experimental setup, results, and comparative analysis of multiple machine learning algorithms applied to stock market prediction. Predicting stock market trends is a complex and challenging task due to the highly volatile nature of the market, influenced by countless economic, political, and social factors. To address this, we explore a diverse set of algorithms, each known for its strength in different types of data patterns and predictive modelling.

The primary objective of this chapter is to assess the performance of several widely used algorithms - K-Nearest Neighbors, Decision Tree, Random Forest, Gradient Boosting and Linear Regression on the task of predicting stock prices or stock index movements. Each algorithm's results will be evaluated based on key performance metrics such as accuracy, mean squared error (MSE), and prediction precision.

The section begins by discussing the dataset used for the experiments, including data preprocessing techniques such as normalization, feature selection, and data splitting. Subsequently, we outline the specific configurations and hyperparameters optimized for each algorithm to ensure the most accurate predictions.

5.1 K-Nearest Neighbour Algorithm

This Topic introduces the K-Nearest Neighbours(K-NN) algorithm, a fundamental supervised machine learning technique used for classification and regression tasks. It explores the origins, development, and applications of K-NN, emphasizing its simplicity, versatility, and effectiveness in various fields such as image recognition, text classification, and stock market prediction. The Topic also discusses the algorithm's benefits and challenges, providing a foundation for understanding how K-NN can be applied to real-world data problems.

The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning method employed to tackle classification and regression problems. Evelyn Fix and Joseph Hodges developed this algorithm in 1951, which was subsequently expanded by Thomas Cover.

5.1.1 History of K-NN

Evelyn Fix and Joseph Hodges created kNN for the first time in 1951 while conducting research for the US military. They released a paper outlining the non-parametric classification technique known as discriminant analysis. Thomas Cover and Peter Hart published their "Nearest

Neighbor Pattern Classification" paper in 1967, which further developed the non-parametric classification technique. James Keller improved the method over two decades later by creating a "fuzzy KNN" that results in reduced error rates.

a. Origins in Statistical Estimation (1951)

The conceptual foundation of KNN dates back to 1951 when Evelyn Fix and Joseph Hodges introduced the Fix and Hodges estimator in their paper titled "Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties." They proposed a method for pattern classification based on proximity to a given point without assuming any specific distribution for the data. This work laid the groundwork for what would later become the KNN algorithm.

b. Further Development (1967)

The KNN algorithm was formalized and popularized by Thomas Cover and Peter Hart in their influential 1967 paper, "Nearest Neighbour Pattern Classification." In this paper, Cover and Hart explored the properties of nearest neighbour classification and provided mathematical proofs for the algorithm's performance, including its convergence to the Bayes error rate (the minimum possible error rate for a classifier) as the number of neighbours and data points increase.

c. KNN in Machine Learning

KNN was one of the early algorithms used in machine learning and pattern recognition. Its simplicity and effectiveness made it a popular choice for early classification tasks. Despite its simplicity, KNN remained relevant over the years due to its adaptability and strong theoretical foundation.

d. Expansion to Regression (Late 20th Century)

Initially, KNN was primarily used for classification tasks. However, it was later adapted for regression tasks, where the predicted value is calculated as the average (or median) of the values of the nearest neighbours. This adaptation allowed KNN to be used in a broader range of applications, including stock market prediction, image recognition, and more.

e. Modern Usage and Enhancements

In recent years, KNN has continued to be a useful tool, particularly in situations where the data is complex or the relationships between variables are non-linear. However, KNN has some limitations, such as sensitivity to the choice of 'k' and computational inefficiency with large datasets. Modern machine learning has seen the development of various optimizations and

extensions of KNN, including the use of KD-trees and Ball-trees for faster neighbour searches, and ensemble methods that combine KNN with other algorithms [15].

5.1.3 What is K-NN?

The kNN algorithm is the most popular algorithm in use today because it can be used to a wide range of industries, including banking, genomics, and customer service. The K-Nearest Neighbours (KNN) is a supervised machine learning technique which is used to solve regression and classification issues. From genetics to Netflix suggestions, this supervised machine learning technique is one of the most popular since it is straightforward and simple to apply. One of the most fundamental yet important machine learning classification methods is KNN. It is heavily used in pattern recognition, data mining, and intrusion detection and is a member of the supervised learning domain. Since it is non-parametric, which means it does not make any underlying assumptions about the distribution of data (unlike other algorithms like GMM, which assume a Gaussian distribution of the given data), it is extensively applicable in real-life circumstances. An attribute-based prior data set (also known as training data) is provided below, allowing to classify coordinates into groups. As an example, consider the following table of data points containing two features:

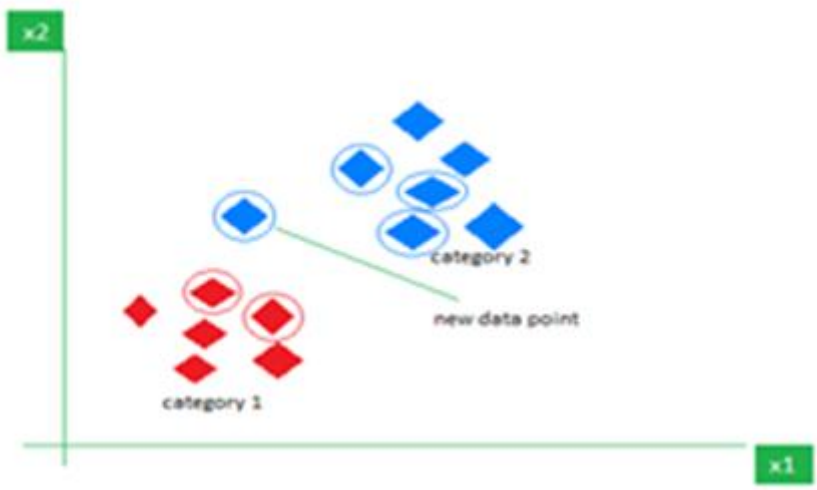


Fig 5.1.1 Data points containing two features

Now, assign these points to a group by examining the training set, given a different set of data points (also known as testing data). Keep in mind that the unclassified locations have a "White" designation. These points can help us find certain clusters or groups if we plot them on a graph. Now that a point is unclassified, we can assign it to a group by looking at which group its

closest neighbors are in. Accordingly, there is a greater chance that a point near a group of points designated as "Red" will also be classed as "Red."

It seems obvious to us that the first point (2.5, 7) belongs in the "Green" category, whereas the second point (5.5, 4.5) belongs in the "Red" category [16].

5.1.4 Why do we need K-NN?

Because of its simplicity and ease of use, the (K-NN) method is a popular and adaptable machine learning technique. No presumptions on the distribution of the underlying data are necessary. It is a versatile option for a range of dataset types in classification and regression applications because it can handle both numerical and categorical data. This non-parametric technique bases its predictions on how similar the data points in a particular dataset are to one another. By comparison, K-NN is less susceptible to outliers than other algorithms. The K-NN algorithm locates the K closest neighbors, using a distance metric like Euclidean distance, to a given data point. The majority vote or the average of the K neighbors are then used to establish the class or value of the data item. Using this method enables the algorithm to anticipate outcomes based on the local structure of the data and adjust to various patterns [17].

5.1.5 Distance Metrics Used in KNN Algorithm

As it is well known, the KNN algorithm aids in locating the groups or closest points to a query point. However, we require a measure in order to identify the closest points or groups for a given query point. We employ the following distance measurements for this purpose: **The Euclidean Distance**

The cartesian distance between the two locations in the plane or hyperplane is all that this represents. Another way to represent Euclidean distance is as the length of the straight line connecting the two points under investigation. This metric aids in the computation of the net displacement that an object undergoes between its two states.

Equation: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

The Euclidean distance is also known as the Pythagorean distance because it can be calculated using the Pythagorean theorem and the Cartesian coordinates of the points [18].

5.1.6 Choosing the best value of k

When defining the number of neighbors in the KNN algorithm, the value of k is highly important. In the k-nearest neighbors (k-NN) procedure, the input data should be used to

determine the value of k. A greater value of k would be preferable if the input data contained more noise or outliers. Selecting an odd value for k is advised in order to prevent ties in the classification. The best k value for the dataset can be chosen with the use of cross-validation techniques**.** The K-NN algorithm exhibits a jump in prediction values as a result of an input unit change. This is because the neighbors have changed. We can use the algorithm's weighting of neighbors to manage this scenario. We want less influence from a neighbor if the distance between them is great [19].

5.1.7 Working of K-NN

Based on the similarity principle, the K-Nearest Neighbors (KNN) method predicts the label or value of a new data point by taking into account the labels or values of its K nearest neighbors in the training dataset.

Below is a detailed breakdown of how KNN functions step-by-step:

Step 1: Determining K's ideal value

K is the number of nearby neighbors that must be taken into account while making a prediction.

Step 2: Distance calculation

The Euclidean distance is used to calculate how similar the target and training data points are to each other. Every data point in the dataset has its distance from the target point estimated.

Step 3: Locate the Closest Neighbors

The closest neighbors are the k data points that have the least distances to the target point.

Step 4: Selecting a Classification or Regression Average

Majority voting is used in the classification problem to decide the K-nearest neighbors class labels. The predicted class for the target data point is the one with the highest frequency among the neighbors.

The class label in the regression issue is determined by averaging the K nearest neighbor’s goal values. The anticipated result for the desired data point is the computed average value [20].

5.1.8 Benefits of KNN Algorithm

- **Simple**: Because of the algorithm's low complexity, it is simple to implement.

- **Easily Adapts:** The KNN algorithm operates by storing all of the data in memory storage. As a result, whenever a new example or data point is supplied, the algorithm automatically modifies itself to take into account the new example and contributes to the future forecasts.
- **Few Hyper parameters:** The value of k and the distance metric we want to select from our evaluation metric are the only parameters needed for the training of a KNN algorithm.

5.1.9 Challenges of K-NN Algorithm

- **Scaling:** Scaling is challenging because kNN raises storage costs due to its large memory and data storage requirements. The approach is computationally demanding, which requires a lot of resources, due to its dependency on memory.
- **Curse of Dimensionality:** This describes a phenomenon that arises in computer science when an increasing number of dimensions and the resulting increase in feature values in these dimensions pose a challenge to a fixed collection of training examples. Stated differently, the training data of the model is not able to keep up with the hyperspace's increasing dimensionality. when a result, forecasts lose accuracy when the gap widens on different dimensions between the query point and comparable points.
    - **Overfitting:** The behaviour of the algorithm is dependent on the value of k. This is more likely to occur when k is too low. Higher values of k will "smooth" the prediction values since the method averages values over a larger region, whereas lower values of k may over fit the data [21].

5.1.10 Use of K-NN in real life

a. Images recognition

To recognize comparable images based on their features—such as colour, shape and texture—KNN can be utilized in image recognition. Applications like recommendation engines, security systems, and search engines may find use for this. In order to recognize images, the KNN method calculates the distance between an input image's features and the features of images in a dataset. The KNN algorithm finds the K closest images with the least distance by calculating the distance between each pixel value in the input image and each pixel value in the dataset. Next, the resulting image is categorized using the K nearest neighbors most common class. The KNN algorithm, for instance, can be used in a facial recognition system to compare an individual's face features in an input image with the features of faces in a database of known people. The algorithm determines the K closest matches by calculating the distances between the input image's pixel values and the database's image values. The system then makes a prediction about the person's identification in the input image based on the classes of the K

closest matches. Applications for KNN-based image recognition are numerous and include object and face recognition as well as image classification. It is employed in a number of sectors, including healthcare, entertainment, and security.

b. Text Classification

Using similarity in content and structure, KNN can be applied to text classification to group documents into distinct subjects or themes. Another use for the KNN (K-Nearest Neighbors) method is text classification. It functions in this way: it uses the labels of a new document's k nearest neighbors in the training data set to determine the category or label to apply to it. We must first encode the text data in a numerical representation before we can utilize KNN for text categorization. Methods like bag-of-words and TF-IDF (Term Frequency-Inverse Document Frequency) can be used for this. After the text data is numerically represented, we can classify new documents using the KNN algorithm.

c. Environment Monitoring

In environmental monitoring, KNN can be used to categorize data about the quality of the air, water, and soil as well as to spot trends and patterns over time. A supervised machine learning technique called K-nearest neighbors (KNN) can be used in environmental monitoring to categorize environmental data according to its attributes. For instance, KNN can be used to categorize data on water quality into groups such as "excellent," "good," "fair," and "poor" according to variables like turbidity, pH, and dissolved oxygen. Using a distance metric like Euclidean distance, the KNN technique locates the K data points that are closest to a new data point. The new data point is then classified using the majority class of the K-NN.

d. Sports analytics

KNN is a useful tool for classifying players according to performance indicators like assists, rebounds, and scoring as well as for identifying comparable players based on skill and playing style. Coaches and teams can use this to enhance player recruitment and game strategy. Sports analytics can employ the machine learning method KNN, or k-nearest neighbors, to categorize players according to their performance data and find comparable players. The method finds the k-nearest neighbors based on a distance metric, like Euclidean distance, by comparing a player's performance metrics to those of other players in a dataset [22].

5.1.11 How is K-NN used to predict stock market prices

The buying, selling, and issuing of shares of publicly traded corporations are collectively referred to as the stock market. Predicting the stock market is essential for wise and profitable

investing. Some people believe that predicting future stock values in the stock market is difficult and fraught with risk, including the potential for enormous losses. A machine learning model-based stock market forecast utilizing the K Nearest Neighbor (KNN) algorithm is suggested as a solution to these problems. Because KNN can process relationships between the numerical data, it is particularly effective in numerical prediction problems for predicting changes in stock value the following day.

K-Nearest Neighbors (K-NN) can be used to predict stock market prices by leveraging historical data to find patterns that are similar to current market conditions. Here's how the process generally works:

Step 1: Data Collection: Collect historical stock price data, which could include features like the stock's closing price, opening price, volume, high and low prices, moving averages, and other relevant indicators.

Step 2: Data Pre-processing: Normalize or standardize the data, as KNN is sensitive to the scale of the data. Split the data into training and testing sets.

Step 3: Feature Selection: Identify the key features that will be used for prediction. These could include technical indicators (e.g., moving averages, RSI, MACD) or other relevant financial data.

Step 4. Choosing K and Distance Metric: Select the value of (K) (number of neighbors to consider). This can be tuned using techniques like GridSearchCV. Choose an appropriate distance metric, often Euclidean distance is used.

Step 5: Training the Model: For each data point in the testing set, K-NN identifies the (K) nearest neighbors in the training set based on the selected features.

Step 6: Prediction: The stock price prediction is usually done by averaging the target values (e.g., the stock price) of the (K) nearest neighbors. For classification-based predictions (like predicting whether the price will go up or down), a majority voting mechanism might be used.

Step 7: Evaluation: Evaluate the model's performance using metrics like Mean Squared Error (MSE) for regression tasks or accuracy for classification tasks.

Step 8: Optimization: Tune the model by adjusting the value of (K), distance metric and possibly the features used to improve accuracy.

5.1.12 About Dataset

We analyse the stock market data for Axis Bank over the last three months, which was downloaded from Yahoo Finance. The dataset provides comprehensive information on the daily trading activity, including key financial indicators such as the opening price, closing price, highest price, lowest price, adjusted closing price, and trading volume. The opening price reflects the stock's initial value at the start of each trading day, while the closing price captures the final value at the end of the trading day. The high and low prices indicate the peak and minimum values reached during the trading session, offering insights into the stock's volatility. The adjusted closing price accounts for corporate actions such as dividends and stock splits, providing a more accurate reflection of the stock's value. Lastly, the volume represents the total number of shares traded, highlighting the stock's liquidity and investor interest. This dataset serves as a valuable resource for understanding Axis Bank's recent market performance and can be used for further analysis, such as trend identification and predictive modelling.

5.1.13 Code and its Explanation

a) Importing Necessary Libraries

```
[1]: import pandas as pd
     import numpy as np
     import os
     import matplotlib.pyplot as plt
```

- **pandas:** A powerful data manipulation library used to create DataFrames, which are 2D labelled data structures similar to Excel spreadsheets. It is especially useful for loading, cleaning, transforming, and analysing structured data.
- **numpy:** A fundamental package for scientific computing in Python. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- **os:** Provides a way to interact with the operating system, allowing you to handle files, directories, and paths in a platform-independent manner.
- **matplotlib.pyplot:** A plotting library that provides an interface similar to MATLAB for creating static, animated, and interactive visualizations in Python. It's commonly used for simple and complex plots, such as line plots, scatter plots, histograms, etc.

b) Loading and Cleaning the Data

```
file_path = r'C:\Users\HP\Downloads\AXISBANK.NS.csv'
```

[2]:
```
data = pd.read_csv(file_path)
```

[3]:
```
data.dropna(inplace=True)
```

- **file_path:** Defines the file path where the CSV file containing the stock data is located. The r' before the string indicates a raw string, meaning backslashes are treated literally
- **pd.read_csv(file_path):** Reads the CSV file at the specified path into a pandas DataFrame named data. The DataFrame structure allows you to easily manipulate and analyse the data.
- **data.dropna(inplace=True):** Removes any rows from the DataFrame that contain missing values (NaN). This ensures that all the data used for analysis is

[4]:
```
data.head(63)
```

[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2024-05-03	1150.900024	1163.250000	1134.099976	1141.500000	1140.619751	9993632
1	2024-05-06	1143.900024	1156.000000	1135.650024	1143.650024	1142.768066	7490104
2	2024-05-07	1146.050049	1148.800049	1124.050049	1127.699951	1126.830322	7115509
3	2024-05-08	1124.650024	1135.449951	1118.250000	1128.650024	1127.779663	8175609
4	2024-05-09	1122.000000	1134.550049	1112.199951	1115.650024	1114.789673	5226693
...	...	...	...	...	...	...	...
58	2024-07-29	1182.349976	1194.599976	1163.650024	1170.050049	1170.050049	19598573
59	2024-07-30	1164.099976	1180.000000	1160.199951	1170.000000	1170.000000	18090045
60	2024-07-31	1157.000000	1171.000000	1154.000000	1166.099976	1166.099976	17347076
61	2024-08-01	1168.000000	1175.650024	1165.449951	1172.300049	1172.300049	11190500
62	2024-08-02	1164.000000	1167.800049	1156.099976	1160.849976	1160.849976	10581445

63 rows × 7 columns

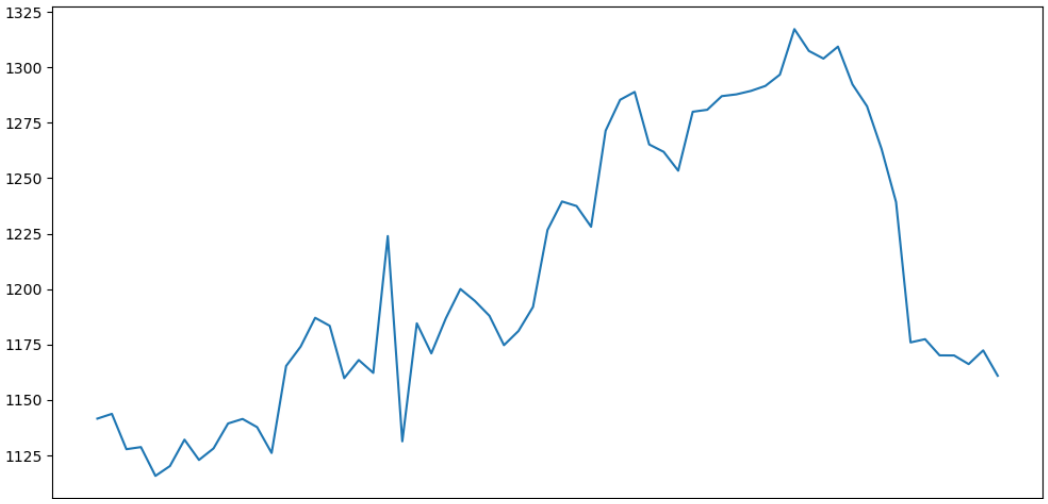
complete. The inplace=True argument modifies the DataFrame in place, so no new DataFrame is created.

- **data.head(63):** Displays the first 63 rows of the DataFrame to give a preview of the data. This is often done to verify that the data has been loaded correctly and to understand its structure.

c) Plotting the Closing Price

```
[5]: plt.figure(figsize=(12,6))
     plt.plot(data['Close'], label='Closing Price')
```

```
[5]: [<matplotlib.lines.Line2D at 0x1e8eb137010>]
```



- **plt.figure(figsize=(12, 6)):** Sets up a new figure with a size of 12 inches by 6 inches. This ensures the plot has a suitable size for viewing, especially if it will be displayed in a report or presentation.
- **plt.plot(data['Close'], label='Closing Price'):** Creates a line plot of the 'Close' column from the DataFrame. The 'Close' price represents the price of the stock at the end of the trading day. By plotting this, you can visualize how the stock price has changed over time.

d) Defining Features (X) and Target (y)

```
[6]: # Features (X) and target (y)
     X = data[['Open', 'High', 'Low','Adj Close']]
     Y = data['Close']
```

- **X:** This variable represents the feature set used to predict the target. It is a DataFrame containing the columns 'Open', 'High', 'Low' and 'Adj Close':

  'Open': The stock's opening price at the start of the trading day.

  'High': The highest price the stock reached during the trading day.

  'Low': The lowest price the stock reached during the trading day.

  'Adj Close': The stock's closing price adjusted for corporate actions such as dividends, stock splits, etc.
- **Y:** This variable represents the target variable you want to predict, which is the 'Close' price, or the stock's price at the end of the trading day.

e) Splitting the Data into Training and Testing Sets

```
[7]: from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=45)
```

- **train_test_split:** A function from sklearn.model_selection used to split the dataset into two parts: a training set and a testing set. The training set is used to train the model, and the testing set is used to evaluate its performance.
- **X_train, X_test:** These variables hold the training and testing subsets of the features (X). X_train is used to train the model, while X_test is used to test it.
- **y_train, y_test:** These variables hold the training and testing subsets of the target variable (Y). y_train is used in training the model, and y_test is used for evaluation.
- **test_size=0.20:** Specifies that 20% of the data will be used for testing, and 80% will be used for training.
- **random_state=45:** Sets a seed for the random number generator to ensure the split is reproducible. This means that every time you run the code with the same data and random_state, you'll get the same split.

f) Scaling the Features

```
[8]: from sklearn.neighbors import KNeighborsRegressor
     from sklearn import neighbors
     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

- **StandardScaler:** A pre-processing tool from sklearn that standardizes features by removing the mean and scaling them to unit variance. This is important for algorithms like KNN, which are sensitive to the scale of the data.
- **scaler.fit_transform(X_train):** Fits the StandardScaler on the training data (calculates the mean and standard deviation) and then transforms the data according to this scaling. This ensures that the training data has a mean of 0 and a standard deviation of 1.
- **scaler.transform(X_test):** Transforms the test data using the scaling parameters (mean and standard deviation) calculated from the training data. This ensures that both the training and test data are on the same scale.

g) Hyper parameter Tuning using GridSearchCV

```
[9]: from sklearn.model_selection import  GridSearchCV
     params = {'n_neighbors': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}
     knn = neighbors.KNeighborsRegressor()
     model = GridSearchCV(knn, params,cv=5)
```

```
[10]: model.fit(X_train_scaled, y_train)
      best_params = model.best_params_
      print(f"Best k: {best_params}")
```

```
Best k: {'n_neighbors': 3}
```

- **GridSearchCV:** A tool for performing an exhaustive search over a specified parameter grid. It allows you to automatically find the best hyper parameters for your model based on cross-validation performance.
- **params:** A dictionary specifying the hyper parameter grid to search. Here, n_neighbors (the number of neighbors considered in KNN) ranges from 1 to 15.
- **KNeighborsRegressor:** Initializes a K-Nearest Neighbors regressor, a type of machine learning algorithm that predicts the target based on the average of the nearest neighbors.
- **GridSearchCV (knn, params, cv=5):** Sets up the grid search to find the best n_neighbors value using 5-fold cross-validation.
- **model.fit (X_train_scaled, y_train):** Fits the grid search model to the training data. This process tests all values of n_neighbors specified in the grid and determines the best one.
- **best_params = model.best_params_:** Retrieves the best parameters found by the grid search.
- **best_k = best_params['n_neighbors']:** Extracts the best value of n_neighbors (k) from the grid search results.
- print(f'Best k: {best_k}"): Outputs the best value of n_neighbors.

h) Training the KNN Regressor with the Best k

```
[11]: # Training the KNN regressor
      knn = KNeighborsRegressor(n_neighbors=3)
      knn.fit(X_train_scaled, y_train)
```

```
[11]: KNeighborsRegressor
      KNeighborsRegressor(n_neighbors=3)
```

- **KNeighborsRegressor(n_neighbors=3):** Initializes the KNN regressor using the best value of n_neighbors found during the grid search, which is 3 in this case.
- **knn.fit(X_train_scaled, y_train):** Trains the KNN model on the scaled training data. The model now learns the relationship between the features and the target using the training data.

i) Evaluating Training Set Performance

[12]:
```
from sklearn.metrics import r2_score
# Predicting on the training set
y_train_pred = knn.predict(X_train_scaled)

# Calculating training accuracy score
r2_train = r2_score(y_train, y_train_pred)
train_accuracy_percentage = r2_train * 100
print(f"R-squared Score on training set: {r2_train:.4f}")
print(f"Training Accuracy: {train_accuracy_percentage:.2f}%")
```

```
R-squared Score on training set: 0.9855
Training Accuracy: 98.55%
```

- **knn.predict(X_train_scaled):** Uses the trained KNN model to predict the target values (Y) for the training data. This is done after the model has been trained (knn.fit(X_train_scaled, y_train)).
- **y_train_pred:** Stores the predicted values for the training data. These are the values the model estimates based on the features X_train_scaled.
- **r2_score(y_train, y_train_pred):** Computes the R-squared score (coefficient of determination) for the training set. This metric measures how well the model's predictions match the actual target values. It ranges from 0 to 1, where 1 indicates perfect prediction and 0 indicates no predictive power.
- **r2_train:** Stores the R-squared score for the training data. This value reflects the proportion of variance in the target variable that is explained by the features used in the model.
- **train_accuracy_percentage:** Converts the R-squared score to a percentage format. This makes it easier to interpret the model's performance in terms of percentage accuracy.
- **print(f"R-squared Score on training set: {r2_train:.4f}"):** Prints the R-squared score for the training set, formatted to four decimal places. This provides a measure of how well the model has learned from the training data.

- **print(f"Training Accuracy: {train_accuracy_percentage:.2f}%"):** Prints the R-squared score converted to a percentage, formatted to two decimal places. This provides a clearer view of the model's performance in percentage terms.

j) Predicting and Evaluating the Model

[13]:
```
from sklearn.metrics import r2_score
# Predicting on the test set
y_pred = knn.predict(X_test_scaled)
# Calculating test accuracy score
r2 = r2_score(y_test, y_pred)
accuracy_percentage = r2 * 100
```

[14]:
```
print(f"R-squared Score on test set: {r2:.4f}")
print(f"Test Accuracy: {accuracy_percentage:.2f}%")
```

```
R-squared Score on test set: 0.9951
Test Accuracy: 99.51%
```

- **y_pred:** Generates predictions on the scaled test data using the trained KNN model.
- **r2_score:** A metric that measures the proportion of variance in the dependent variable that is predictable from the independent variables. It provides a measure of how well the predictions match the actual values. An R-squared score of 1 indicates perfect prediction.
- **accuracy_percentage = r2 * 100:** Converts the R-squared score into a percentage to interpret it as the accuracy of the model.
- **print (f"R-squared Score on test set: {r2:.4f}"):** Outputs the R-squared score, showing how well the model performed on the test set.
- **print(f"Test Accuracy: {accuracy_percentage:.2f}%"):** Outputs the test accuracy as a percentage.

k) Predicting the Next Day's Value

[15]:
```
# Extract the latest data point
latest_data = X.iloc[-1:,].values
latest_data_scaled = scaler.transform(latest_data)

# Predict the next day's value
next_day_prediction = knn.predict(latest_data_scaled)

print("Predicted value for the next day:", next_day_prediction[0])
```

```
Predicted value for the next day: 1168.2833253333333
```

- **latest_data = X.iloc[-1:,].values:** Extracts the most recent (last) row from the feature set X, which represents the latest available data for making predictions. iloc[-1:,] selects the last row, and .values converts it to a NumPy array.
- **latest_data_scaled = scaler.transform(latest_data):** Scales the latest data point using the same scaler that was fitted on the training data, ensuring consistency in the data's scale.
- **next_day_prediction = knn.predict(latest_data_scaled):** Predicts the target value (closing price) for the next day based on the latest scaled data.
- **print("Predicted value for the next day:", next_day_prediction[0]):** Outputs the predicted closing price for the next day.

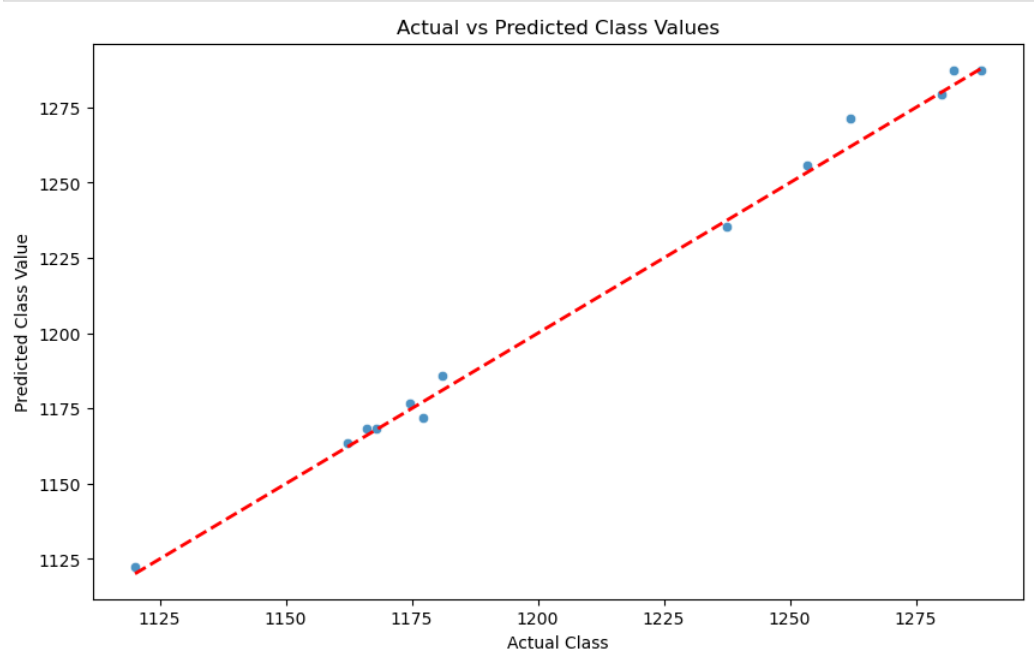
```
[16]: from sklearn.metrics import mean_squared_error, mean_absolute_error
      mse = mean_squared_error(y_test, y_pred)
      mae = mean_absolute_error(y_test, y_pred)
      print(f"KNN Mean Squared Error: {mse:.2f}")
      print(f"KNN Mean Absolute Error: {mae:.2f}")

      KNN Mean Squared Error: 14.46
      KNN Mean Absolute Error: 2.93
```

l) Calculating and Displaying Errors

- **mean_squared_error:** Calculates the Mean Squared Error (MSE), which is the average of the squares of the errors (differences between actual and predicted values). It penalizes larger errors more than smaller ones.
- **mean_absolute_error:** Calculates the Mean Absolute Error (MAE), which is the average of the absolute differences between actual and predicted values. It gives an idea of the magnitude of errors in the predictions.
- **print (f"KNN Mean Squared Error: {mse:.2f}"):** Outputs the MSE, giving an indication of the overall error in the predictions.
- **print (f"KNN Mean Absolute Error: {mae:.2f}"):** Outputs the MAE, providing a sense of how far off the predictions are from the actual values, on average.

m) Visualizing Actual vs. Predicted Values

[17]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create the DataFrame
done = pd.DataFrame({'Actual Class': y_test, 'Predicted Class value': y_pred})

# Set up the matplotlib figure
plt.figure(figsize=(10, 6))

# Create a scatter plot
sns.scatterplot(data=done, x='Actual Class', y='Predicted Class value', alpha=0.8)

# Add a line for perfect prediction
plt.plot([done['Actual Class'].min(), done['Actual Class'].max()],
         [done['Actual Class'].min(), done['Actual Class'].max()],
         color='red', linestyle='--', linewidth=2)

# Add labels and title
plt.xlabel('Actual Class')
plt.ylabel('Predicted Class Value')
plt.title('Actual vs Predicted Class Values')

# Show plot
plt.show()
```



- **seaborn:** A Python data visualization library built on top of matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **sns.scatterplot:** Creates a scatter plot where each point represents an actual-predicted pair. The x-axis shows the actual values, and the y-axis shows the

predicted values. If the predictions are perfect, all points would lie on the line y=x.

- **plt.plot:** Adds a diagonal line to the plot (red, dashed line) that represents perfect predictions (where the actual value equals the predicted value).
- **plt.xlabel('Actual Class'):** Labels the x-axis as "Actual Class," which refers to the actual values of the target variable.
- **plt.ylabel('Predicted Class Value'):** Labels the y-axis as "Predicted Class Value," which refers to the values predicted by the model.
- **plt.title('Actual vs Predicted Class Values'):** Adds a title to the plot to describe what is being visualized.
- **plt.show():** Displays the plot, allowing you to visually assess how well the model's predictions match the actual values.

These components work together to load, pre-process, train and evaluate a KNN model for predicting stock prices based on historical data.

5.1.14 Mathematics Behind KNN

K-Nearest Neighbours (KNN) algorithm is a simple, intuitive method that can be used for both classification and regression tasks. In the context of predicting the closing price of a stock for the next day (regression) the mathematics behind KNN involves the following steps:

a) **Feature Space and Distance Calculation**

Feature Space: Each stock data point (or day) is represented as a point in a multidimensional feature space, where each dimension corresponds to one of the features (e.g., 'Open', 'High', 'Low', 'Adj Close').

Distance Calculation: To make a prediction, the algorithm first needs to identify the "neighbours" of the target day. It does this by calculating the distance between the target data point (e.g., the latest day's data) and all other points in the dataset. The most commonly used distance metric is the Euclidean distance.

The Euclidean distance d between two points A = (a_1, a_2 .....a_n) & B = (b_1, b_2 …..b_n) in n-dimensional space is given by

$$d(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

In the case of predicting the closing price, the points from A to B represent two different days, and the features a_1, a_2 .....a_n and b_1, b_2 ….b_n correspond to the 'Open', 'High', 'Low' and 'Adj Close' prices for those days.

b) **Identifying Neighbours**

After calculating the distances, the algorithm identifies the *k* closest data points (i.e., the k smallest distances) to the target day. These are the "nearest neighbours".

k: The number of neighbours considered is a hyper parameter and can be chosen based on cross-validation or domain knowledge. In this case, k=3 was found to be optimal through grid search.

c) **Predicting the Closing Price**

Averaging Neighbour Values: In KNN regression, the predicted value (in this case, the closing price for the next day) is typically the average of the target values (closing prices) of the k

d) **Making Predictions**

Next Day Prediction: Once the average closing price of the nearest neighbours is computed, it is used as the predicted closing price for the next day. nearest neighbours.

5.2 Decision Tree Algorithm

5.2.1 What is Decision Tree Algorithm

A decision tree is a popular machine learning algorithm used for both classification and regression tasks. It works by breaking down a dataset into smaller subsets while at the same time developing an associated decision tree incrementally. The process involves:

a) **Choosing the Best Split:** At each node in the tree, the algorithm chooses the best feature to split the data based on a certain criterion (like Gini impurity, entropy for classification, or mean squared error for regression).

b) **Recursive Splitting:** This splitting process is repeated recursively for each child node until a stopping criterion is met. The stopping criteria can include a maximum depth of the tree, a minimum number of samples per leaf, or no further improvement in the split.

c) **Prediction:** For classification tasks, the final decision is made based on the majority class of samples in the leaf node. For regression, it is the average of the target values of the samples in the leaf node.

Pros:

- Easy to understand and interpret.
- Requires little data preprocessing.
- Can handle both numerical and categorical data.

Cons:

- Can easily over fit the data if the tree is too deep.
- May not perform well with very large datasets or datasets with many features.

Decision trees are often used as building blocks for more complex ensemble methods like Random Forests and Gradient Boosted Trees, which aim to improve performance and reduce over fitting [24].

5.2.2 History

The decision tree algorithm has a rich history in both theory and application. Here's a brief overview:

a) **Early Concepts (1960s-1970s):**The foundational ideas for decision trees began in the 1960s. Early research focused on the theory of decision-making and classification. The concept of splitting data based on certain criteria was explored in various contexts.

b) **ID3 Algorithm (1986):**

The first practical decision tree algorithm was developed by Ross Quinlan and introduced as the ID3 (Iterative Dichotomiser 3) algorithm. ID3 uses entropy and information gain to determine the best features for splitting the data.

c) **C4.5 Algorithm (1993):**

Quinlan later introduced C4.5, an improved version of ID3. C4.5 addressed some limitations of ID3, such as handling missing values and pruning the tree to prevent overfitting. C4.5 became one of the most popular algorithms for decision trees.

d) **CART Algorithm (1986):**

Around the same time as ID3, the Classification and Regression Trees (CART) algorithm was developed by Breiman, Friedman, Olshen, and Stone. CART uses the Gini impurity criterion for classification and mean squared error for regression. Unlike ID3 and C4.5, CART produces binary trees, meaning each internal node splits into exactly two branches.

e) **Modern Developments:**

In the 1990s and 2000s, decision trees became integral to ensemble methods. Random Forests (proposed by Breiman in 2001) use multiple decision trees to improve performance and robustness. Gradient Boosted Trees (such as XGBoost) also emerged, building trees sequentially to correct the errors of previous trees.

f) **Ongoing Research:**

Research continues to improve decision tree algorithms and their applications. Advances focus on enhancing interpretability, handling large datasets, and integrating decision trees with other machine learning techniques.

Decision trees have evolved significantly since their inception, becoming a cornerstone in machine learning and data science [25].

5.2.3 Evolution of Decision Tree algorithm

The evolution of decision tree algorithms can be seen as a progression from early theoretical foundations to sophisticated practical applications. Here's a detailed timeline of the key developments:

a) **Early Concepts (1960s-1970s):**

- **Theoretical Foundations:** Early work in decision theory explored concepts related to decision-making and classification, laying the groundwork for decision tree algorithms.

b) **ID3 Algorithm (1986):**

- **Ross Quinlan's ID3:** The ID3 (Iterative Dichotomiser 3) algorithm was one of the first practical decision tree algorithms. It uses entropy and information gain to decide the best splits for classification tasks. It introduced the concept of recursively partitioning the data to build a tree structure.

c) **C4.5 Algorithm (1993):**

- **Quinlan's Improvements:** Quinlan introduced C4.5, an enhancement of ID3. C4.5 addressed several limitations of ID3:
- **Handling Missing Values:** C4.5 can handle missing data by using probabilistic methods.
- **Pruning:** C4.5 includes a pruning step to reduce the risk of overfitting by removing branches that have little predictive power.
- **Handling Continuous Attributes:** It introduced methods for handling continuous attributes, which ID3 struggled with.

d) **CART Algorithm (1986):**

- **Breiman et al.'s CART:** The Classification and Regression Trees (CART) algorithm, developed by Breiman, Friedman, Olshen, and Stone, provided an alternative approach:
- **Binary Trees:** CART generates binary trees (each node splits into exactly two branches), unlike the multi-way splits in ID3 and C4.5.
- **Gini Impurity and Mean Squared Error:** CART uses the Gini impurity for classification and mean squared error for regression.

e) **Random Forests (2001):**

- **Breiman's Random Forests:** Random Forests, introduced by Leo Breiman, enhance decision trees by using an ensemble method:

- **Bagging:** It builds multiple decision trees using bootstrapped samples of the data and aggregates their predictions to improve accuracy and robustness.
- **Feature Randomization:** Random Forests select a random subset of features for each split, reducing correlation between trees and improving performance.

f) **Gradient Boosting Machines (2000s):**

- **Boosted Trees:** Techniques like Gradient Boosting, pioneered by researchers such as Jerome Friedman, build trees sequentially. Each new tree corrects errors made by the previous ones:
- **XGBoost (2016):** Developed by Tianqi Chen, XGBoost (Extreme Gradient Boosting) is a highly optimized and efficient implementation of gradient boosting that includes additional regularization to prevent overfitting.

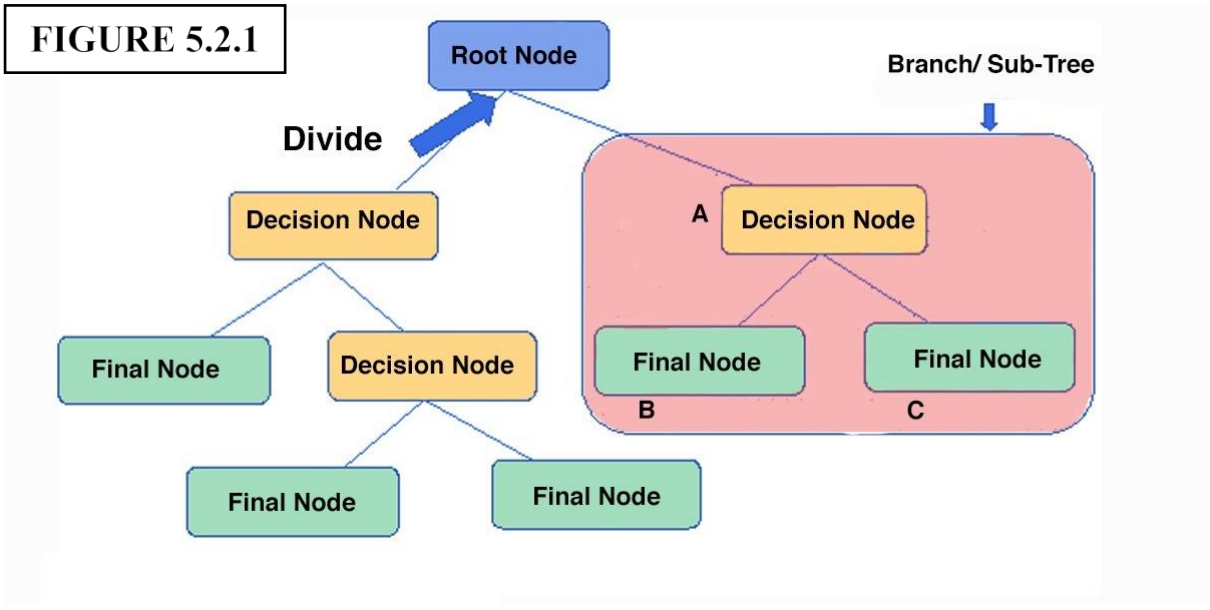
g) **Recent Developments (2010s-Present):**

- **Integration with Deep Learning:** Decision trees are increasingly combined with deep learning methods to handle complex tasks and improve predictive performance.
- **Explainability and Interpretability:** Efforts continue to enhance the interpretability of decision tree models and their integration into various real-world applications.

Decision trees have evolved from simple theoretical concepts to powerful, flexible tools used in a wide range of machine learning and data science tasks [26].

5.2.4 How It Is Used

FIGURE 5.2.1



The decision tree algorithm is used in various domains for classification, regression, and more complex tasks. Here's a detailed look at its applications and how it is used:

a) Classification:

- **Purpose:** To categorize data into predefined classes or categories.
- **How It Works:** The tree splits the data based on features to classify samples into different categories. Each leaf node represents a class label.

Example Applications:

- **Medical Diagnosis:** Predicting diseases based on patient symptoms and test results.
- **Spam Detection:** Classifying emails as spam or not spam based on their content.

b) Regression:

- **Purpose:** To predict continuous numerical values.
- **How It Works:** The tree splits the data to predict continuous values by averaging the target values in the leaf nodes.

Example Applications:

- **House Price Prediction:** Estimating house prices based on features like size, location, and number of rooms.
- **Stock Price Forecasting:** Predicting future stock prices based on historical data and market indicators.

c) Decision-Making:

- **Purpose:** To support decision-making by evaluating different options based on specific criteria.
- **How It Works:** The decision tree models different decisions and their possible outcomes, helping to choose the best option.

Example Applications:

- **Credit Scoring:** Evaluating the likelihood of a customer defaulting on a loan based on their credit history and other factors.
- **Marketing Strategies:** Determining the best marketing strategy based on customer behavior and demographics.

d) Customer Segmentation:

- **Purpose:** To divide customers into distinct groups based on similarities in their behavior or characteristics.
- **How It Works:** The tree splits the data to identify segments or clusters of customers with similar attributes.

Example Applications:

- **Targeted Advertising:** Segmenting customers to create targeted advertising campaigns.
- **Product Recommendations:** Identifying customer segments that are likely to be interested in specific products.

e) Feature Selection:

- **Purpose:** To select the most important features for modeling.
- **How It Works:** Decision trees can highlight the most significant features based on how often they are used for splitting in the tree.

Example Applications:

- **Dimensionality Reduction:** Reducing the number of features in a dataset to simplify models and improve performance.

f) Anomaly Detection:

- **Purpose:** To identify unusual or outlier observations in the data.
- **How It Works:** Decision trees can help detect anomalies by highlighting data points that do not fit well within the learned patterns.

Example Applications:

- **Fraud Detection:** Identifying unusual transactions that may indicate fraudulent activity.

g) Rule Extraction:

- **Purpose:** To generate human-readable rules from a model.
- **How It Works:** The decision tree can be converted into a set of if-then rules that describe the decision-making process.

Example Applications:

- **Business Rules:** Extracting business rules for decision support systems.

Implementation Steps:

a) **Data Preparation:** Clean and preprocess the data, handling missing values and encoding categorical variables if needed.
b) **Model Training:** Fit the decision tree model to the training data by selecting the best features and splitting criteria.
c) **Model Evaluation:** Assess the model's performance using metrics like accuracy, precision, recall, or mean squared error, depending on the task.
d) **Model Tuning:** Adjust hyperparameters (e.g., tree depth, minimum samples per leaf) to improve performance and prevent overfitting.
e) **Deployment:** Use the trained model to make predictions on new data or inform decision-making processes.

Decision trees are versatile and can be applied in various fields, making them a valuable tool for many machine learning and data analysis tasks [27].

5.2.5 Use of decision tree algorithm

Decision trees are widely used in real-life applications due to their simplicity and interpretability. Here are some notable examples across different domains:

a) Healthcare:

- **Diagnosis and Treatment Planning:** Decision trees help in diagnosing diseases based on patient symptoms, medical history, and test results. For instance, they can assist in determining the likelihood of diseases like cancer or diabetes and suggest appropriate treatments or further tests.
- **Predicting Patient Outcomes:** They are used to predict patient outcomes and potential complications, helping doctors make informed decisions about patient care.

b) Finance:

- **Credit Scoring:** Financial institutions use decision trees to assess the creditworthiness of applicants. The model evaluates factors like income, credit history, and existing debt to determine the likelihood of loan repayment.
- **Fraud Detection:** Decision trees can identify fraudulent activities by analyzing transaction patterns and flagging anomalies that deviate from typical behavior.

c) Marketing:

- **Customer Segmentation:** Businesses use decision trees to segment customers based on purchasing behavior, demographics, and preferences. This helps in targeting marketing campaigns more effectively.
- **Churn Prediction:** They predict customer churn by analyzing factors that contribute to customers leaving, allowing companies to implement retention strategies.

d) Retail:

- **Inventory Management:** Decision trees assist in predicting inventory needs by analyzing sales data, seasonal trends, and customer demand, helping retailers optimize stock levels.
- **Product Recommendations:** Retailers use decision trees to recommend products based on customer purchase history and preferences.

e) Insurance:

- **Risk Assessment:** Insurance companies use decision trees to assess risk and determine premiums based on factors such as age, health, and lifestyle for health insurance, or driving history and vehicle type for auto insurance.
- **Claims Processing:** They help in processing claims by evaluating the validity and potential payout based on historical data and claim details.

f) Manufacturing:

- **Quality Control:** Decision trees are used to identify factors affecting product quality and to predict potential defects in manufacturing processes.
- **Predictive Maintenance:** They help in predicting equipment failures by analyzing historical maintenance data and operational conditions.

g) Telecommunications:

- **Customer Service:** Decision trees assist in automating customer service processes, such as troubleshooting common issues based on customer complaints and system diagnostics.
- **Network Optimization:** They help in optimizing network performance by analyzing usage patterns and predicting potential network issues.

h) Human Resources:

- **Employee Attrition:** Decision trees predict employee turnover by analyzing factors such as job satisfaction, compensation, and work environment, helping HR departments take proactive measures.
- **Recruitment:** They assist in evaluating job candidates by analyzing qualifications, experience, and performance metrics.

i) Transportation:

- **Route Optimization:** Decision trees help in optimizing routes for logistics and delivery services by analyzing factors such as traffic conditions, delivery times, and fuel costs.
- **Demand Forecasting:** They are used to forecast transportation demand and plan services accordingly, such as predicting passenger numbers for public transport.

j) Education:

- **Student Performance:** Decision trees predict student performance based on factors like attendance, study habits, and prior academic achievements, helping educators tailor interventions and support.
- **Course Recommendation:** They assist in recommending courses or career paths based on students' interests, skills, and academic performance.

In each of these applications, decision trees provide a clear and interpretable way to make decisions based on data, making them a valuable tool in real-life scenarios [28].

5.2.6 How is decision tree algorithm used to predict stock market prices

Using decision tree algorithms to predict stock market prices involves a few steps, adapting the algorithm to handle the specific characteristics of time series data. Here's a detailed approach on how this can be done:

a) Data Collection and Preprocessing

➢ **Data Collection:**

- **Historical Stock Prices:** Gather historical data including features like opening price, closing price, high, low, volume, and other market indicators.
- **Additional Features:** Consider incorporating other relevant features such as economic indicators, news sentiment, or financial ratios.

➢ **Preprocessing:**

- **Feature Engineering:** Create features from raw data that might help in prediction, such as moving averages, price volatility, or technical indicators (e.g., Relative Strength Index, Moving Average Convergence Divergence).
- **Normalization:** Normalize or standardize features if needed, especially if using multiple features with different scales.
- **Handling Missing Values:** Address missing or incomplete data through imputation or removal.
- **Creating Lag Features:** For time series prediction, create lagged features (previous time periods' prices) to capture temporal dependencies.

b) Splitting Data

➢ **Training and Testing Data:**

- **Temporal Split:** Since stock prices are time-dependent, split the data chronologically into training and testing sets. This ensures that the testing data simulates future unseen data and avoids lookahead bias.

c) Building the Decision Tree Model

➢ **Model Training:**

- **Training:** Use historical data to train the decision tree model. The features can include lagged prices and other technical indicators, while the target variable is the future price or price movement (e.g., up/down, increase/decrease).
- **Feature Selection:** Select features that have predictive power. Decision trees can help in understanding which features are most important for predictions.

➢ **Tree Structure:**

- **Splitting Criteria:** The decision tree algorithm will use criteria like mean squared error (for regression) or Gini impurity/entropy (for classification) to determine the best splits.
- **Pruning:** Prune the tree to avoid overfitting. This involves cutting back the tree to make it more generalizable by removing branches that have little predictive power.

d) Model Evaluation

➢ **Performance Metrics:**

- **Regression Metrics:** If predicting continuous prices, use metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared to evaluate the model's performance.
- **Classification Metrics:** If predicting price direction (up/down), use metrics like accuracy, precision, recall, or F1-score.

➢ **Cross-Validation:**

- **Time Series Cross-Validation:** Use techniques like rolling window cross-validation to assess how well the model generalizes over different time periods.

e) Making Predictions

➢ **Future Predictions:**

- **Input Features:** Feed the model with the latest features derived from recent stock prices and indicators.
- **Forecasting:** Use the trained decision tree to predict future prices or price movements.

f) Post-Processing and Analysis

➢ **Result Analysis:**

- **Comparison:** Compare the model's predictions with actual market outcomes to assess accuracy and reliability.
- **Visualization:** Visualize predictions against actual prices to understand the model's performance and to identify any patterns or biases.

➢ **Integration with Other Models:**

- **Ensemble Methods:** Combine decision trees with other models (e.g., Random Forests, Gradient Boosting) or time series methods (e.g., ARIMA) for improved predictions.

➢ **Challenges and Considerations**

- **Overfitting:** Decision trees can easily overfit historical data, so it's crucial to prune the tree and use techniques like cross-validation to ensure the model generalizes well.
- **Feature Selection:** Choosing the right features is critical for effective predictions. Irrelevant or noisy features can degrade model performance.

- **Temporal Dynamics:** Stock prices are influenced by complex, often non-stationary factors, making prediction challenging. Decision trees may not capture all temporal patterns effectively.

In summary, while decision trees can be used for predicting stock market prices, they are often combined with other techniques and feature engineering methods to improve accuracy and robustness.

5.2.7 Code:

[32]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

[33]:
```
# Load dataset
data = pd.read_csv(r'C:\Users\Radhey Sharma\OneDrive\cuty pie file.csv')

# Display first few rows
data.head()
```

[33]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	03-05-2024	1150.900024	1163.250000	1134.099976	1141.500000	1140.619751	9993632
1	06-05-2024	1143.900024	1156.000000	1135.650024	1143.650024	1142.768066	7490104
2	07-05-2024	1146.050049	1148.800049	1124.050049	1127.699951	1126.830322	7115509
3	08-05-2024	1124.650024	1135.449951	1118.250000	1128.650024	1127.779663	8175609
4	09-05-2024	1122.000000	1134.550049	1112.199951	1115.650024	1114.789673	5226693

[41]:
```
# Features: use relevant columns like 'Open', 'High', 'Low', 'Volume', etc.
X = data[['Open', 'High', 'Low', 'Volume']]  # Modify according to your dataset

# Target: predict the 'Close' price
y = data['Close']
print(X)
```

```
          Open         High          Low    Volume
0   1150.900024  1163.250000  1134.099976   9993632
1   1143.900024  1156.000000  1135.650024   7490104
2   1146.050049  1148.800049  1124.050049   7115509
3   1124.650024  1135.449951  1118.250000   8175609
4   1122.000000  1134.550049  1112.199951   5226693
..          ...          ...          ...       ...
58  1182.349976  1194.599976  1163.650024  19598573
59  1164.099976  1180.000000  1160.199951  18090045
60  1157.000000  1171.000000  1154.000000  17347076
61  1168.000000  1175.650024  1165.449951  11190500
62  1164.000000  1167.800049  1156.099976  10581445

[63 rows x 4 columns]
```

[45]:
```
# Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Verify the shapes of training and testing sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (50, 4)
X_test shape: (13, 4)
y_train shape: (50,)
y_test shape: (13,)
```

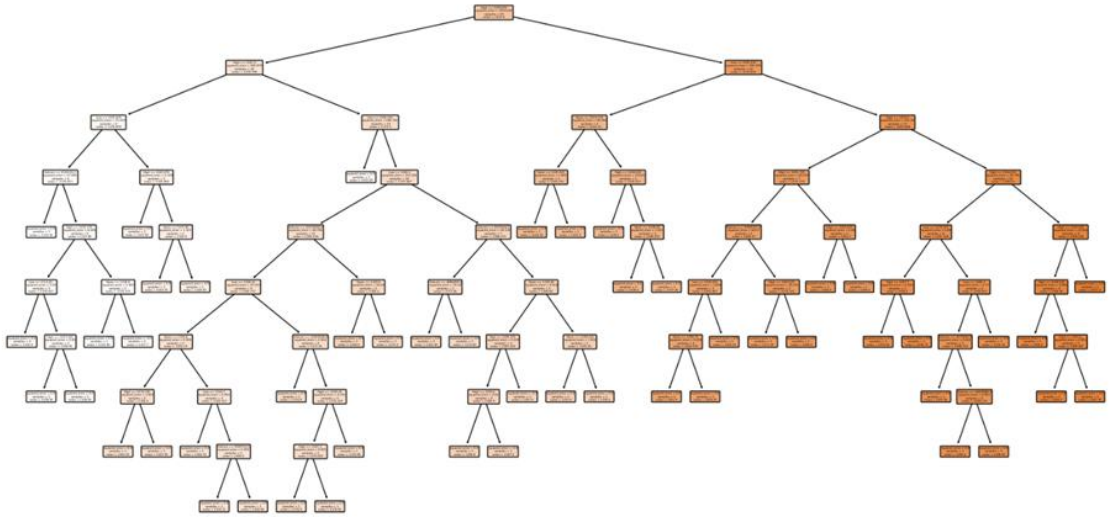
[47]:
```
# Initialize the Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)

# Train the model
regressor.fit(X_train, y_train)
```

[47]:
```
DecisionTreeRegressor(random_state=42)
```

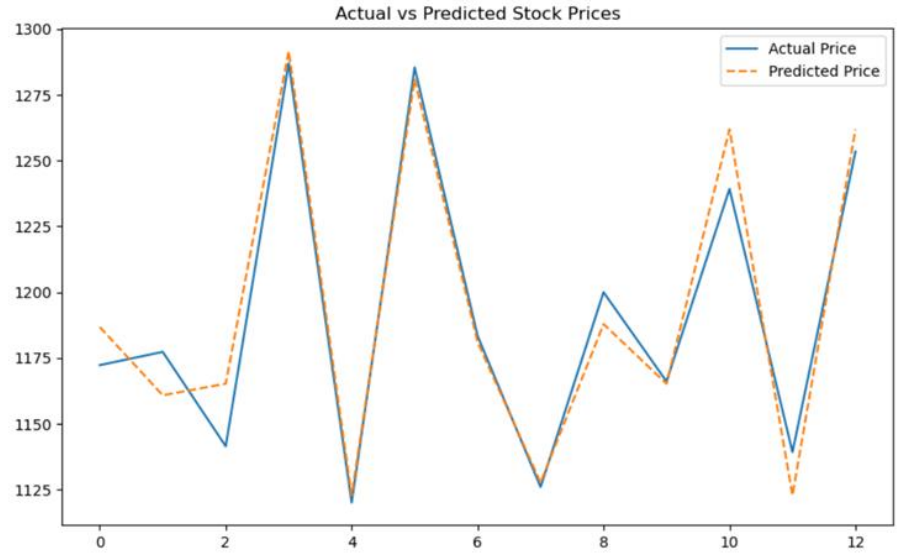
[49]:
```
# Predict on the test set
y_pred = regressor.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE) and R-squared (R2)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Mean Squared Error (MSE): 162.07
R-squared (R2): 0.95
```

[51]:
```
# Visualize the decision tree
plt.figure(figsize=(20,10))
plot_tree(regressor, feature_names=X.columns, filled=True, rounded=True)
plt.show()
```



[53]:
```
# Plot the true vs predicted stock prices
plt.figure(figsize=(10,6))
plt.plot(y_test.values, label='Actual Price')
plt.plot(y_pred, label='Predicted Price', linestyle='dashed')
plt.legend()
plt.title('Actual vs Predicted Stock Prices')
plt.show()
```



5.2.8 Important Terms in code

- **import pandas as pd**: This imports the pandas library, which is essential for working with datasets, especially in CSV or Excel formats.
- **import numpy as np**: Imports NumPy, a library used for numerical computations and handling arrays.
- **train_test_split**: This function splits your dataset into two parts—one for training the model and one for testing it. Here, 80% of the data is used for training, and 20% is used for testing.
- **DecisionTreeRegressor**: A type of regression model that uses a decision tree algorithm to predict continuous values (like stock prices). It works by splitting the dataset into smaller segments based on feature values.
- **fit(X_train, y_train)**: This trains the Decision Tree model using the training data (X_train, y_train).
- **predict(X_test)**: Once the model is trained, this method is used to predict the target variable (stock prices) for the test dataset.
- **mean_squared_error (MSE)**: An evaluation metric that calculates the average of the squared differences between the predicted and actual values. It measures the accuracy of the model—the lower, the better.
- **r2_score (R^2)**: The coefficient of determination, which indicates how well the regression model fits the data. A score of 1 means perfect fit, and 0 means no correlation between predictions and actual values.
- **plot_tree**: This function from sklearn is used to visualize the decision tree, showing how the model splits the data at different levels.
- **plt.plot**: A function from matplotlib used for plotting graphs. Here, it is used to plot the actual vs predicted stock prices.
- **plt.show()**: Displays the plotted figures, whether it's the decision tree or the stock price comparison plot.

5.2.9 Mathematic used in Decision tree algorithm

a) Variance Reduction (for Regression Problems)

For regression trees, instead of using entropy or Gini index, **variance reduction** is used as a splitting criterion. Variance measures how much the values in a dataset deviate from the mean.

The variance of a dataset SSS is calculated as:

$$\text{Var}(S) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mu)^2$$

Where:

- yiy_iyi is the value of the ithi^{th}ith instance in the dataset.
- μ\muμ is the mean of the target variable in the dataset.
- n is the number of instances.

To find the best split, the decision tree algorithm looks for the attribute that results in the greatest reduction in variance after splitting.

b) Variance Reduction:

The reduction in variance is calculated as:

$$\Delta \text{Var}(S, A) = \text{Var}(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \text{Var}(S_v)$$

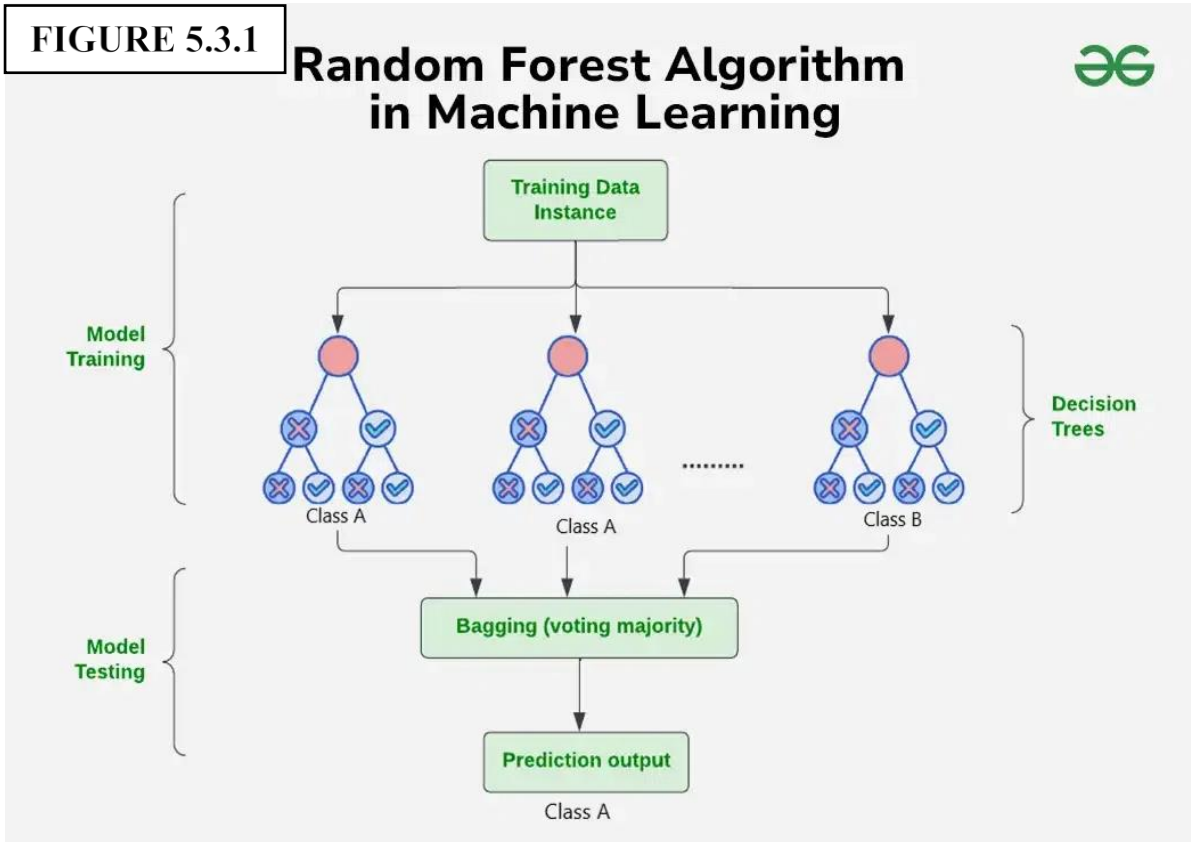
5.3 Random Forest

5.3.1 Introduction to Random Forest

Random Forest is a powerful and versatile supervised machine learning algorithm that belongs to the family of ensemble methods. Introduced by Leo Breiman in 2001, it has become one of the most widely used algorithms in both academic research and industry due to its simplicity and effectiveness. Random Forest excels in both regression and classification problems, often producing great results even without extensive hyperparameter tuning.

The algorithm works by building a number of decision trees on different samples of the data. In classification problems, it takes the majority vote from the trees to determine the final result. For regression tasks, it averages the predictions from the trees. Random Forest is particularly valued for its ability to handle complex datasets with high dimensionality, reduce overfitting, and provide accurate predictions [29].

FIGURE 5.3.1



Source:https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

- **Example of Random Forest:**

Suppose you're planning a solo trip but can't decide whether to visit a hill station or go on an adventure. You ask your friends for advice. Friend 1 (F1) suggests a hill station, reasoning that

it's November, a perfect time to enjoy the hills. Friend 2 (F2), on the other hand, recommends an adventurous destination. Each of your friends provides their own suggestion. In the end, you have two options: either choose a place based on your personal preference, or go with the location that the majority of your friends recommended.

Similarly, in a Random Forest, multiple decision trees are trained, and the final output is determined either by the majority vote (for classification problems) or by averaging the predictions (for regression problems).

5.3.2 Historical Background

The concept of Random Forest builds upon earlier work on decision trees and ensemble methods. Decision trees, which have been in use since the 1960s, are simple yet powerful models used for classification and regression tasks. However, decision trees are prone to overfitting, especially when dealing with complex datasets.

In 1996, Leo Breiman introduced the concept of bagging (Bootstrap Aggregating), an ensemble method that trains multiple models on different subsets of the data and averages their predictions. Bagging laid the groundwork for the development of Random Forest. In 2001, Breiman, along with Adele Cutler, introduced Random Forest as an extension of bagging, adding randomness to the feature selection process for each tree, which helps reduce correlation among the trees and improves overall model performance [29].

5.3.3 Advantages and Disadvantages of Random Forest

a) Advantages-

- **High accuracy**: Random Forest typically delivers higher accuracy due to its ensemble nature, which aggregates the results of multiple decision trees.
- **Robust to overfitting:** By averaging multiple trees, Random Forest reduces the risk of overfitting, leading to better generalization.
- **Handles missing data:** Random Forest can handle missing values in both features and target variables, making it robust to imperfect data.
- **Provides feature importance insights:** Random Forest can rank the importance of each feature in predicting the target variable, aiding in feature selection.
- **Versatile (classification & regression):** Random Forest can be used for both classification and regression tasks, making it a versatile tool.

- **Works well with large datasets:** Random Forest can process large datasets efficiently and is scalable to high-dimensional data.
- **Reduces variance (better generalization):** By combining multiple trees, Random Forest reduces the variance of predictions, leading to more stable results.

b) Disadvantages-

- **Complexity:** The model is complex, involving numerous decision trees, which makes it less interpretable compared to simpler models like decision trees.
- **Computationally intensive:** Training a Random Forest requires significant computational resources, especially with large datasets.
- **Slower predictions:** Due to the ensemble of trees, making predictions can be slower compared to single decision trees or other simpler models.
- **Bias in imbalanced datasets:** Random Forest may produce biased results if the dataset is highly imbalanced, as it might not accurately represent minority classes.
- **Doesn't perform well on low-dimensional data:** In datasets with a small number of features, simpler models may outperform Random Forest due to its complexity [34].

5.3.4 Decision Tree: The Foundation of Random Forest

Before delving deeper into Random Forest, it's essential to understand the foundation upon which it is built: the decision tree. A decision tree is a non-parametric, supervised learning model used for both classification and regression tasks. It works by recursively splitting the data into subsets based on the value of input features, creating a tree-like structure.

➢ **Key Characteristics of Decision Trees:**

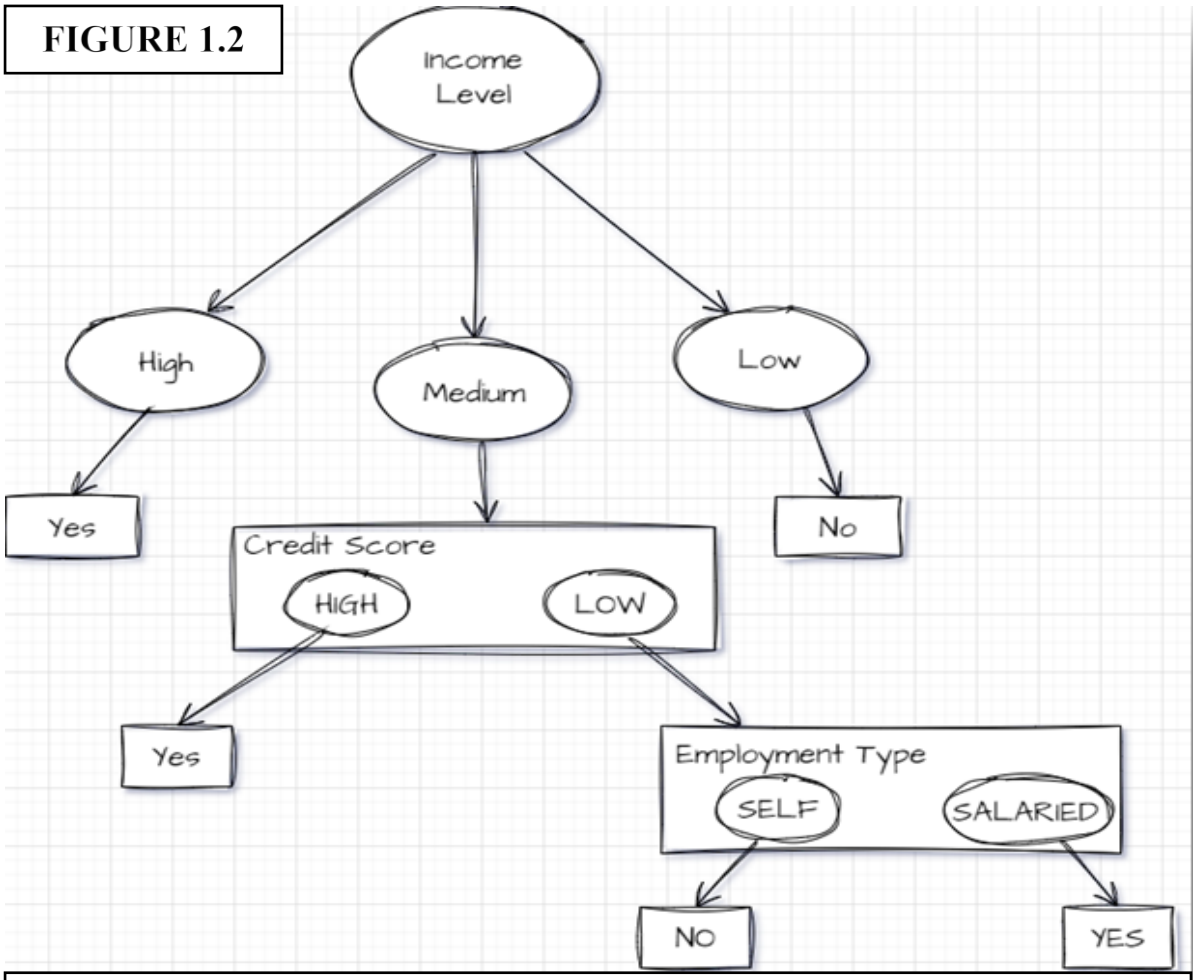
- **Nodes and Leaves:** In a decision tree, each internal node represents a decision based on the value of a feature, while each leaf node represents the outcome or the final decision (class label in classification or value in regression).
- **Splitting Criteria:** The decision tree algorithm selects the best feature to split the data at each node based on criteria such as Gini impurity, information gain (for classification), or variance reduction (for regression).
- **Interpretability:** One of the significant advantages of decision trees is their interpretability. The tree structure provides a clear visual representation of the decision-making process, making it easy to understand how the model arrives at a particular decision.

- **Overfitting:** A major drawback of decision trees is their tendency to overfit the training data, especially when the tree is deep and has many nodes. Overfitting occurs when the model learns the noise in the training data rather than the underlying pattern, leading to poor generalization on new, unseen data [31].

➢ **Example of Decision Tree in Action:**

Imagine a decision tree model designed to classify whether a loan applicant should be approved or denied based on their income and credit score. The tree might first split the data based on income, with applicants earning above a certain threshold moving down one branch and those below it moving down another. The tree might then further split based on credit score and employment type, eventually reaching a leaf node that decides whether to approve or deny the loan.

FIGURE 1.2



Source: https://blog.devgenius.io/implementing-a-loan-eligibility-checker-using-decision-trees-in-c-5737bafeec8b

5.3.5 Evolution of Random Forest

Random Forest evolved as a solution to the limitations of individual decision trees. By aggregating the predictions of multiple decision trees, Random Forest significantly improves prediction accuracy and reduces overfitting.

- **Ensemble of Decision Trees:** Random Forest consists of multiple decision trees, each trained on a random subset of the data. The final prediction is made by aggregating the predictions of all the individual trees.
- **Random Feature Selection:** During the construction of each tree, Random Forest selects a random subset of features for splitting at each node. This randomness reduces the correlation between the trees, leading to more robust and generalized models.
- **Bootstrap Sampling:** Each tree in the Random Forest is trained on a bootstrap sample of the data, which is a random sample with replacement. This further enhances the diversity among the trees and reduces the risk of overfitting.
- **Out-of-Bag Error:** Random Forest can estimate the accuracy of the model without the need for a separate validation set. The out-of-bag error is calculated using the data points that were not included in the bootstrap sample for each tree [32].

5.3.6 Applications of Random Forest

Random Forest is a highly versatile algorithm used in various domains for different types of tasks:

- **Classification:** Random Forest is widely used for classification tasks where the goal is to predict categorical outcomes. Examples include spam detection, medical diagnosis, and customer segmentation.
- **Regression:** Random Forest is also used for regression tasks where the goal is to predict continuous values. Examples include predicting house prices, stock market trends, and sales forecasting.
- **Feature Selection:** Random Forest can be used to identify the most important features in a dataset. This is particularly useful in situations where the dataset contains a large number of features, and not all of them are relevant to the task at hand.

- **Clustering:** Although less common, Random Forest can also be used for clustering tasks, where the goal is to group similar data points together [33].

5.3.7 Types of Random Forest and Applied Type

Random Forest can be broadly categorized into two types based on the nature of the task it performs: **Random Forest for Classification** and **Random Forest for Regression**. These types are defined by the kind of target variable they predict—categorical or continuous.

a) **Random Forest for Classification:**

- **Purpose:** In classification tasks, the target variable is categorical. The model predicts discrete classes or labels. Random Forest for classification is commonly used in applications like email spam detection, customer segmentation, and medical diagnosis.
- **Process:** Each decision tree in the forest predicts a class, and the final prediction is made based on the majority vote among the trees. This approach reduces the risk of overfitting and improves the model's generalization ability.

b) **Random Forest for Regression:**

- **Purpose:** In regression tasks, the target variable is continuous. The model predicts numerical values. Random Forest for regression is widely applied in domains such as stock market prediction, house price estimation, and sales forecasting.
- **Process:** Each decision tree predicts a continuous value, and the final prediction is obtained by averaging the outputs of all the trees. This reduces variance and provides a more stable prediction, especially in cases with noisy data [33-34].

➢ **Applied Type in Stock Market Prediction**

In the context of stock market prediction, **Random Forest for Regression** is typically used. Since stock prices are continuous variables, the model aims to predict the future price of a stock based on historical data and other financial indicators. By leveraging the averaging mechanism, Random Forest regression provides robust predictions in the inherently volatile and noisy financial market.

5.3.8 Mathematics behind Random Forest

i. Gini Index

In Random Forest, the Gini Index is a metric that measures the impurity or purity of a split in a decision tree. The goal of the algorithm is to create splits that result in the most homogeneous or pure subgroups of data, which can either be all positive (e.g., "yes") or all negative (e.g., "no"). The Gini Index helps quantify how mixed or impure a group is after a split.

- **Splitting Using the Gini Index**

Consider a dataset with multiple features, and we want to determine which feature to use as the root node for our decision tree. For example, when we select **Feature 1** as the root node, the resulting split is pure, with all instances belonging to one class. However, when selecting **Feature 2**, the split is impure, with instances from both classes appearing in the subgroups. The Gini Index helps us quantify this impurity, and we choose the feature with the lowest Gini Index for the root node to achieve the purest split.

Mathematically, the Gini Index is calculated as:

$$Gini = 1 - (P^2_+ + P^2_-) \quad \text{(i)}$$

where P_+ is the probability of the positive class, and P_- is the probability of the negative class. A Gini Index of 0 indicates a perfectly pure split, while a Gini Index closer to 0.5 indicates a more impure split.

- **Example Calculation**

Let's illustrate this with a toy dataset related to loan approvals. Assume we use the feature **Loan Amount** as the root node and split the dataset into two groups: those who receive the loan and those who don't. After splitting, we calculate the Gini Index for each subgroup.

For the **left split** (those who receive the loan):

$$Gini_{left} = 1 - (P^2_+ + P^2_-)$$

For the **right split** (those who don't receive the loan):

$$Gini_{Right} = 1 - (P^2_+ + P^2_-)$$

Finally, the weighted Gini Index is calculated as the average Gini Index of all the subgroups, weighted by the size of each subgroup. The Random Forest algorithm will calculate the Gini

Index for every possible feature split and select the feature that results in the lowest Gini Index, i.e., the purest split.

- **Gini Index vs. Entropy**

The Gini Index is often favoured over another metric called **Entropy** when building decision trees. Entropy, used in algorithms like ID3, measures impurity but involves a logarithmic calculation:

$$Entropy = -\Sigma Pi \log_2(P_i) \quad \text{(ii)}$$

Though Entropy is effective, the Gini Index is computationally more efficient since it doesn't involve logarithmic operations. This makes it a popular choice in boosting algorithms and decision trees, as it reduces the time complexity of the training process [37].

ii. Regression Metrics for Random Forest

Random Forest is also widely used for regression tasks, where the goal is to predict continuous values rather than classes. In these cases, several metrics are used to evaluate the model's performance:

a) **Mean Squared Error (MSE)**: MSE is a standard measure of model performance in regression tasks. It calculates the average squared difference between actual and predicted values. The formula for MSE is:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (yi - \hat{y}i)^2 \quad \text{(iii)}$$

A lower MSE indicates that the model's predictions are closer to the actual values, while a higher MSE indicates greater deviation.

b) **Mean Absolute Error (MAE)**: MAE measures the average absolute difference between actual and predicted values. Unlike MSE, it is less sensitive to outliers, as it does not square the errors. The formula for MAE is:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} (yi - \hat{y}i) \quad \text{(iv)}$$

MAE provides a direct interpretation of the average error in the same units as the target variable.

c) **R-Squared (R^2) Score**: The R^2 score, or coefficient of determination, indicates the proportion of the variance in the target variable that can be explained by the model. It ranges

from 0 to 1, where 1 indicates perfect prediction and 0 indicates no predictive power. The formula for R^2 is:

$$R^2 = 1 - \frac{\sum_{I=1}^{n}(yi - \hat{y}i)^2}{\sum_{I=1}^{n}(yi - \bar{y})^2} \quad \text{(v)}$$

A high R^2 score indicates that the model captures most of the variability in the target variable, while a low R^2 score suggests the model may not be a good fit for the data.

5.3.9 Real-Life Applications of Random Forest

Random Forest is employed in a wide range of real-life applications, demonstrating its flexibility and effectiveness:

- **Healthcare:** In the healthcare industry, Random Forest is used for predicting patient outcomes, diagnosing diseases, and identifying high-risk patients. It helps medical professionals make data-driven decisions and improve patient care.
- **Finance:** In the financial sector, Random Forest is used for credit scoring, fraud detection, and stock market prediction. It helps financial institutions assess risk, detect fraudulent activities, and make informed investment decisions.
- **Marketing:** In marketing, Random Forest is used for customer segmentation, churn prediction, and recommendation systems. It helps businesses understand customer behavior, predict churn, and personalize marketing strategies.
- **Environmental Science:** In environmental science, Random Forest is used for predicting weather patterns, modeling ecosystems, and assessing the impact of climate change. It helps scientists and policymakers make informed decisions to protect the environment [34-35].

5.3.10 Comparison between Random Forest and Decision Tree

While Random Forest and decision trees are closely related, they differ in several key aspects:

- **Model Complexity:** A decision tree is a single model, whereas Random Forest is an ensemble of multiple decision trees. Random Forest tends to be more complex and computationally intensive compared to a single decision tree.

- **Overfitting:** Decision trees are prone to overfitting, especially with complex datasets. Random Forest mitigates this issue by averaging the predictions of multiple trees, leading to more generalized models.
- **Prediction Accuracy:** Random Forest generally provides more accurate predictions compared to a single decision tree. This is due to the ensemble nature of the model, which combines the strengths of multiple trees.
- **Stability:** Random Forest is more stable and less sensitive to changes in the training data compared to a single decision tree. Small variations in the training data can lead to significant changes in a decision tree, while Random Forest remains more robust.
- **Interpretability:** While decision trees are highly interpretable due to their simple structure, Random Forest, as an ensemble of trees, can be more challenging to interpret. However, the trade-off is that Random Forest usually offers better performance and accuracy [32].

5.3.11 Application of Random Forest in Stock Market Prediction

Random Forest is increasingly being used to predict stock market prices due to its ability to handle complex and noisy financial data. The stock market is influenced by numerous factors, including historical prices, trading volume, economic indicators, and market sentiment. Random Forest can analyze these diverse features and identify patterns that help in predicting future stock prices.

In a typical stock market prediction task, the following steps are often involved:

a) **Data Collection:** Historical stock price data and other relevant financial indicators are collected. This data is typically obtained from financial databases or APIs.

b) **Feature Engineering:** New features are created based on the raw data. For example, moving averages, trading volume changes, and sentiment scores can be calculated and added as features.

c) **Model Training:** The Random Forest model is trained on the historical data. The model learns the relationships between the input features and the stock prices.

d) **Prediction:** Once trained, the model can be used to predict future stock prices based on new data.

e) **Evaluation:** The model's performance is evaluated using metrics such as mean squared error (MSE) to assess its accuracy in predicting stock prices [36].

5.3.12 Dataset Description

In this application, we use a dataset consisting of **Axis Bank NSE data from the last three months**, which has been derived from Yahoo Finance. This dataset is crucial for building the predictive model, as it contains historical stock price data and other relevant financial metrics.

Key Features of the Dataset:

- **Date Range:** Last three months
- **Source:** Yahoo Finance
- **Key Variables:**
  - **Date:** The specific trading day.
  - **Open Price:** The price of the stock at the start of the trading day.
  - **Close Price:** The price of the stock at the end of the trading day.
  - **High Price:** The highest price the stock reached during the trading day.
  - **Low Price:** The lowest price the stock reached during the trading day.
  - **Volume:** The number of shares traded during the day.
  - **Adjusted Close Price:** The closing price adjusted for corporate actions such as dividends and stock splits.

This dataset serves as the foundation for building and training the Random Forest model for stock market prediction. The historical price data and volume information will be used to engineer features that capture the trends and patterns in the stock's movement, which are critical for making accurate predictions.

5.3.13 Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import warnings
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

warnings.filterwarnings("ignore")

# Retrieve data
df = pd.read_csv("C:/Users/Divya/OneDrive/Desktop/AXISBANK.NS.csv", parse_dates=['Date'],
                 index_col='Date')

# Define the feature columns and target column
features = ['Open', 'High', 'Low' , 'Adj Close']
target = 'Close'

# Prepare features and target variable
x = df[features]
y = df[target]

# Train/test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=45,
                                                    shuffle=False)

# Create Random Forest Regressor object
rf = RandomForestRegressor(n_estimators=100, max_depth=5, min_samples_split=2, min_samples_leaf=1,
                           max_features='sqrt', bootstrap=True)

# Fitting RF Regression to the Training set
rf.fit(x_train, y_train)
```

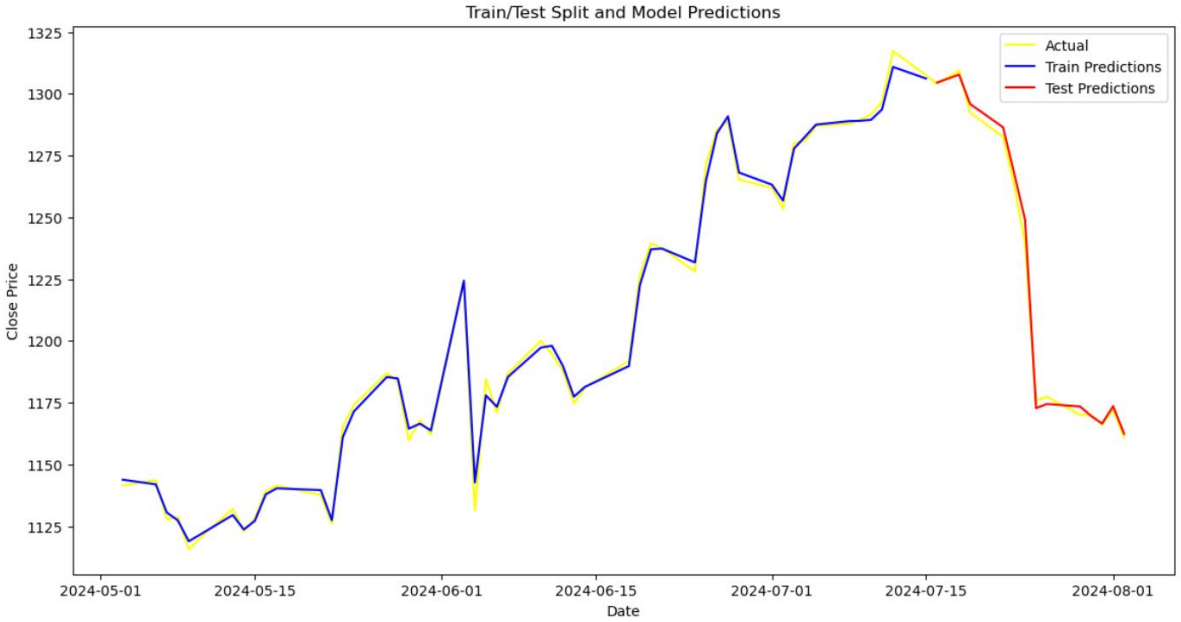
```
# Predicting the Train and Test set results
y_train_pred = rf.predict(x_train)
y_test_pred = rf.predict(x_test)

# Evaluate model's performance on train data
r2_train = r2_score(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = mean_squared_error(y_train, y_train_pred, squared=False)
mae_train = mean_absolute_error(y_train, y_train_pred)

print("Train Data Evaluation:")
print("R2 Score:", r2_train)
print("Mean Squared Error:", mse_train)
print("Root Mean Squared Error:", rmse_train)
print("Mean Absolute Error:", mae_train)
```

```
# Evaluate model's performance on test data
r2_test = r2_score(y_test, y_test_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = mean_squared_error(y_test, y_test_pred, squared=False)
mae_test = mean_absolute_error(y_test, y_test_pred)

print("Test Data Evaluation:")
print("R2 Score:", r2_test)
print("Mean Squared Error:", mse_test)
print("Root Mean Squared Error:", rmse_test)
print("Mean Absolute Error:", mae_test)
```

```
# Plot the results
plt.figure(figsize=(14, 7))

# Plot actual values
plt.plot(df.index, df[target], label='Actual', color='yellow')

# Plot train predictions
plt.plot(x_train.index, y_train_pred, label='Train Predictions', color='blue')

# Plot test predictions
plt.plot(x_test.index, y_test_pred, label='Test Predictions', color='red')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Train/Test Split and Model Predictions')
plt.legend()
plt.show()
```

```
done = pd.DataFrame({'Actual Close': y_test, 'Predicted Close': y_test_pred})
done
```

5.3.14 Output of code




```
Train Data Evaluation:
R2 Score: 0.9974303433956839
Mean Squared Error: 9.844189874385707
Root Mean Squared Error: 3.1375451987797254
Mean Absolute Error: 2.4454648862793467
```

```
Test Data Evaluation:
R2 Score: 0.9958465767990438
Mean Squared Error: 14.002324024373793
Root Mean Squared Error: 3.741967934706789
Mean Absolute Error: 2.8521930502746065
```

Date	Actual Close	Predicted Close
2024-07-16	1304.000000	1303.653223
2024-07-18	1309.400024	1308.475005
2024-07-19	1292.349976	1297.083203
2024-07-22	1282.500000	1285.804267
2024-07-23	1263.250000	1270.367786
2024-07-24	1239.250000	1250.477703
2024-07-25	1175.900024	1172.604606
2024-07-26	1177.349976	1174.061039
2024-07-29	1170.050049	1174.143823
2024-07-30	1170.000000	1169.525721
2024-07-31	1166.099976	1165.939323
2024-08-01	1172.300049	1173.496358
2024-08-02	1160.849976	1163.082613

5.3.15 Important terms in code

a) Libraries and Modules

- **numpy (np)**: A library for numerical operations and working with arrays.
- **matplotlib.pyplot (plt)**: A module for creating visualizations.
- **pandas (pd)**: A library for data manipulation and analysis.
- **warnings**: A module for managing and suppressing warnings.
- **sklearn.ensemble.RandomForestRegressor**: A machine learning model for regression tasks.
- **sklearn.model_selection.train_test_split**: A function to split data into training and testing sets.
- **sklearn.metrics**: Module containing functions to evaluate model performance.

b) Data Handling

- **pd.read_csv**: Reads data from a CSV file into a Pandas DataFrame.
- **parse_dates**: Specifies columns to parse as datetime objects.
- **index_col**: Sets a column as the index of the DataFrame.

c) Feature and Target Definition

- **features**: List of columns used as input features for the model.
- **target**: Column to be predicted by the model.

d) **Data Preparation**

- **x**: DataFrame containing the feature columns.
- **y**: Series containing the target column.

e) Train/Test Split

- **train_test_split**: Function to split data into training and testing sets.
- **test_size**: Proportion of the dataset to include in the test split.
- **random_state**: Seed for random number generation, ensuring reproducibility.

- **shuffle**: Option to shuffle data before splitting.

f) Model Creation and Training

- **RandomForestRegressor**: Machine learning model for regression using a random forest approach.
- **n_estimators**: Number of trees in the forest.
- **max_depth**: Maximum depth of each tree.
- **min_samples_split**: Minimum number of samples required to split an internal node.
- **min_samples_leaf**: Minimum number of samples required to be at a leaf node.
- **max_features**: Number of features to consider when looking for the best split.
- **bootstrap**: Whether to use bootstrap samples when building trees.

g) Model Evaluation

- **fit**: Method to train the model.
- **predict**: Method to make predictions using the trained model.
- **r2_score**: Metric to evaluate the proportion of variance explained by the model.
- **mean_squared_error**: Metric to measure the average squared error between actual and predicted values.
- **mean_absolute_error**: Metric to measure the average absolute error between actual and predicted values.
- **squared=False**: Option in mean_squared_error to compute the root mean squared error (RMSE).

h) Plotting

- **plt.figure**: Initializes a new figure for plotting.
- **plt.plot**: Plots data on the figure.
- **plt.xlabel**: Sets the label for the x-axis.
- **plt.ylabel**: Sets the label for the y-axis.

- **plt.title**: Sets the title of the plot.
- **plt.legend**: Adds a legend to the plot.
- **plt.show**: Displays the plot.

5.4 Gradient Boosting Algorithm

5.4.1 Introduction to Gradient boosting

Gradient Boosting is a machine learning method that combines the power of multiple simple models to make better predictions. It works well for problems where you want to predict numbers (like prices) or categories (like spam or not spam). The word "gradient" refers to the technique used to reduce mistakes while the model is learning. The term "boosting" means that it takes several weaker models (that aren't very accurate on their own) and combines them to create a stronger, more accurate model. This approach helps improve the overall accuracy of predictions [38].

5.4.2 How Gradient Boosting Works

Gradient Boosting is a method that helps a computer make better predictions by fixing its mistakes step by step. It has three main parts: a way to measure mistakes (loss function), a simple model to make predictions (weak learner), and a method to add more simple models to improve accuracy (additive model).

Here's how it works:

a) **Start with a Simple Model:** The process begins with a basic model, like a small decision tree. This model makes predictions on the data, but it might not be very accurate.

b) **Check for Mistakes:** The computer then checks how far off the predictions are from the real answers. This difference is called the "error" or "loss."

c) **Add a New Model:** Instead of fixing the first model, Gradient Boosting adds a new model that focuses on correcting the mistakes made by the first one.

d) **Repeat the Process:** This new model is trained to fix the remaining errors. Then, it is combined with the first model to improve the predictions. The process is repeated, with each new model trying to correct the errors made by the models before it.

e) **Stop When It's Good Enough:** The computer keeps adding new models until it reaches a point where the mistakes are small enough, or it has added a set number of models.

This step-by-step improvement helps the computer make very accurate predictions by gradually reducing the errors [39].

5.4.3 Historical Background

Boosting was first introduced by Robert Schapire in 1990 with the idea that combining several weak learners could create a strong learner. This concept was further developed by Schapire and Yoav Freund, leading to the creation of the AdaBoost Algorithm in 1996. The idea of

gradient boosting came from Leo Breiman, who saw boosting as an optimization problem. Jerome H. Friedman advanced this concept, developing explicit gradient boosting algorithms for regression in 1999 and 2001. His influential paper, "Greedy Function Approximation: A Gradient Boosting Machine," published in 2001, laid the groundwork for Gradient Boosting Machines (GBMs).Around the same time, Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean introduced a broader view of gradient boosting, seeing it as an iterative process that optimizes a cost function by selecting functions (or weak learners) that move in the direction of the negative gradient. Over the years, gradient boosting has evolved, leading to the development of faster and more accurate algorithms like XGBoost, LightGBM, and CatBoost. These modern versions are popular in data science because of their flexibility and ability to handle large datasets effectively [40].

5.4.4 Decision Tree: The Foundation of Gradient Boosting

A decision tree is a flowchart-like structure where each internal node represents a feature (or attribute), each branch represents a decision rule, and each leaf node represents the outcome. The tree starts with a root node and splits the data into subsets based on the value of the feature. This process is repeated recursively, creating a tree structure that helps in decision-making.

Key components of a decision tree:

- Root Node: The topmost node that represents the entire dataset.
- Internal Nodes: Nodes that represent features and are used to split the data.
- Leaf Nodes: Terminal nodes that represent the final decision or classification.

Advantages:

- Easy to understand and interpret.
- Can handle both numerical and categorical data.
- Requires little data preprocessing.

Disadvantages:

- Prone to overfitting, especially with deep trees.
- Can be unstable with small variations in data [39].

5.4.5 Decision Trees in Gradient Boosting

In the context of Gradient Boosting, decision trees are used as weak learners. Here's how they fit into the boosting framework:

a) **Sequential Learning:** Gradient Boosting builds trees sequentially. Each new tree aims to correct the errors made by the previous trees. This is done by fitting the new tree to the residual errors of the previous tree.
b) **Weak Learners:** The decision trees used in Gradient Boosting are typically shallow, meaning they have a limited depth. This prevents overfitting and ensures that each tree is a weak learner, contributing only a small improvement to the model.
c) **Combining Trees:** The final model is a weighted sum of all the trees. Each tree's prediction is scaled by a learning rate, which controls the contribution of each tree to the final model. This helps in fine-tuning the model and prevents overfitting.
d) **Optimization:** Gradient Boosting optimizes a loss function by adding trees that point in the direction of the negative gradient of the loss function. This iterative process continues until the model reaches a specified number of trees or the improvement in the loss function becomes negligible [39].

Example:

Imagine you are predicting house prices. The first decision tree might predict an initial estimate. The second tree will then predict the errors (residuals) of the first tree. The third tree will predict the errors of the second tree, and so on. Each tree corrects the mistakes of the previous ones, leading to a more accurate final model.

By using decision trees as weak learners, Gradient Boosting effectively combines their simplicity and interpretability with the power of ensemble learning to create robust predictive models.

5.4.6 Applications of Gradient Boosting:

a) **Finance**: Gradient Boosting is widely used in the finance industry to make better predictions-
   - **Credit Scoring:** It helps banks predict if someone might not repay a loan by looking at past data.
   - **Fraud Detection:** It identifies fake transactions by spotting unusual patterns in how money is spent.
   - **Stock Market Prediction:** It helps forecast stock prices and market trends by analyzing past stock data and financial information.
b) **Healthcare:** In healthcare, Gradient Boosting improves patient care and hospital efficiency-

- **Disease Prediction:** It predicts the chances of someone getting a disease, helping doctors catch it early and plan treatments.
- **Patient Risk Assessment:** It assesses the risk of a patient needing to return to the hospital, helping hospitals manage resources and care better.

c) **Marketing**: Gradient Boosting is useful in marketing for understanding customers and improving sales strategies-

- **Customer Segmentation:** It groups customers based on their behavior and preferences, allowing companies to create more targeted marketing campaigns.
- **Churn Prediction:** It predicts which customers might stop using a service, helping businesses take action to keep them.

d) **Examples in Stock Market Prediction:** Gradient Boosting is especially good at predicting stock prices because it can handle large amounts of data and find complex patterns. For example-

- **CalixBoost Model:** This model uses Gradient Boosting to predict stock market indexes by looking at economic data, financial indicators, and even social media sentiment.
- **Gradient Boosting Neural Networks (GBNN):** By combining Gradient Boosting with neural networks, this approach improves the accuracy of stock market predictions. Overall, Gradient Boosting is a powerful tool that helps in making better decisions across different fields by analyzing complex data [42].

5.4.7 Types of Gradient Boosting

Gradient Boosting has several versions that are specially designed to be faster, more accurate, or handle specific types of data. Here are three popular ones

a) **XGBoost (eXtreme Gradient Boosting)**

- **Strengths:** XGBoost is fast and performs really well. It has built-in features to prevent the model from becoming too complex and making mistakes (called overfitting). It can also use multiple computer cores at once, making training quicker.
- **Use Cases:** XGBoost is used in many areas, like predicting if someone will repay a loan (credit scoring), finding health risks (disease prediction), and understanding customer behavior (marketing).

b) **LightGBM (Light Gradient Boosting Machine)**

- **Strengths:** LightGBM is designed to be super fast and work well with large amounts of data. It uses a method that saves memory and speeds up training, especially with big datasets.

- **Use Cases:** LightGBM is great when you need quick results and are working with a lot of data, such as in real-time predictions, big data analysis, and fast stock trading.

c) **CatBoost (Categorical Boosting)**

- **Strengths:** CatBoost is excellent at handling data that includes categories (like color, type, or brand). It also works well even if some data is missing, so you don't have to spend a lot of time cleaning up the data.
- **Use Cases:** CatBoost is used in areas where the data has lots of categories, like recommending products in online stores, detecting fraud in finance, and any field where data is tricky to prepare.

These different versions of Gradient Boosting are useful for different tasks, making them flexible tools for solving various problems in machine learning [41].

5.4.8 Mathematics behind Gradient Boosting

a) Initial Model:

- Start with an initial prediction F0(x). For regression, this is typically the mean of the target values:

$$F_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$$

- Where yi are the target values and n is the number of data points.

b) Calculate Residuals:

- For each iteration mmm, calculate the residuals, which are the differences between the actual values yi and the current predictions Fm−1(xi):

$$r_{im} = y_i - F_{m-1}(x_i)$$

- These residuals represent the errors the current model is making.

c) Train a Weak Learner:

- Train a weak learner (often a decision tree) to predict the residuals rim. Let hm(x) represent the predictions made by this weak learner.

d) Update the Model:

- Update the current model by adding the predictions of the weak learner, scaled by a learning rate α:

$$F_m(x) = F_{m-1}(x) + \alpha \cdot h_m(x)$$

- Here, α is a small number (e.g., 0.1) that controls the contribution of each weak learner.

e) Repeat:

- Repeat steps 2-4 for M iterations, where M is the number of weak learners you want to include in the final model.

f) Final Prediction:

- After M iterations, the final model FM(x) is the sum of all the weak learners:

$$F_M(x) = F_0(x) + \alpha \sum_{m=1}^{M} h_m(x)$$

- This model is used to make predictions on new data.

Summary of Key Formulas:

- **Residuals**: rim=yi−Fm−1(xi)
- **Model Update**: Fm(x)=Fm−1(x)+α·hm(x)
- **Final Model**: FM(x)=F0(x)+α∑m=1Mhm(x)

These formulas form the backbone of the Gradient Boosting algorithm. Each weak learner helps to gradually improve the model by focusing on the mistakes made in previous iterations [6].

5.4.9 Application of Gradient Boosting in Stock Market Prediction

Gradient Boosting is a great tool for predicting stock prices because it can handle complicated data and make accurate predictions. Here's a simple guide on how it can be used:

a) Dataset

What's in the Data: The data usually includes past stock prices and other related information, which you can get from places like Yahoo Finance or Bloomberg.

Types of Data:

- **Open, High, Low, Close Prices:** These are the daily prices of the stock.
- **Volume**: The number of shares traded each day.
- **Technical Indicators:** Tools like moving averages and RSI (Relative Strength Index) that help predict stock movements.
- **Fundamental Data:** Information like company earnings and financial ratios.

b) Feature Engineering

What It Is: Feature engineering means creating new data from the existing data to help the model make better predictions.

Steps:

- **Creating Lag Features:** Using past prices to help predict future ones.
- **Technical Indicators:** Calculating things like moving averages and Bollinger Bands to get more insights.
- **Sentiment Analysis:** Adding information from news articles or social media to see how people feel about a stock.
- **Fundamental Features:** Including important financial data like earnings and ratios.

c) Model Training

Getting Ready: Before training the model, you need to prepare the data.

Steps:

- **Data Preprocessing:** Clean the data, fill in any missing parts, and make sure everything is on the same scale.
- **Splitting the Data**: Divide the data into two parts: one for training the model and one for testing it.
- **Hyperparameter Tuning:** Adjust the settings of the model to find the best ones using techniques like GridSearchCV.
- **Training the Model:** Use the prepared data to teach the Gradient Boosting model how to make predictions.

d) Prediction

- **Making Predictions**: After training, the model can predict future stock prices.
- **Keeping It Updated**: As new data comes in, update the predictions to keep them accurate.

e) Evaluation

Why It's Important: You need to check how well the model is doing to make sure it's reliable.

Steps:

- **Mean Squared Error (MSE):** This checks how far off the predictions are on average.
- **R-squared (R^2) Score:** This shows how much of the stock price movement the model can explain.
- **Backtesting:** Compare the model's predictions with actual past prices to see how well it would have worked in real life [44].

5.4.10 Real-Life Applications of Gradient Boosting

Gradient Boosting is a powerful tool used in the financial sector for making accurate predictions and handling complex data. Here's a simpler explanation of how it's applied:

a) Detecting Fraud

Banks and financial institutions use Gradient Boosting to spot fraudulent transactions. By analyzing transaction patterns and finding anything unusual, these models can quickly alert the system to potential fraud, helping to prevent financial losses and keep customer accounts safe.

b) Credit Scoring

Credit bureaus like Experian and Equifax use Gradient Boosting to improve how they calculate credit scores. By looking at various financial details, like how well someone pays their bills or how much credit they're using, these models provide more accurate credit scores. This helps lenders decide whether someone is likely to pay back a loan or not.

c) Predicting Stock Prices

Investment firms use Gradient Boosting to predict how stock prices might change. They analyze past stock data, market trends, and economic factors to make smarter investment choices. This helps them to buy or sell stocks at the right time, optimizing their profits.

d) Keeping Customers

While not strictly financial, e-commerce companies use Gradient Boosting to understand which customers might stop buying from them. By analyzing shopping habits and customer engagement, they can identify who might leave and take steps to keep them, like offering special deals.

These examples show how Gradient Boosting helps companies in finance and beyond make better decisions and run their operations more smoothly [44].

5.4.11 Dataset Description

In this application, we are using data from the past three months of Axis Bank's stock on the NSE, which we got from Yahoo Finance. This data is important for building our predictive model because it includes historical stock prices and other key financial information.

Key Features of the Dataset:

- **Date Range**: Last three months
- **Source:** Yahoo Finance

Key Variables:

- **Date:** The specific trading day.
- **Open Price:** The price of the stock at the start of the trading day.

- **Close Price:** The price of the stock at the end of the trading day.
- **High Price:** The highest price the stock reached during the trading day.
- **Low Price:** The lowest price the stock reached during the trading day.
- **Volume:** The number of shares traded during the day.
- **Adjusted Close Price:** The closing price adjusted for corporate actions such as dividends and stock splits.

This dataset is the base for creating and training our Gradient Boosting model to predict stock prices. We'll use the past price data and trading volume to create features that help us understand the stock's trends and patterns. These features are essential for making accurate predictions about future stock movements.

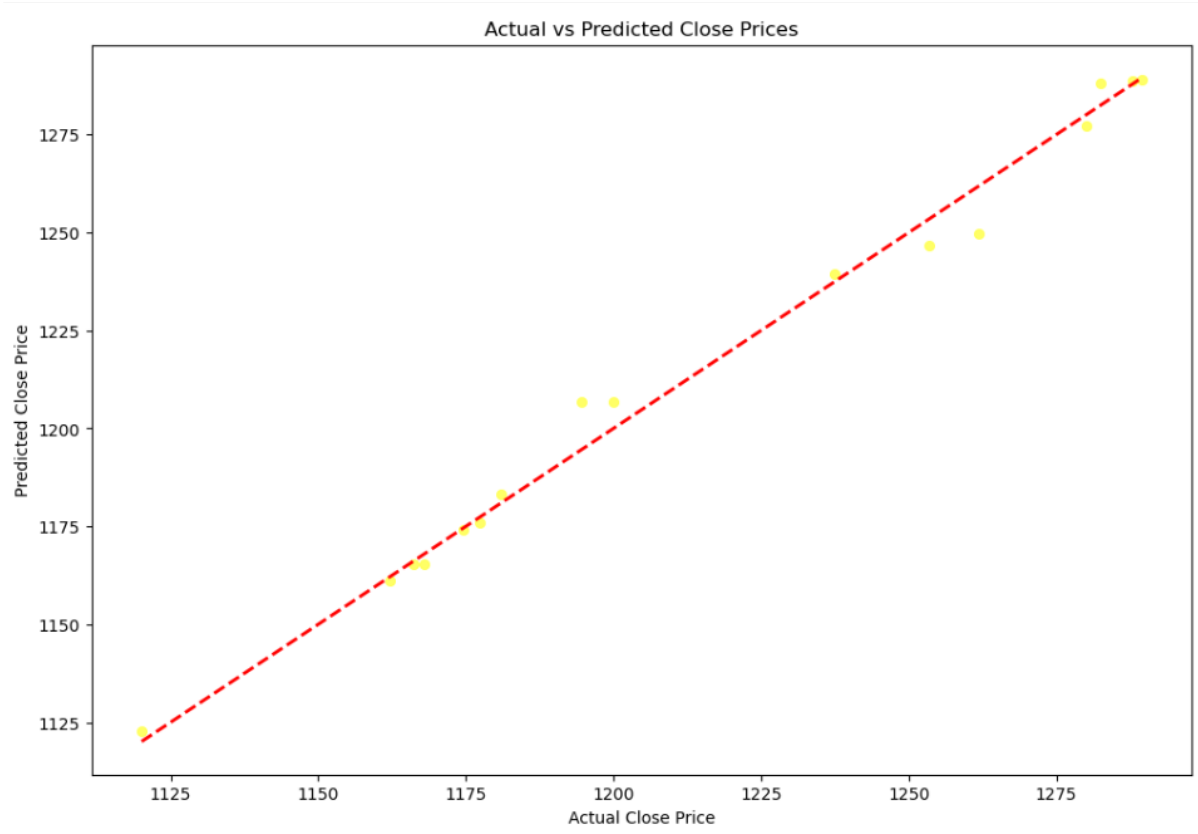
5.4.12 Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset
# Assuming the dataset is in CSV format and has the required columns
data = pd.read_csv('C:/Users/ISHAAN UPPAL/Desktop/Axis Bank.csv', parse_dates=['Date'], index_col='Date')

data = data.dropna()

# Features and target variable
features = ['Open', 'High', 'Low', 'Adj Close']
X = data[features]
y = data['Close']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=45)

# Standardizing the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initializing the model
gbr = GradientBoostingRegressor()

# Training the model
gbr.fit(X_train, y_train)
```

```
# Making predictions
y_train_pred = gbr.predict(X_train)
y_test_pred = gbr.predict(X_test)

# Evaluating the model
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Train MSE: {train_mse}")
print(f"Test MSE: {test_mse}")
print(f"Train R²: {train_r2}")
print(f"Test R²: {test_r2}")
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Create the DataFrame with actual and predicted values
done = pd.DataFrame({'Actual Close': y_test, 'Predicted Close': y_test_pred})

# Set up the matplotlib figure
plt.figure(figsize=(12, 8))

# Create a scatter plot
sns.scatterplot(data=done, x='Actual Close', y='Predicted Close', alpha=0.6, color='yellow', edgecolor=None)

# Add a line for perfect prediction
min_val = done['Actual Close'].min()
max_val = done['Actual Close'].max()
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--', linewidth=2)

# Add labels and title
plt.xlabel('Actual Close Price')
plt.ylabel('Predicted Close Price')
plt.title('Actual vs Predicted Close Prices')

# Show plot
plt.show()
```

5.4.13 Output of code:



Date	Actual Close	Predicted Close
13-06-2024	1174.650024	1174.211724
14-06-2024	1181.050049	1183.347756
31-07-2024	1166.099976	1165.263932
03-07-2024	1280.000000	1277.253826
10-05-2024	1120.099976	1122.779968
02-07-2024	1253.400024	1246.690803
01-07-2024	1261.900024	1249.640102
22-07-2024	1282.500000	1288.064508
26-07-2024	1177.349976	1175.814460
21-06-2024	1237.449951	1239.351963
08-07-2024	1287.849976	1288.619412
31-05-2024	1162.150024	1161.133891
30-05-2024	1167.949951	1165.355310
10-06-2024	1200.000000	1206.683088
11-06-2024	1194.599976	1206.738928
09-07-2024	1289.400024	1288.945833

```
Train MSE: 0.0005923834882248535
Test MSE: 28.35835715546793
Train R²: 0.9999998555055705
Test R²: 0.9899270071623459
```

5.4.14 Important Terms in Code

a) Libraries and Modules

- numpy (np): Used for performing calculations and working with arrays of numbers.
- matplotlib.pyplot (plt): Helps create graphs and visual representations of data.
- pandas (pd): A tool for managing and analyzing data in table form.
- warnings: Handles and suppresses warnings that may arise during code execution.
- sklearn.ensemble.RandomForestRegressor: A machine learning model for predicting continuous outcomes, like stock prices.
- sklearn.model_selection.train_test_split: Splits data into parts for training and testing the model.
- sklearn.metrics: Offers various functions to measure how well the model performs.

b) Data Handling

- pd.read_csv: Loads data from a CSV file into a DataFrame, which is like a spreadsheet.
- parse_dates: Indicates which columns should be treated as dates.
- index_col: Chooses a column to serve as the index or reference for rows in the DataFrame.

c) Feature and Target Definition

- features: The columns from the data that the model will use to make predictions.
- target: The specific column that represents what the model is trying to predict.

d) **Data Preparation**

- x: Contains the feature data, which is what the model will use as input.
- y: Contains the target data, representing the output the model needs to predict.

e) Train/Test Split

- train_test_split: Divides the data into training and testing sets to check how well the model performs.
- test_size: Determines what portion of the data is reserved for testing.
- random_state: Ensures the split is consistent each time the code runs, so results can be reproduced.

- shuffle: Mixes the data before splitting it to avoid bias.

f) Model Creation and Training

- RandomForestRegressor: A model that uses many decision trees to make more accurate predictions.
- n_estimators: The total number of trees in the forest.
- max_depth: The maximum depth each tree can grow, which controls how detailed the tree is.
- min_samples_split: The minimum number of data points needed to split a node and make a decision.
- min_samples_leaf: The minimum number of data points required in a final leaf node of the tree.
- max_features: The number of input features considered when choosing how to split the data in the tree.
- bootstrap: Decides whether to use random subsets of data when creating each tree.

g) Model Evaluation

- fit: Trains the model using the training data.
- predict: Uses the trained model to make predictions on new or test data.
- r2_score: Shows how much of the variation in the target data the model can explain.
- mean_squared_error: Measures the average of the squares of the differences between predicted and actual values.
- mean_absolute_error: Measures the average of the absolute differences between predicted and actual values.
- squared=False: An option to calculate the root mean squared error, which is easier to interpret.

h) Plotting

- plt.figure: Starts a new plot.
- plt.plot: Draws the data on the plot.
- plt.xlabel: Adds a label to the x-axis (horizontal axis).

- plt.ylabel: Adds a label to the y-axis (vertical axis).
- plt.title: Sets a title for the plot.
- plt.legend: Explains what different lines or symbols on the plot represent.
- plt.show: Displays the plot on the screen.

5.5 Linear Regression Algorithm

5.5.1. What is Linear Regression?

Linear Regression is a statistical method that models the relationship between a dependent variable (usually denoted as *yy*y) and one or more independent variables (denoted as *xx*x). The goal is to fit a straight line to the data points that minimizes the distance (error) between the actual and predicted values. In simple terms, Linear Regression finds the "best fit" line to predict the value of one variable based on the values of others.

The equation of a simple linear regression model is:

$y=\beta0+\beta1x+\epsilon y = \beta_0 + \beta_1 x + \epsilon$y=β0 +β1 x+ϵ

Where:

- *yy*y is the dependent variable (e.g., stock price).
- *xx*x is the independent variable (e.g., volume, open, high prices).
- $\beta0\beta_0$β0 is the intercept (the value of *yy*y when $x=0x = 0$x=0).
- $\beta1\beta_1$β1 is the slope of the line.
- $\epsilon\epsilon$ϵ is the error term.

In multiple linear regression, where multiple independent variables are used, the equation becomes:

$y=\beta0+\beta1x1+\beta2x2+\cdots+\beta nxn+\epsilon y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$y=β0 +β1 x1 +β2 x2 +⋯+βn xn +ϵ

5.5.2. History of Linear Regression

Linear regression has its roots in the early works of Sir Francis Galton, a British polymath, in the 19th century. Galton studied the relationship between the heights of parents and their offspring, coining the term "regression" to describe the tendency of offspring's heights to regress toward the mean height. His work led to the development of the regression model, which was later refined by Karl Pearson, who introduced the method of "least squares" to find the best fit line through the data points.

- Galton (1886)[45]: The foundation of regression was laid when Galton published his paper on the relationship between parental heights and their children's heights.

- Pearson (1901)[46]: Karl Pearson developed the formula for the correlation coefficient and worked on the concept of minimizing the squared differences between the observed and predicted values.
- Legendre & Gauss (1805, 1809)[47]: Adrien-Marie Legendre and Carl Friedrich Gauss independently developed the least squares method, which is essential in Linear Regression.

5.5.3. Evolution of Linear Regression

Linear regression has evolved significantly since its early introduction:

- **Early 19th Century:** Least squares method was formally introduced.
- **20th Century:** Linear regression was used extensively in various scientific fields, including economics and biology.
- **Modern Day:** With the advent of computers, linear regression can now handle large datasets and has become a foundation for machine learning and data science. In addition, regularization techniques like Lasso and Ridge regression were introduced to handle overfitting in models with multiple variables[48].

5.5.5. How Linear Regression is Used

Linear regression is widely used in different fields for prediction and analysis:

- **Finance:** To predict stock prices, sales revenue, and market trends.
- **Healthcare:** For predicting patient outcomes based on historical data (e.g., blood pressure, glucose levels).
- **Economics:** For demand forecasting, understanding consumer behavior, and predicting GDP growth.
- **Marketing:** To understand customer behavior, sales forecasting, and pricing optimization.
- **Engineering:** For quality control and reliability analysis.

5.5.6. Use of Linear Regression in Real Life

Some real-life applications of linear regression include:

- **Stock Price Prediction:** As used in this project, linear regression predicts stock prices based on features like past prices, trading volume, and other financial indicators.
- **House Price Prediction:** In real estate, it is used to estimate house prices based on features like area, number of rooms, and location.

- **Sales Forecasting:** Retail companies use linear regression to predict future sales based on historical data and seasonal trends.
- **Risk Management:** In finance, linear regression helps assess risk factors and predict the likelihood of financial losses.

5.5.7. Difference between Linear Regression and Logistic Regression

- **Nature of the Dependent Variable:**
  - **Linear Regression:** The dependent variable is continuous (e.g., stock price).
  - **Logistic Regression:** The dependent variable is categorical or binary (e.g., success/failure, yes/no).
- **Equation:**
  - **Linear Regression:** It uses the formula $y=\beta0+\beta1x+\epsilon$*y = \beta_0 + \beta_1 x + \epsilon*y=β0 +β1 x+ϵ.
  - **Logistic Regression:** It uses the logistic function (sigmoid curve) to map predictions to probabilities. *P(y=1)=11+e−(β0+β1x)P(y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}*P(y=1)=1+e−(β0 +β1 x)1
- **Output:**
  - **Linear Regression:** Directly predicts the value of the dependent variable.
  - **Logistic Regression:** Predicts the probability of the dependent variable belonging to a particular class [49].

5.5.8. How is Linear Regression Used to Predict Stock Market Prices

In stock price prediction, linear regression is used to model the relationship between stock prices and various factors (features) such as:

- **Open Price:** The price at the beginning of the trading day.
- **High and Low Prices:** The highest and lowest prices during the day.
- **Volume:** The total number of shares traded.

The algorithm predicts the stock's closing price based on historical data. By using historical stock data (such as 'Open', 'High', 'Low', 'Volume'), the linear regression model identifies trends and relationships between these variables and the stock's closing price.

Example Workflow for Stock Prediction:

a) **Gather Data:** Historical stock prices and volume.
b) **Feature Selection:** Use independent variables like 'Open', 'High', 'Low', and 'Volume'.

c) **Train Model:** Fit a linear regression model on historical data.

d) **Make Predictions:** Use the model to predict future stock prices based on the available data.

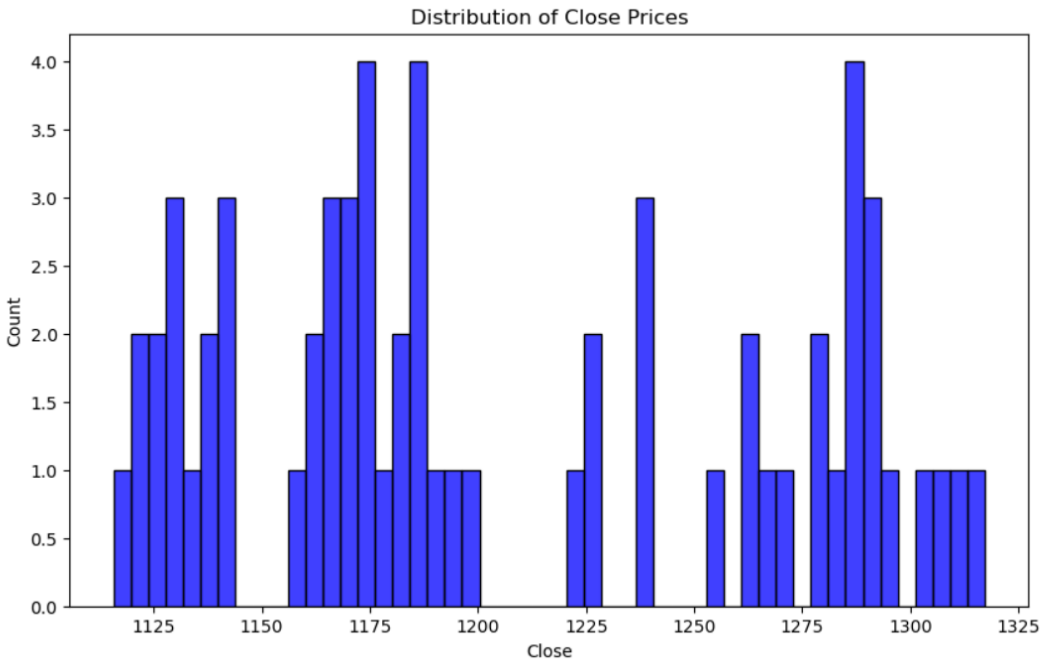
5.5.9. Code

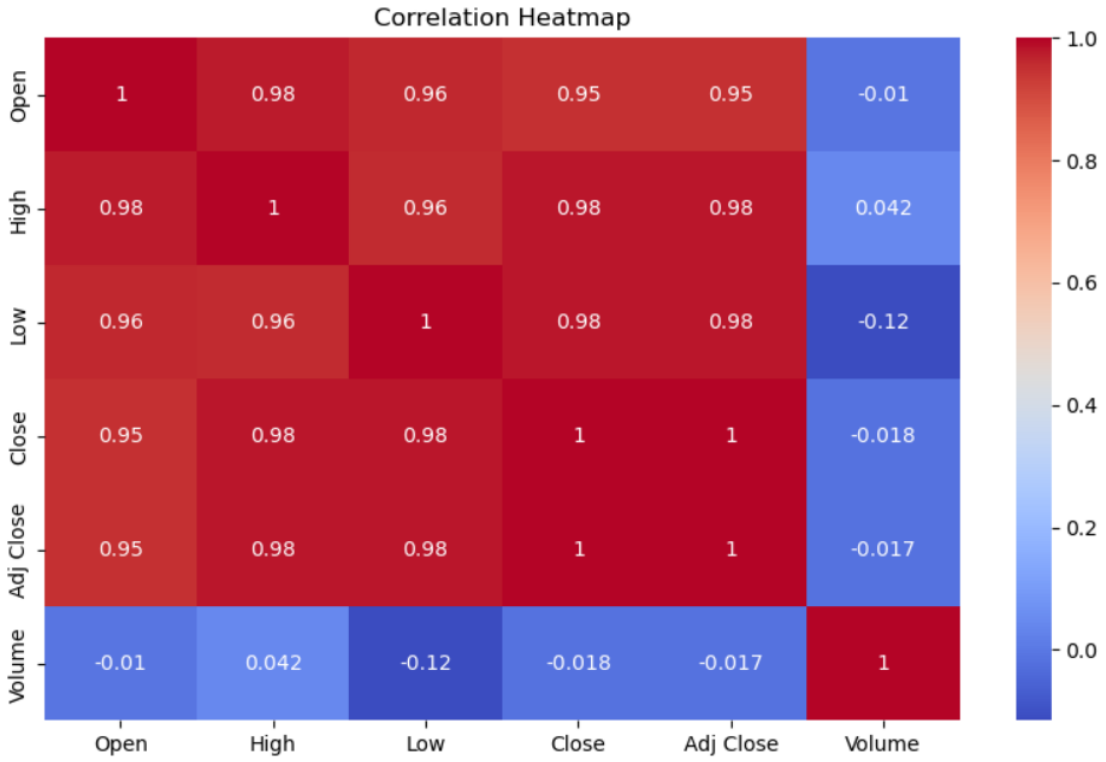
[1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

[3]:
```
# Load the dataset
file_path = r'C:\Users\hp\Downloads\AXISBANK.NS.csv'  # Your dataset path
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()

# Check for missing values
print(data.isnull().sum())
```

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

[11]:
```
# Drop rows with missing values
data = data.dropna()

# Visualize the distribution of 'Close' prices
plt.figure(figsize=(10, 6))
sns.histplot(data['Close'], bins=50, color='blue')
plt.title('Distribution of Close Prices')
plt.show()

# Correlation matrix to analyze relationships between features
plt.figure(figsize=(10, 6))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

# Select features and target variable
X = data[['Open', 'High', 'Low', 'Volume']]  # Features
y = data['Close']  # Target variable
# Check if there's a column that looks like a date and handle it
if 'Date' in data.columns:
    data['Date'] = pd.to_datetime(data['Date'])
    data['Year'] = data['Date'].dt.year
    data['Month'] = data['Date'].dt.month
    data['Day'] = data['Date'].dt.day

    # Add these new features to your X (feature set)
    X = data[['Open', 'High', 'Low', 'Volume', 'Year', 'Month', 'Day']]
else:
    # If there's no 'Date' column, use only existing features
    X = data[['Open', 'High', 'Low', 'Volume']]
```


Distribution of Close Prices
Count
Close
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1125
1150
1175
1200
1225
1250
1275
1300
1325


Correlation Heatmap
Open
High
Low
Close
Adj Close
Volume
1
0.98
0.96
0.95
0.95
-0.01
0.98
1
0.96
0.98
0.98
0.042
0.96
0.96
1
0.98
0.98
-0.12
0.95
0.98
0.98
1
1
-0.018
0.95
0.98
0.98
1
1
-0.017
-0.01
0.042
-0.12
-0.018
-0.017
1
1.0
0.8
0.6
0.4
0.2
0.0

[13]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print the size of the training and test sets
print(f'Training data shape: {X_train.shape}, Testing data shape: {X_test.shape}')
```

```
Training data shape: (50, 4), Testing data shape: (13, 4)
```

[15]:
```
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

# Print the coefficients
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

```
Intercept: 6.021893699021803
Coefficients: [-5.54212260e-01  8.26033093e-01  7.20704272e-01  2.19617465e-07]
```

[17]:
```
# Evaluate the model
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f'R² Score: {r2}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
```

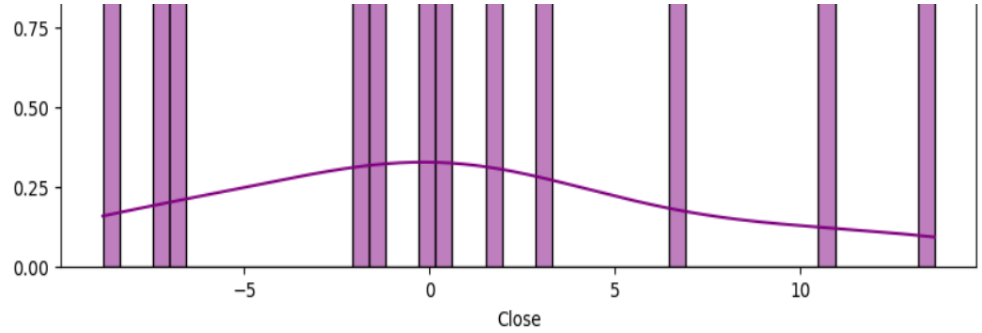
-

[17]:
```
# Evaluate the model
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f'R² Score: {r2}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
```

```
R² Score: 0.986498879016482
Mean Squared Error (MSE): 41.56623643752237
Root Mean Squared Error (RMSE): 6.447188258265953
```

[21]:
```
# Plot residuals
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(0, color='red', linestyle='--', lw=2)
plt.title('Residuals Plot')
plt.xlabel('Predicted Prices')
plt.ylabel('Residuals')
plt.show()
```

[23]:
```
# Distribution of residuals
plt.figure(figsize=(10, 6))
sns.histplot(residuals, bins=50, kde=True, color='purple')
plt.title('Distribution of Residuals')
plt.show()
```



[27]:
```
# Example: Replace with actual values for 'Open', 'High', 'Low', 'Volume'
# For instance, these could be historical stock data for which you want to predict the 'Close' price.
new_data = [[900.50, 910.0, 895.0, 5000000]]  # Replace these with real values

# Make a prediction based on new data
new_pred = model.predict(new_data)

print(f"Predicted stock price for the new data: {new_pred[0]}")
```

```
Predicted stock price for the new data: 904.7722785614919
```

5.5.10. Important Terms in the Code

- **LinearRegression():** This is the linear regression model from scikit-learn, used to fit the data.
- **train_test_split():** This function splits the dataset into training and testing sets.
- **fit():** Trains the model using the training data.
- **predict():** Predicts the dependent variable (e.g., stock price) based on new data.
- **r2_score:** The coefficient of determination that indicates how well the model fits the data.
- **mean_squared_error (MSE):** The average of the squared differences between actual and predicted values.
- **Root Mean Squared Error (RMSE):** The square root of the mean squared error, indicating the model's prediction accuracy in the same units as the data.

5.6 Comparison of Machine Learning Algorithms Based on MSE and R^2 Score

In this section, we evaluate the performance of the machine learning models used for predicting Axis Bank's stock prices. The models are compared based on two key performance metrics: Mean Squared Error (MSE) and R^2 score. MSE measures the average squared difference between actual and predicted values, with a lower MSE indicating better model accuracy. The R^2 score indicates how well the model explains the variance in the target variable, with a value closer to 1 signifying higher accuracy.

I. K-Nearest Neighbors (KNN)

- **R^2 Score**: 0.9951
- **MSE**: 14.46

Analysis: KNN performs exceptionally well with an R^2 score of 0.9951, suggesting that it captures 99.51% of the variance in stock prices. Its MSE of 14.46 is relatively low, indicating accurate predictions with minimal error. The algorithm excels in modeling localized patterns, which may be why it performs so well in this context.

II. Decision Tree

- **R^2 Score**: 0.9533
- **MSE**: 131.61

Analysis: The Decision Tree algorithm has a decent R^2 score of 0.9533, which means it explains 95.33% of the variance in the stock price data. However, its MSE of 131.61 is relatively high compared to other models, indicating that it struggles with accuracy on this dataset. Decision Trees tend to overfit and may not generalize well, especially in complex financial data.

III. Random Forest

- **R^2 Score**: 0.9958
- **MSE**: 14.00

Analysis: Random Forest outperforms most models with an R^2 score of 0.9958, suggesting it explains 99.58% of the variance. It also has the lowest MSE (14.00), making it the most accurate model for predicting stock prices. Random Forest's ability to average multiple decision trees likely contributes to its robust performance and resistance to overfitting.

IV. Gradient Boosting

- **R^2 Score**: 0.9899
- **MSE**: 28.36

Analysis: Gradient Boosting achieves an R^2 score of 0.9899, meaning it explains 98.99% of the variance. Its MSE of 28.36 is higher than both KNN and Random Forest but lower than Decision Tree. Gradient Boosting is a powerful model, especially for handling complex patterns, but it may require more tuning to reach its full potential.

V. Linear Regression

- **R^2 Score**: 0.9865
- **MSE**: 41.57

Analysis: Linear Regression, with an R^2 score of 0.9865, explains 98.65% of the variance in stock prices. Its MSE is the highest among all models (41.57), indicating lower accuracy. While Linear Regression is simpler and interpretable, its assumption of a linear relationship may limit its ability to model complex stock price movements, leading to higher errors.

Model Performance Summary

ALGORITHM	R^2 SCORE	MSE	PERFORMANCE SUMMARY
K-Nearest Neighbors	0.9951	14.46	High accuracy, low error, sensitive to noise.
Decision Tree	0.9533	131.61	Moderate accuracy, high error, prone to overfitting.
Random Forest	0.9958	14.00	Best performer with low error and high accuracy.
Gradient Boosting	0.9899	28.36	Strong accuracy, slightly higher error than Random Forest.
Linear Regression	0.9865	41.57	Simplest model, highest error, limited by linear

- **Best Performance**: Random Forest (R^2: 0.9958, MSE: 14.00)
- **Worst Performance**: Decision Tree (MSE: 131.61)

Random Forest and KNN outperform the other models in terms of both R^2 score and MSE, suggesting they are the best choices for short-term stock price prediction in this context.

Decision Tree, while having a reasonable R^2 score, shows the highest MSE, indicating it is less accurate and prone to error. Linear Regression, though easy to interpret, struggles with higher error rates, whereas Gradient Boosting strikes a balance but still lags behind Random Forest and KNN.

CHAPTER 6

CONCLUSION

This project explored the effectiveness of various machine learning algorithms in predicting the next-day closing price of Axis Bank's stock. We utilized five key models: K-Nearest Neighbors (KNN), Decision Tree, Random Forest, Gradient Boosting, and Linear Regression. The results revealed that ensemble methods such as Random Forest and Gradient Boosting performed better than simpler models like Linear Regression and Decision Tree. Random Forest demonstrated the best overall performance with an R^2 score of 0.9958 and the lowest Mean Squared Error (MSE) of 14.00.

In financial applications, accurate predictions can help investors make better decisions. The application of these machine learning algorithms for stock price predictions provides a robust foundation for further exploration and optimization. Machine learning models, particularly ensemble methods, show great potential for enhancing the accuracy of stock market predictions.

6.1 Recommendations

Based on the results of this study, the following recommendations are made:

- **Prioritize ensemble models:** Random Forest and Gradient Boosting performed exceptionally well. These models should be considered for further financial predictive tasks, as they effectively minimize error and maximize predictive accuracy.
- **Optimize hyperparameters:** While the models performed well, further hyperparameter optimization through methods such as GridSearchCV could potentially improve performance even more.
- **Expand the dataset:** The study focused on a three-month dataset for Axis Bank. Future studies should consider expanding the dataset to include multiple years and other stocks to generalize findings and improve model robustness.

6.2 Future Scope

There are several avenues for future research:

- **Incorporating external data:** Integrating external factors such as macroeconomic indicators, market sentiment, and global events could further improve the model's ability to predict stock prices.

- **Automated trading systems:** The predictive models developed in this study can be integrated into automated trading systems, which execute trades based on real-time stock predictions.
- **Deep learning models**: Investigating the use of deep learning models such as Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) models could yield even more accurate stock price predictions due to their ability to capture temporal dependencies in time-series data.

REFERENCES

[1] IBM. *Machine learning.* Retrieved from https://www.ibm.com/topics/machine-learning

[2] Virtusa. *Regression*. Retrieved from https://www.virtusa.com/digital-themes/regression

[3] Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, *25*(2), 383–417. https://doi.org/10.2307/2325486

[4] Bollen, J., Mao, H., & Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of Computational Science*, *2*(1), 1–8. https://doi.org/10.1016/j.jocs.2010.12.007

[5] Tsai, C. F., & Hsiao, Y. C. (2010). Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decision Support Systems*, *50*(1), 258–269. https://doi.org/10.1016/j.dss.2010.08.028

[6] Ding, X., Zhang, Y., Liu, T., & Duan, J. (2015). Deep learning for event-driven stock prediction. *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*, 2327–2333. https://doi.org/10.24963/ijcai.2015/324

[7] Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications*, *42*(1), 259–268. https://doi.org/10.1016/j.eswa.2014.07.040

[8] Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669. https://doi.org/10.1016/j.ejor.2017.11.054

[9] Shah, D., Isah, H., & Zulkernine, F. (2019). Stock market analysis: A review and taxonomy of prediction techniques. *International Journal of Financial Studies*, *7*(2), 26. https://doi.org/10.3390/ijfs7020026

[10] Huang, W., Nakamori, Y., & Wang, S. Y. (2005). Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, *32*(10), 2513–2522. https://doi.org/10.1016/j.cor.2004.03.016

[11] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(Feb), 281–305. https://jmlr.org/papers/v13/bergstra12a.html

[12] Zhong, X., & Enke, D. (2017). Predicting the daily return direction of the stock market using hybrid machine learning algorithms. *Financial Innovation*, *3*, 11. https://doi.org/10.1186/s40854-017-0067-0

[13] Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long short-term memory. *Neurocomputing*, *356*, 151–162. https://doi.org/10.1016/j.neucom.2017.07.031

[14] de Prado, M. L. (2018). *Advances in financial machine learning*. John Wiley & Sons. https://doi.org/10.1002/9781119482086

[15] Elastic.co. What is K-NN. Retrieved From-

https://www.elastic.co/what-is/knn

[16] Sarala, V. and BHUSHAN, G.P., 2022. Stock market trend prediction using k-nearest neighbor (KNN) algorithm. *J. Eng. Sci.*, *13*(8), pp.249-256.

[17] History of datascience.com. K- nearest neighbour. Retrieved From- https://www.historyofdatascience.com/k-nearest-neighbors-algorithm-classification-and-regression-star/

[18] Towardsdatascience. Maths behind knn. Retrieved From-

https://towardsdatascience.com/the-math-behind-knn-3d34050efb71

[19] Medium .com. K-NN. Retrieved From-

https://medium.com/dsckiit/all-about-k-nearest-neighbors-3027e60aad25

[20] Analyticsvidhya.com. Working. Retrieved From-

https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/#:~:text=It%20works%20by%20finding%20the,the%20field%20of%20artificial%20intelligence

[21] My educator.com. advantages and disadvantages. Retrieved From-

https://app.myeducator.com/reader/web/1421a/11/q07a0/#:~:text=Second%2C%20KNN%20can%20be%20effective,different%20than%20some%20other%20algorithms

[22] Blog.aiensured.com uses in real world. Retrieved From-

https://blog.aiensured.com/knn-algorithm-in-real-world/

[23] Alkhatib, K., Najadat, H., Hmeidi, I. and Shatnawi, M.K.A., 2013. Stock price prediction using k-nearest neighbor (kNN) algorithm. *International Journal of Business, Humanities and Technology*, *3*(3), pp.32-44.

[24] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*, CRC Press, 1984.

[25] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2009.

[26] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[27] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *JMLR*, vol. 12, pp. 2825-2830, 2011.

[28] UCI Machine Learning Repository, "Telecom Churn Dataset", [online]. Available: https://archive.ics.uci.edu/ml/datasets/customer+churn.

[29] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.

[30] Gupta, S. K., & Gupta, M. (2017). A comparison of machine learning algorithms for stock price prediction. *International Journal of Engineering and Advanced Technology*, 6(5), 1017-1023.

[31] Javatpoint. (2024, July 24). *Machine learning decision tree classification algorithm*. Retrieved from https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

[32] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: with applications in R* (2nd ed.). Springer.

[33] Scikit-learn. (2024, January 6). *Random forest regressor*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[34] GeeksforGeeks. (2024, May 22). *Random forest algorithm in machine learning*. Retrieved from https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

[35] Built In. (2024, July 24). *Random forest algorithm*. Retrieved from https://builtin.com/data-science/random-forest-algorithm

[36] Ayala, G. A., & García-Carrión, J. (2021). Stock market forecasting using the random forest and deep neural network models before and during the COVID-19 period. *Frontiers in Environmental Science*, 9, 717047.

[37] Analytics Vidhya. (2021, October). *An Introduction to Random Forest Algorithm for Beginners*. Retrieved from https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-random-forest-algorithm-for-beginners/

[38] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), 1189-1232.

[39] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 785-794.

[40] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting framework. Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS), 3155-3164.

[41] GeeksforGeeks. (2019). *ML | XGBoost (Extreme Gradient Boosting)*. Retrieved from https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/

[42] Liu, W., Zhang, L., & Wang, X. (2019). Stock price prediction based on gradient boosting machine. Neural Computing and Applications, 31(1), 197-207.

[43] Li, Y., & Liu, H. (2018). Stock price prediction based on gradient boosting machine and LSTM. Journal of Physics: Conference Series, 1014(1), 012017.

[44] Zhang, Y., & Zhang, C. (2019). Stock price prediction using gradient boosting machine with feature selection. Journal of Financial Data and Analytics, 2(2), 113-129.

[45] Galton, F. (1886). "Regression Towards Mediocrity in Hereditary Stature." *The Journal of the Anthropological Institute of Great Britain and Ireland.*

[46] Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space." *Philosophical Magazine.*

[47] Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes.* Paris: Firmin Didot.

[48] Gauss, C. F. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientum.*

[49] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning.* Springer.