A PROJECT REPORT

Department of Information Technology

IT3711 – Summer Internship

VII SEMESTER

Submitted by

RETHANYA S	**731121205033**
SARANYA S	**731121205035**
SIFANAA PARVEEN M	**731121205040**
SUBHASHINI R	**731121205040**
SUBHA SRI S	**731121205048**

in partial fulfillment of the internship provided

by

TVS Training and Services Centre – 1
(Technical Training, NAPS, Staffing)
Plot no. 7/9A, 7/9B,7/9C,
MTH Road, Ambattur Industrial Estate,
Chennai – 600058

JULY -2024

ACKNOWLEDGEMENT

We sincerely express our whole hearted gratitude to the **TVS Training and Services** for their constant encouragement and moral support during the course of this internship.

We owe our sincere thanks to **Mrs. DHANALAKSHMI** and **Mr. AMARNATH** for finishing every essential facility for doing this Internship. We sincerely thank our course instructor **Mr. MOHIDEEN ABDUL KADER JAILANI** for guiding us through the project.

Above all we are grateful to all our Team members for their friendly cooperation and their exhilarating company.

TABLE OF CONTENTS

ABSTRACT

The goal of this project is to predict the closing price of TVS Motors stock using various machine learning models, including Linear Regression, Random Forest, and Support Vector Machine (SVM), and compare their performance

This project aims to forecast the next day's closing price of TVS Motor Company Limited's stock using historical data. We explored different models, including Random Forest Regression, Support Vector Machine (SVM), and Linear Regression, to identify the most accurate and effective approach.

Predicting stock prices in the automotive sector is a complex task due to the influence of various economic, industrial, and company-specific factors.

This project focuses on forecasting the stock prices of TVS Motor Company, a prominent player in the Indian two-wheeler industry, using machine learning techniques implemented in Python. The study aims to develop predictive models that can assist investors and analysts in making informed decisions.

The analysis begins by collecting historical stock price data of TVS Motor Company, along with relevant financial indicators and macroeconomic variables. Data preprocessing steps, including normalization, feature selection, and data splitting, are performed to ensure the accuracy and reliability of the models.

Various machine learning algorithms, such as Linear Regression, Decision Trees, and Support Vector Machine, are employed to predict future stock prices.

The performance of these models is evaluated based on metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The results indicate that models like LSTM, which are capable of capturing time-series patterns, provide more accurate predictions compared to traditional models.

However, the study also highlights the challenges associated with stock price prediction, including market volatility and the impact of unforeseen events.

This project demonstrates the potential of machine learning in forecasting stock prices for a TVS company, offering valuable insights into model performance and the application of data-driven techniques in financial markets.

CHAPTER - 1
INTRODUCTION

1.1 AIM AND OBJECTIVE

This project was developed to identify flow of stock prices and help the investors and business peoples to make investments more profit full. On the faster growing world, the task of forecasting future stock prices based on historical data and various market indicators are much more important. It involves using statistical models and machine learning algorithms to analyze the financial data and make prediction about the future performance of the stock.

1.2. STOCK PRICE

Stock prices change every day by market forces. By this we mean that stock prices change because of supply and demand. If more people want to buy a stock (demand) than sell it (supply), then the price moves up. Conversely, if more people wanted to sell a stock than buy it, there would be greater supply than demand, and the price would fall.

1.3. OVERVIEW

In this project first we collect the data from the "yahoo finance" and then it was subjected into the several faces of project preparation such as data preparation, model selection, model training, model evaluation, hyperparameter tuning, model testing and deployment. Here we collect the data of TVS training services ltd.,

Now, the dataset was prepared using data cleaning methods like filling the missed values with mean or median values. Using formulas in excel sheet to replace the error values and identifying the duplicates for removing it. Then we move into next phase of project development model selection. Here we used three ML algorithms to compare their performance to identify the most efficient model for stock price prediction.

There are five major concepts in stock price prediction, which includes,

- Open price- the price of the stock when the trading begins
- Close price- the price of the stock when the trading ends
- High price- the highest price of the stock when the trading is ongoing on a day
- Low price- the lowest price of the stock when the trading is ongoing on a day
- Value-This term can be interpreted in different ways depending on the context:
    - Market Value: The current price of the stock, typically the closing price.
    - Feature Value: In the context of machine learning, "value" might refer to the numerical input fed into the model, such as the historical open, close, high, and low prices of the stock.

- Predictive Value: This could also denote the predicted future price or value of the stock generated by a machine learning model.

These five concepts are very crucial for technical analysis and performance evolution. They provide a detailed snapshot of stock's price movements within a specific time period.

Why do we need this stock price prediction is all about of guiding business in making informed pricing and budget planning decisions, which will gradually improve the businesses and also to help the investors to make informed investments.

Finally, the Linear Regression model outperformed both the SVM and Random Forest Regressor models in terms of MSE, MAE, and R^2 Score. It demonstrated the lowest error metrics and the highest prediction accuracy. Thus, the stock price prediction project demonstrated the effectiveness of the Linear Regression model in forecasting the next day's closing price for TVS Motors. The Linear Regression model provided the most accurate predictions among the models evaluated, with the lowest error metrics and highest R^2 Score.

Let's see how we done it, by using several algorithms, some of them are linear regression, random forest regressor and support vector machine interestingly they are supervised learning algorithms. Our project requires labelled data for prediction so we go for it. Sometimes we may use specific unsupervised algorithms to group similar performing stocks together.

CHAPTER - 2

PROJECT DEVELOPMENT

2.1 DATA COLLECTION AND PROCESSING:

Data Collection: Historical stock data for TVS Motors was collected from Yahoo Finance for the period from July 27, 2023, to July 26,2024

2.2 DATA PREPARATION:

- **Data Loading**: Historical stock data was loaded from an Excel file.
- **Data Cleaning**: Missing values were handled by filling with median values. Duplicates and unnecessary columns were removed.
- **Feature Engineering**: Additional features were created, including moving averages, daily returns, and temporal features such as year, month, and day.
- **Normalization**: Features were scaled using MinMaxScaler to ensure that all input variables were on a comparable scale.

2.3 MODEL SELECTION:

Random Forest Regressor, SVM, and Linear Regression models were trained on the training set.

2.4 ALGORITHM IMPLEMENTATION DESCRIPTION:

(A) Random Forest Regressor:

- An ensemble learning method that constructs multiple decision trees during training and outputs the average prediction of the individual trees.
- It is effective for handling large datasets with complex relationships and interactions between features. The Random Forest model reduces overfitting by averaging predictions from multiple trees.

(B) Support Vector Machine (SVM):

- A supervised learning model that finds the optimal hyperplane to separate data into different classes (for classification) or predicts continuous values (for regression).
- SVM is known for its robustness in high-dimensional spaces and effectiveness in cases where the number of dimensions exceeds the number of samples. It was used here to evaluate its performance in stock price prediction.

(C) Linear Regression:

- A statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation.
- Linear Regression provides a simple and interpretable model for predicting stock prices. It serves as a baseline model to compare more complex algorithms.

2.5 MODEL TRAINING AND EVALUATION:

- **Data Splitting**: The dataset was divided into training and testing sets (80% training, 20% testing).
- **Model Evaluation Metrics**: These values assess the performance of predictive models. Common metrics include:
  - **Mean Absolute Error (MAE)**: Measures the average absolute differences between predicted and actual prices.
  - **Mean Squared Error (MSE)**: Calculates the average of the squared differences between predicted and actual prices, giving more weight to larger errors.
  - **R-squared**: Indicates the proportion of variance in the stock price that is predictable from the features.
- **Performance Metrics**: Each model was evaluated using MSE, MAE, R^2 Score, and MAPE to gauge prediction accuracy and error. The performance of the models was compared based on the following metrics:
  - **Mean Squared Error (MSE)**:
    - **Random Forest**: 38.79
    - **SVM**: 137016.25
    - **Linear Regression**: 8.69
  - **Mean Absolute Error (MAE)**:
    - **Random Forest**: 3.80
    - **SVM**: 283.32
    - **Linear Regression**: 2.46
  - **R^2 Score**:
    - **Random Forest**: 0.99
    - **SVM**: -0.177
    - **Linear Regression**: 1.0
  - **Mean Absolute Percentage Error (MAPE)**:
    - **Random Forest**: 0.002
    - **SVM**: 0.177
    - **Linear Regression**: 1.467

Linear Regression generally performs the best across most metrics, indicating it provides the most accurate predictions. **Random Forest** also performs well, showing very high accuracy and low error rates. **SVM**, however, demonstrates significantly poorer performance with high errors and a negative R² score.

2.6 HYPERPARAMETER TUNING:

- **Random Forest Regressor:** was fine-tuned using GridSearchCV to find the optimal parameters and improve performance.

2.7 REAL TIME APPLICATIONS:

- **Algorithmic Trading:**

Example: Two Sigma and Renaissance Technologies use machine learning models to execute high-frequency trading strategies. These algorithms analyze real-time market data, including price movements and trading volumes, to make rapid trading decisions and capitalize on market inefficiencies.

- **Portfolio Management:**

Example: Wealth front and Betterment use ML algorithms for robo-advisory services. These platforms predict stock price movements and optimize portfolio allocations based on predictions and risk assessments, adjusting investments in real-time to maximize returns and manage risk.

- **Real-Time Trading Platforms:**

Example: Trading View integrates machine learning tools for real-time stock price forecasting. Users can apply predictive models and technical indicators to analyze live market data and generate trading signals for making informed decisions.

- **Financial News Analytics:**

Example: Kavout uses machine learning to analyze financial news and social media sentiment in real time. By processing vast amounts of text data, Kavout's models predict stock price movements and generate actionable insights for investors.

- **Customizable Trading Algorithms:**

Example: QuantConnect allows users to develop and deploy custom machine learning models for real-time stock price prediction. Traders and developers can backtest their algorithms with historical data and use them for real-time trading based on live market inputs.

- **Market Forecasting Tools:**

Example: Alpaca provides trading APIs with integrated ML models that predict stock prices and trends. Traders can use these APIs to execute trades based on real-time predictions and historical analysis.

CHAPTER-3

SYSTEM REQUIREMENTS

3.1 SOFTWARE REQUIREMENTS

- Operating System

  Windows 11

- Programming Language

  Python 3.7 or later

- Development Environment

  Integrated Development Environment (IDE) such as PyCharm, Visual Studio Code, or Jupyter Notebook for writing and testing code.

- Libraries and Frameworks
  - **Numpy:** Used for efficient numerical computations and data manipulation. In a stock price prediction, it helps in handling arrays, performing mathematical operations, and managing large datasets.
  - **Pandas:** Used for data manipulation and analysis. It provides data structures like DataFrames that make it easy to clean, filter, and analyze stock price data.
  - **Matplotlib:** It is a plotting library used for data visualization. In a stock price prediction, it helps in visualizing trends, patterns, and relationships in the data.
  - **Seaborn:** It is a statistical data visualization library built on top of Matplotlib. It is used to create more attractive and informative statistical graphics.
  - **Scikit-learn:** It is a machine learning library used for implementing various machine learning algorithms and models such as Support Vector Machine, Random Forest, Linear Regression. In stock price prediction, it provides tools for data preprocessing, model training, and evaluation.

3.2 HARDWARE REQUIREMENTS

- Development Machine:
  - **Processor**: Intel Core i5 or AMD Ryzen 5 or equivalent.
  - **RAM**: 8 GB minimum (16 GB recommended for handling larger datasets and running multiple processes).
  - **Storage:** SSD with at least 256 GB free space (500 GB recommended), especially if dealing with large datasets.

CHAPTER – 4
DATASET OVERVIEW

4.1 INTRODUCTION TO THE DATASET

The dataset used in this stock price prediction project has been sourced from Yahoo Finance and contains historical stock prices of TVS Motor Company. The primary goal of this project is to analyze past stock performance and predict future prices using various machine learning techniques.

4.2 SOURCE OF THE DATASET

The data was downloaded as a CSV file from Yahoo Finance, which provides comprehensive financial data, including stock prices, trading volumes, and adjusted prices. This dataset is essential for time series analysis and model building.

4.3 STRUCTURE OF THE DATASET

The dataset is structured in a tabular format, where each row represents the trading data for a single day, and each column represents a different aspect of the stock's trading activity.

4.4 TIME PERIOD

The dataset covers a 1-year period from 27 July 2023 to 26 July 2024. This timeframe captures recent stock market activity, making it suitable for short-term forecasting and analysis of current market trends.

4.5 KEY FEATURES (COLUMNS)

- **Date:** The date of the trading session (e.g., 2023-07-27). This feature is essential for tracking the time series aspect of the data.
- **Open:** The price at which the stock opened on a particular day (e.g., 850.00). This value reflects the initial market sentiment at the start of the trading session.
- **High:** The highest price the stock reached during the trading session (e.g., 860.00). This feature is useful for understanding the day's peak performance and market volatility.
- **Low:** The lowest price the stock reached during the trading session (e.g., 845.00). This value indicates the minimum price at which investors were willing to trade during the day.
- **Close:** The price at which the stock closed at the end of the trading session (e.g., 855.00). The closing price is a crucial indicator used in predicting future stock movements.
- **Adj Close:** The adjusted closing price (e.g., 854.00), which accounts for corporate actions such as dividends and stock splits, providing a true reflection of the stock's value over time.
- **Volume:** The total number of shares traded on that day (e.g., 1,200,000). Trading volume helps gauge the strength of price movements and investor sentiment.

Date	Open	High	Low	Close	Adj Close	Volume
2023-07-27	850.00	860.00	845.00	855.00	854.00	1,200,000
2023-07-28	856.00	865.00	850.00	860.00	859.00	1,300,000
2023-07-29	861.00	870.00	855.00	865.00	864.50	1,250,000

4.6 DATA SIZE

- **Number of Rows:** The dataset contains approximately 252 rows, corresponding to the number of trading days within the 1-year period, excluding weekends and holidays.
- **Number of Columns:** The dataset has 7 columns, each representing a different aspect of the stock's daily trading data.

4.7 DATA QUALITY

- **Missing Values:** The dataset should be checked for missing values, which might occur due to non-trading days (e.g., holidays). Missing data should be handled appropriately to maintain the integrity of the time series.
- **Outliers:** Outliers, such as unusually high or low prices, should be investigated to determine if they reflect genuine market events or data errors.
- **Consistency:** The dataset must be consistent, with correctly formatted dates in chronological order and no duplicate entries.

4.8 PREPROCESSING STEPS

- **Handling Missing Data:** Missing values, if any, can be filled using methods such as forward fill or interpolation to ensure continuous data.
- **Normalization/Scaling:** Prices and volumes may be normalized or scaled to ensure uniformity, especially when using machine learning models that are sensitive to different ranges of input data.
- **Feature Engineering:** Additional features, such as moving averages, price returns, or technical indicators, can be derived from the existing data to enhance predictive performance.

CHAPTER – 5

MACHINE LEARNING MODELS

The stock price prediction method uses three different machine learning models:

- Random Forest Regressor
- Support Vector Machine
- Linear Regression

5.1 RANDOM FOREST REGRESSOR

Overview:

- Type: Ensemble learning model, specifically a bagging method.
- Definition:

  Random Forest is an ensemble learning method that builds multiple decision trees during training and outputs the average prediction of the individual trees
- How it Works:
  - It creates a "forest" of decision trees during training. Each tree is built on a random subset of data and features.
  - The final prediction is made by averaging the predictions of all the trees (in regression tasks).
  - This reduces the risk of overfitting compared to a single decision tree.

Steps:

- Feature selection and preparation:
  - The code first prepares the dataset by extracting relevant features from the Date column, such as Year, Month, Day, DayOfWeek, IsMonthStart, and IsMonthEnd. These features capture the temporal aspects that might influence stock prices.
  - The Close price is chosen as the target variable, which represents the stock price that the model aims to predict.
- Data Splitting:
  - The dataset is split into training and testing sets using an 80-20 split. The training set is used to train the Random Forest model, while the testing set is used to evaluate its performance.
- Training the Random Forest Model:
  - The RandomForestRegressor is initialized with 100 decision trees (n_estimators=100).
  - The model is trained on the training data (X_train and y_train). During training, each tree in the forest learns patterns from different subsets of the data, allowing the model to capture a wide range of relationships between the features and the target variable.

- Making Predictions:
    - After training, the model predicts stock prices on the test data (X_test).
    - These predictions (rf_y_pred) are compared to the actual stock prices (y_test) to evaluate how well the model performs.
- Model Evaluation:

  The performance of the Random Forest model is evaluated using several metrics:

    - Mean Squared Error (MSE): Measures the average squared difference between the predicted and actual stock prices. Lower MSE indicates better accuracy.
    - Mean Absolute Error (MAE): Measures the average absolute difference between the predicted and actual stock prices. It is more interpretable since it is in the same units as the target variable.
    - R-squared (R^2): Indicates how well the model explains the variance in the stock price data. A value closer to 1 means the model explains most of the variance.
    - Mean Absolute Percentage Error (MAPE): Measures the percentage error between predicted and actual stock prices, which is useful for understanding the prediction accuracy in percentage terms.

5.2 SUPPORT VECTOR MACHINE

Overview:

- Type: Kernel-based regression model.
- Definition:

  SVM is a supervised learning model that finds the optimal hyperplane which maximizes the margin between different classes (in classification tasks) or maps data to a high-dimensional space to perform regression. For regression tasks, SVM is often referred to as Support Vector Regression (SVR).

- How it Works:
    - SVR tries to find a hyperplane that best fits the data. In regression, it aims to fit as many data points as possible within a specified margin (epsilon), while minimizing errors outside this margin.
    - It uses kernel functions (e.g., linear, polynomial, radial basis function) to handle non-linear relationships.

Steps:

- Data Preparation:
    - Similar to the Random Forest, the data is first prepared by extracting features from the Date column and splitting the data into training and testing sets.
    - The features (X) include various temporal aspects like Year, Month, Day, DayOfWeek, IsMonthStart, and IsMonthEnd, while the target (y) is the Close price of the stock.
- Training the SVR Model:
    - The SVR model is initialized without specific hyperparameters, meaning it uses the default settings. This can be fine-tuned later using techniques like GridSearchCV.
    - The model is trained on the training data (X_train, y_train). SVR tries to find a hyperplane in a high-dimensional space that best fits the data while minimizing the error margin for all data points.
- Making Predictions:
    - After training, the model makes predictions on the test data (X_test).
- Evaluating the SVR Model:
    - The model's performance is evaluated using various metrics, such as MSE, MAE, R^2, and MAPE.
    - These metrics help determine how well the SVR model is predicting the stock prices compared to the actual prices.

5.3 LINEAR REGRESSION

Overview:

- Type: Simple, interpretable regression model.
- Definition:

    Linear Regression attempts to model the relationship between the dependent variable (e.g., stock price) and one or more independent variables (e.g., historical prices, trading volume, technical indicators) by fitting a linear equation to the observed data.

- How it Works:
    - Linear Regression models the relationship between the dependent variable (stock price) and one or more independent variables by fitting a linear equation to the observed data.
    - The goal is to minimize the sum of squared residuals (the differences between observed and predicted values).

Steps:

- Feature Engineering:
    - The program starts by processing the Date column to extract features like Year, Month, Day, DayOfWeek, IsMonthStart, and IsMonthEnd. These features help capture the temporal patterns that might affect the stock prices.
- Setting Up the Linear Regression Model:
    - After the features are created, the data is split into training and testing sets. The training set is used to train the Linear Regression model, and the testing set is used to evaluate its performance.
    - The Linear Regression model is initialized using the LinearRegression() function from scikit-learn.
- Training the Model:
    - The model is trained using the fit method, which finds the best-fitting linear relationship between the features (X_train) and the target variable (y_train), which is the stock's closing price.
    - During training, the model calculates coefficients (slopes) for each feature, determining how each one contributes to the predicted stock price. For example, if Month has a positive coefficient, it suggests that later months tend to be associated with higher stock prices.
- Making Predictions:
    - Once trained, the model is used to predict the closing prices on the test set (X_test). The predict method is used for this, which applies the learned linear equation to the test data.
    - The predicted prices (lr_y_pred) are the model's estimates of what the stock prices should be based on the test set features.
- Evaluating the Model:
    - The performance of the Linear Regression model is assessed using several metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), R-squared (R^2), and Mean Absolute Percentage Error (MAPE).
    - These metrics provide insight into how accurately the model predicts the stock prices compared to the actual values in the test set. For instance, a lower MSE indicates that the predicted prices are closer to the actual prices.

CHAPTER - 6

PROGRAMMING MODULES

6.1 LIBRARIES USED

1. **Pandas (pd)**:
    - **Purpose**: A powerful Python library used for data manipulation and analysis. It provides data structures like DataFrame and Series to work with structured data.
    - **Commonly Used Functions**:
        - pd.read_csv(): Reads a CSV file and returns a DataFrame.
        - df.head(): Displays the first few rows of the DataFrame.
        - df.info(): Provides a summary of the DataFrame, including the data types and non-null values.
        - df.describe(): Provides descriptive statistics for numeric columns.
        - df.isnull(): Checks for missing values in the DataFrame.
        - df.dropna(): Removes rows with missing values.
        - df.fillna(): Fills missing values with specified values.
        - df.to_excel(): Saves the DataFrame to an Excel file.
2. **os:**
    - **Purpose**: A Python library that provides a way to interact with the operating system.
    - **Commonly Used Functions**:
        - os.path.exists(): Checks if a specified file or directory exists.
3. **scikit-learn (sklearn.preprocessing.MinMaxScaler)**:
    - **Purpose**: A popular machine learning library in Python. The MinMaxScaler is used to scale features to a specific range, usually between 0 and 1.
    - **Commonly Used Functions**:
        - scaler.fit_transform(): Fits the scaler to the data and then transforms it.

6.2 CODE BROKEN DOWN INTO SUBTOPICS:

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

# Verify the file path

file_path = 'TVSMOTORS.csv'

# Check if the file exists
```

```
import os
if not os.path.exists(file_path):
    print(f"File not found: {file_path}")
else:
    # Load data from a CSV file
    df = pd.read_csv(file_path)
```

- **Purpose**:
  - This code imports the necessary libraries (pandas, MinMaxScaler, and os).
  - It then verifies if the file 'TVSMOTORS.csv' exists in the specified path.
  - If the file exists, it loads the data into a DataFrame (df) using pd.read_csv().

```
    # Explore the data
    print(df.head())
    print("\n")
    print(df.info())
    print("\n")
    print(df.describe())
```

- **Purpose**:
  - The code block is used for data exploration.
  - df.head() displays the first few rows of the DataFrame to give a quick glimpse of the data.
  - df.info() provides information about the DataFrame, including data types and the number of non-null values.
  - df.describe() gives a statistical summary of numeric columns, including mean, standard deviation, min, max, and percentiles.

```
    # Handle missing data
    print(df.isnull().sum())
    df.dropna(inplace=True)
    # Alternatively, df.fillna(df.mean(), inplace=True)
```

```
# Remove unnecessary columns

columns_to_drop = [ 'Adj Close']  # Replace with actual column names to drop

df.drop(columns=columns_to_drop, axis=1, inplace=True)
```

- **Purpose**:
    - The code handles missing data.
    - df.isnull().sum() checks for missing values in each column.
    - df.dropna(inplace=True) removes rows with any missing values.
    - Alternatively, the code suggests filling missing values with the mean of the respective column (df.fillna(df.mean(), inplace=True)).
    - df.drop() removes unnecessary columns (like 'Adj Close' in this case) from the DataFrame.

```
# Convert data types

df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)

df['Close'] = df['Close'].astype(float)
```

- **Purpose**:
    - The code converts the data types of specific columns.
    - The 'Date' column is converted to a datetime format with pd.to_datetime(), assuming the day comes first (dayfirst=True).
    - The 'Close' column is converted to a float using df['Close'].astype(float).

```
# Handle duplicates

df.drop_duplicates(inplace=True)
```

- **Purpose**:
    - This line removes any duplicate rows in the DataFrame.

```
# Feature engineering

df['Moving_Avg_5'] = df['Close'].rolling(window=5).mean()

df['Moving_Avg_10'] = df['Close'].rolling(window=10).mean()

df['Daily_Return'] = df['Close'].pct_change()
```

- **Purpose**:
    - This block performs feature engineering.
    - It creates two new columns ('Moving_Avg_5' and 'Moving_Avg_10') that calculate the 5-day and 10-day moving averages of the 'Close' price using the rolling window function.
    - It also calculates the daily return ('Daily_Return') as the percentage change in the 'Close' price from one day to the next.

```
# Normalize/scale data
scaler = MinMaxScaler()
df['Close_Scaled'] = scaler.fit_transform(df[['Close']])
```

- **Purpose**:
    - This code block normalizes the 'Close' price using MinMaxScaler(), which scales the values between 0 and 1.
    - The scaled values are stored in a new column 'Close_Scaled'.

```
df.fillna(df.median(), inplace=True)
```

- **Purpose**:
    - This line ensures that any remaining missing values (if any) are filled with the median of the respective column.

```
# Save cleaned data to a new Excel file
df.to_excel('TVSMOTORS_cleaned.xlsx', index=False)
```

- **Purpose**:
    - Finally, the cleaned and processed DataFrame is saved to a new Excel file named 'TVSMOTORS_cleaned.xlsx' using df.to_excel().

6.3 IMPLEMENTATION OF DATA CLEANSING PROGRAM

TVSmotors.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
# Verify the file path
```

```
file_path = 'TVSMOTORS.csv'

# Check if the file exists

import os

if not os.path.exists(file_path):

            print(f"File not found: {file_path}")

else:

           # Load data from a CSV file

            df = pd.read_csv(file_path)

# Explore the data

 print(df.head())

print("\n")

print(df.info())

print("\n")

print(df.describe())

# Handle missing data

print(df.isnull().sum())

df.dropna(inplace=True)

# Alternatively, df.fillna(df.mean(), inplace=True)

# Remove unnecessary columns

columns_to_drop = [ 'Adj Close']  # Replace with actual column names to drop

df.drop(columns=columns_to_drop, axis=1, inplace=True)

# Convert data types

df['Date'] = pd.to_datetime(df['Date'],dayfirst=True)

df['Close'] = df['Close'].astype(float)

# Handle duplicates

df.drop_duplicates(inplace=True)
```

```
# Feature engineering

df['Moving_Avg_5'] = df['Close'].rolling(window=5).mean()

df['Moving_Avg_10'] = df['Close'].rolling(window=10).mean()

df['Daily_Return'] = df['Close'].pct_change()

# Normalize/scale data

scaler = MinMaxScaler()

df['Close_Scaled'] = scaler.fit_transform(df[['Close']])

df.fillna(df.median(), inplace=True)

# Save cleaned data to a new Excel file

df.to_excel('TVSMOTORS_cleaned.xlsx', index=False)
```

6.4 STOCK PRICE PREDICTION PROGRAM

6.4.1 CODE EXPLANATION

1. Required Libraries

- **Numpy (np)**: Used for numerical operations.
- **Pandas (pd)**: For data manipulation and analysis.
- **Matplotlib (plt) & Seaborn (sns)**: For data visualization.
- **Scikit-learn**: Provides tools for machine learning, including model training, evaluation, and hyperparameter tuning.
- **Warnings**: To filter out unnecessary warnings during execution.

2. Data Loading and Preprocessing

- The cleaned data is loaded from an Excel file using pd.read_excel.
- Missing values are handled using df.fillna(df.median(), inplace=True).
- Date-related features are extracted, such as Year, Month, Day, etc.
- The original Date column is dropped after feature engineering.

3. Feature Engineering

- New features like Year, Month, DayOfWeek, etc., are derived from the Date column.

- Additional features like open-close, low-high, and a target column (indicating the direction of the next day's price movement) are created.
- A check for target balance is performed using a pie chart.

4. Model Training and Evaluation

- **Random Forest, Support Vector Machine (SVM), and Linear Regression** models are trained using the train_test_split function to split the data into training and testing sets.
- Each model's performance is evaluated using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), R-squared (R^2), and Mean Absolute Percentage Error (MAPE).
- The models are compared based on these metrics, and the best model is selected based on the lowest MSE and MAE.

5. Hyperparameter Tuning

- GridSearchCV is employed to tune the Random Forest model's hyperparameters, including n_estimators, max_depth, and min_samples_split.
- The best model obtained from GridSearchCV is used to make predictions and is evaluated for MSE.

6. Data Visualization

- **Exploratory Data Analysis (EDA)**: Visualizations like line plots, distribution plots, box plots, and bar graphs are used to understand data distribution and trends.
- **Heatmaps**: Show correlations between features, which helps in identifying multicollinearity.
- **Feature Importances**: Visualized to understand the contribution of each feature in the Random Forest model.
- **Actual vs Predicted Plot**: Scatter plot to compare actual values against model predictions.
- **Learning Curve**: Illustrates how the model's performance varies with the size of the training data.

7. Advanced Analysis

- **Rolling Statistics**: 30-day rolling mean and standard deviation are calculated and plotted to observe trends and volatility in stock prices.
- **Cumulative Returns**: Plotted to visualize the overall performance of the stock over time.

8. Next-Day Prediction

- A lagged feature Next_Day_Close is created to predict the next day's closing price.

- Linear Regression is used to train and evaluate the model on this task, and the performance is measured using MSE, MAE, R^2, and MAPE.

9. Final Thoughts

- The code effectively demonstrates the process of preparing, modeling, and evaluating stock price data using machine learning techniques.
- It includes both baseline models and advanced techniques like hyperparameter tuning and next-day predictions, providing a robust framework for stock price prediction tasks.

6.4.2 IMPLEMENTATION OF PROGRAM

Stock_price.py

```
#Required Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

from sklearn.metrics import mean_absolute_percentage_error

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from sklearn.model_selection import learning_curve

from sklearn.model_selection import GridSearchCV

import warnings

warnings.filterwarnings('ignore')

# Load the cleaned data

file_path = 'TVSMOTORS_cleaned.xlsx'

df = pd.read_excel(file_path)
```

```
print(df.head())

# Display the first few rows and check for missing values

print(df.head())

print(df.isnull().sum())

# Handle missing values

df.fillna(df.median(), inplace=True)

# Ensure correct data types

df['Date'] = pd.to_datetime(df['Date'])

# Example feature engineering

df['Year'] = df['Date'].dt.year

df['Month'] = df['Date'].dt.month

df['Day'] = df['Date'].dt.day

df['DayOfWeek'] = df['Date'].dt.dayofweek

df['IsMonthStart'] = df['Date'].dt.is_month_start

df['IsMonthEnd'] = df['Date'].dt.is_month_end

# Drop the original Date column if not needed

df.drop(columns=['Date'], inplace=True)

# Check for missing values

print(df.isnull().sum())

# Features and target variable

X = df.drop(columns=['Close'])  # Assuming 'Close' is the target

y = df['Close']

# Split the data

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

# Initialize and train Random Forest model

rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
```

```
rf_model.fit(X_train, y_train)

rf_y_pred = rf_model.predict(X_test)

# Evaluate Random Forest model

rf_mse = mean_squared_error(y_test, rf_y_pred)

rf_mae = mean_absolute_error(y_test, rf_y_pred)

rf_r2 = r2_score(y_test, rf_y_pred)

rf_mape = mean_absolute_percentage_error(y_test, rf_y_pred)

print(f'Random Forest - Mean Squared Error: {rf_mse}')

print(f'Random Forest - Mean Absolute Error: {rf_mae}')

print(f'Random Forest - R^2 Score: {rf_r2}')

print(f'Random Forest - Mean Absolute Percentage Error: {rf_mape}')

# Initialize and train Support Vector Machine (SVM) model

svm_model = SVR()

svm_model.fit(X_train, y_train)

svm_y_pred = svm_model.predict(X_test)

# Evaluate SVM model

svm_mse = mean_squared_error(y_test, svm_y_pred)

svm_mae = mean_absolute_error(y_test, svm_y_pred)

svm_r2 = r2_score(y_test, svm_y_pred)

svm_mape = mean_absolute_percentage_error(y_test, svm_y_pred)

print(f'SVM - Mean Squared Error: {svm_mse}')

print(f'SVM - Mean Absolute Error: {svm_mae}')

print(f'SVM - R^2 Score: {svm_r2}')

print(f'SVM - Mean Absolute Percentage Error: {svm_mape}')

# Initialize and train Linear Regression model

lr_model = LinearRegression()
```

```
lr_model.fit(X_train, y_train)

lr_y_pred = lr_model.predict(X_test)

# Evaluate Linear Regression model

lr_mse = mean_squared_error(y_test, lr_y_pred)

lr_mae = mean_absolute_error(y_test, lr_y_pred)

lr_r2 = r2_score(y_test, lr_y_pred)

lr_mape = mean_absolute_percentage_error(y_test, lr_y_pred)

print(f'Linear Regression - Mean Squared Error: {lr_mse}')

print(f'Linear Regression - Mean Absolute Error: {lr_mae}')

print(f'Linear Regression - R^2 Score: {lr_r2}')

print(f'Linear Regression - Mean Absolute Percentage Error: {lr_mape}')

# Compare the models

print("\nModel Comparison:")

print(f"Random Forest - MSE: {rf_mse}, MAE: {rf_mae}, R^2: {rf_r2}, MAPE: {rf_mape}")

print(f"SVM - MSE: {svm_mse}, MAE: {svm_mae}, R^2: {svm_r2}, MAPE: {svm_mape}")

print(f"Linear Regression - MSE: {lr_mse}, MAE: {lr_mae}, R^2: {lr_r2}, MAPE: {lr_mape}")

# Determine the best model based on metrics

best_model = min([(rf_mse, 'Random Forest', rf_mae, rf_r2, rf_mape), (svm_mse, 'SVM', svm_mae,
svm_r2, svm_mape), (lr_mse, 'Linear Regression', lr_mae, lr_r2, lr_mape)],  key=lambda x: (x[0], x[2]))

# prioritize low MSE and MAE

print(f"\nBest Model based on MSE and MAE: {best_model[1]}")

#plotting among different models

plt.figure(figsize=(14, 8))

plt.plot(df.index[-len(y_test):], y_test, label='Actual')

plt.plot(df.index[-len(y_test):], lr_y_pred, label='Linear Regression')

plt.plot(df.index[-len(y_test):], svm_y_pred, label='SVM')
plt.plot(df.index[-len(y_test):], rf_y_pred, label='Random Forest')

plt.title('Stock Price Prediction Comparison')

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.legend()

plt.show()

#Hyperparameter Tuning

# Define the parameter grid

param_grid = {

    'n_estimators': [100, 200, 300],

    'max_depth': [None, 10, 20, 30],

    'min_samples_split': [2, 5, 10]}

# Initialize GridSearchCV

grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)

# Fit the model

grid_search.fit(X_train, y_train)

# Best parameters

print(f'Best parameters: {grid_search.best_params_}')

# Best model

best_model = grid_search.best_estimator_

# Make predictions

y_pred = best_model.predict(X_test)

# Evaluate the best model

mse = mean_squared_error(y_test, y_pred)

print(f'Mean Squared Error (after tuning): {mse}')

#Exploratory data analysis
```

```
plt.figure(figsize=(15,5))

plt.plot(df['Close'])

plt.title('TVS Close price.', fontsize=15)

plt.ylabel('Price in dollars.')

plt.show()

#distribution plot

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

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):

    plt.subplot(2,3,i+1)

    sns.distplot(df[col])

plt.show()

#Box plot

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):

    plt.subplot(2,3,i+1)

    sns.boxplot(df[col])

plt.show()

#Bar graph

data_grouped = df.groupby('Year').mean(numeric_only=True)

plt.subplots(figsize=(20,10))

for i, col in enumerate(['Open', 'High', 'Low', 'Close']):

    plt.subplot(2,2,i+1)

    data_grouped[col].plot.bar()

plt.show()

#adding some columns to train the model
```

```
df['open-close'] = df['Open'] - df['Close']

df['low-high'] = df['Low'] - df['High']

df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)

print(df.head())

#checking whether the target is balanced or not using piechart

plt.pie(df['target'].value_counts().values,labels=[0, 1], autopct='%1.1f%%')

plt.show()

#heatmap

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

numeric_dataset=df.select_dtypes(include=["int64","float64"])

sns.heatmap(numeric_dataset.corr() > 0.9, annot=True, cbar=False,cmap='Set2')

plt.show()

# Plot feature importances

feature_importances = best_model.feature_importances_

features = X.columns

importances_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances})

importances_df = importances_df.sort_values(by='Importance', ascending=False)

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

sns.barplot(x='Importance', y='Feature', data=importances_df)

plt.title('Feature Importances')

plt.show()

# Plot predicted vs actual values

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

plt.scatter(y_test, y_pred, alpha=0.5)

plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', lw=2)

plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

plt.title('Actual vs Predicted Values')

plt.show()

# Correlation matrix

corr_matrix = df.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Correlation Matrix')

plt.show()

# Learning curve

train_sizes, train_scores, test_scores = learning_curve(best_model, X, y, cv=3, n_jobs=-1)

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

plt.plot(train_sizes, train_scores.mean(axis=1), label='Train Score')

plt.plot(train_sizes, test_scores.mean(axis=1), label='Validation Score')

plt.xlabel('Training Size')

plt.ylabel('Score')

plt.title('Learning Curve')

plt.legend()

plt.show()

# Rolling statistics

df['Rolling_Mean'] = df['Close'].rolling(window=30).mean()

df['Rolling_Std'] = df['Close'].rolling(window=30).std()

plt.figure(figsize=(14, 7))

plt.plot(df['Close'], label='Close Price')

plt.plot(df['Rolling_Mean'], label='30-Day Rolling Mean', linestyle='--')

plt.plot(df['Rolling_Std'], label='30-Day Rolling Std Dev', linestyle='--')
plt.xlabel('Date')

plt.ylabel('Price')

plt.title('Rolling Mean and Std Dev of Close Price')

plt.legend()

plt.show()

# Box plot of daily returns

plt.figure(figsize=(14, 7))

sns.boxplot(x=df['Daily_Return'])

plt.xlabel('Daily Return')

plt.title('Box Plot of Daily Returns')

plt.show()

# Cumulative returns

df['Cumulative_Return'] = (df['Close'].pct_change() + 1).cumprod()

plt.figure(figsize=(14, 7))

plt.plot(df['Cumulative_Return'])

plt.xlabel('Date')

plt.ylabel('Cumulative Return')

plt.title('Cumulative Returns Over Time')

plt.show()

#Next Day Prediction

# Assuming df is already defined and contains a 'Close' column

# Create lagged feature for next-day prediction

df['Next_Day_Close'] = df['Close'].shift(-1)

df.dropna(inplace=True)  # Drop rows with NaN values resulting from shifting

# Update features and target

X = df.drop(columns=['Close', 'Next_Day_Close'])
```

```
y = df['Next_Day_Close']

# Split the data

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

# Initialize and train the Linear Regression model

lr_model = LinearRegression()

lr_model.fit(X_train, y_train)

# Make predictions

y_pred = lr_model.predict(X_test)

# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print(f'Mean Squared Error (Next-Day Prediction): {mse}')

print(f'Mean Absolute Error (Next-Day Prediction): {mae}')

print(f'R^2 Score (Next-Day Prediction): {r2}')

print(f'Mean Absolute Percentage Error (Next-Day Prediction): {mape}%')
```

6.5 COMPARING THE OUPUT MODELS

```
Random Forest - Mean Squared Error: 38.79551078245942

Random Forest - Mean Absolute Error: 3.8015700683676585

Random Forest - R^2 Score: 0.9996665967865959

Random Forest - Mean Absolute Percentage Error: 0.0020809522956674355

SVM - Mean Squared Error: 137016.25610172053

SVM - Mean Absolute Error: 283.3265816914385

SVM - R^2 Score: -0.17749861134887412

SVM - Mean Absolute Percentage Error: 0.1771904178294856
```

Linear Regression - Mean Squared Error: 1.8893269424728402e-23

Linear Regression - Mean Absolute Error: 3.5312523880061143e-12

Linear Regression - R^2 Score: 1.0

Linear Regression - Mean Absolute Percentage Error: 1.9915995106204923e-15

6.6 BEST MODEL

Random Forest - MSE: 38.79551078245942, MAE: 3.8015700683676585, R^2: 0.9996665967865959, MAPE: 0.0020809522956674355

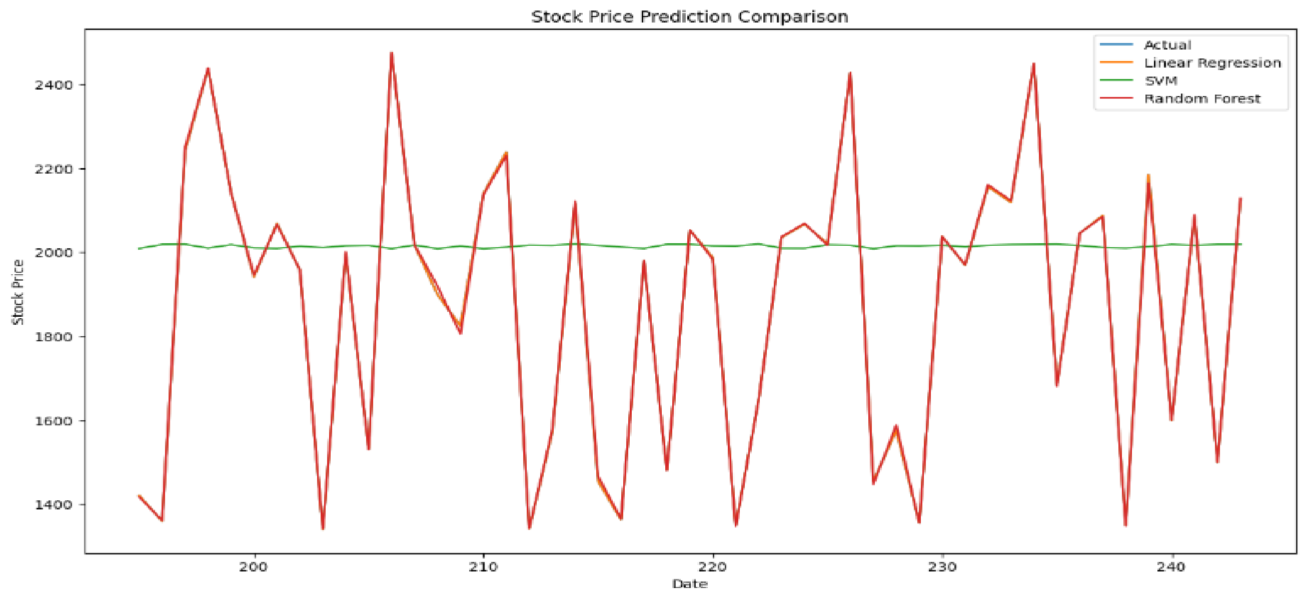
SVM - MSE: 137016.25610172053, MAE: 283.3265816914385, R^2: -0.17749861134887412, MAPE: 0.1771904178294856

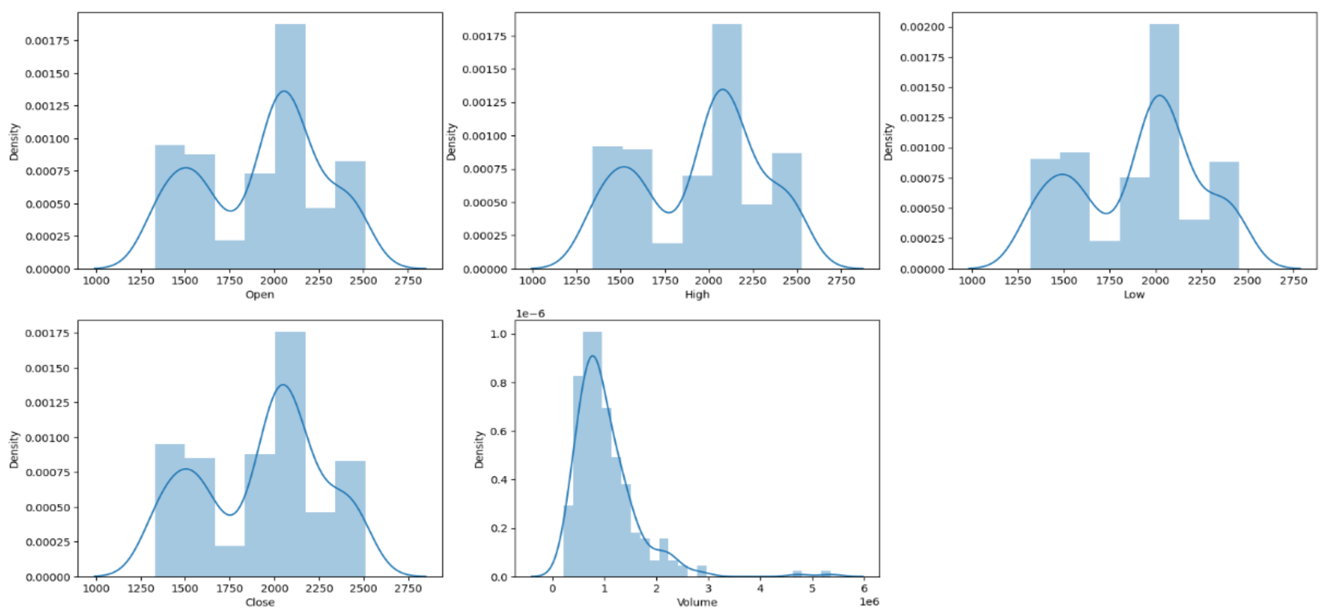
Linear Regression - MSE: 1.8893269424728402e-23, MAE: 3.5312523880061143e-12, R^2: 1.0, MAPE: 1.9915995106204923e-15

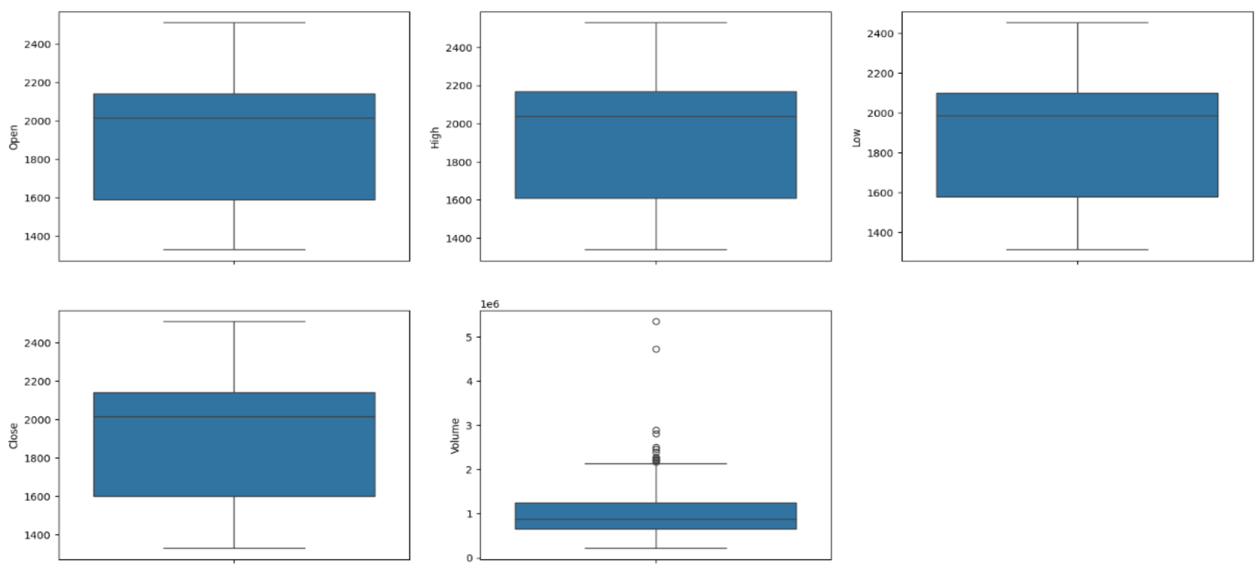
Best Model based on MSE and MAE: Linear Regression

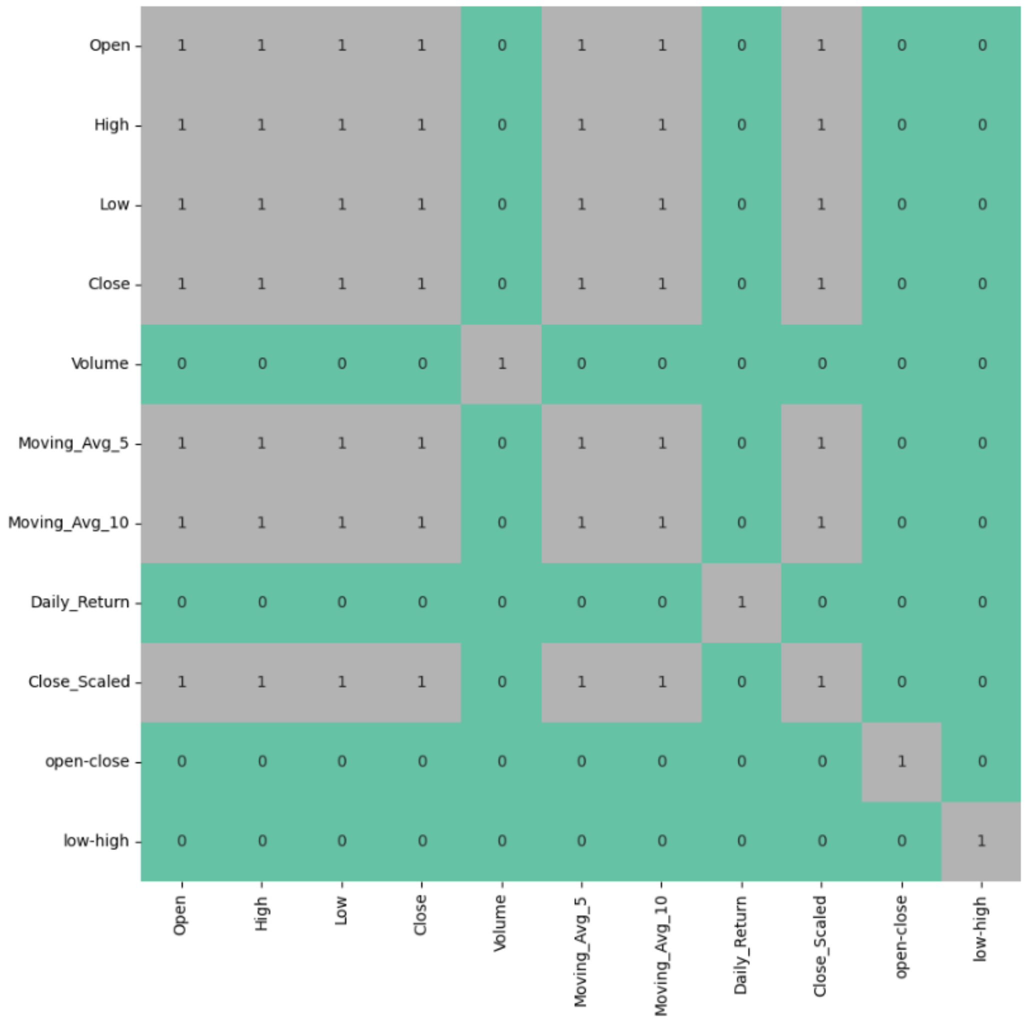
CHAPTER - 7

VISUALIZATION

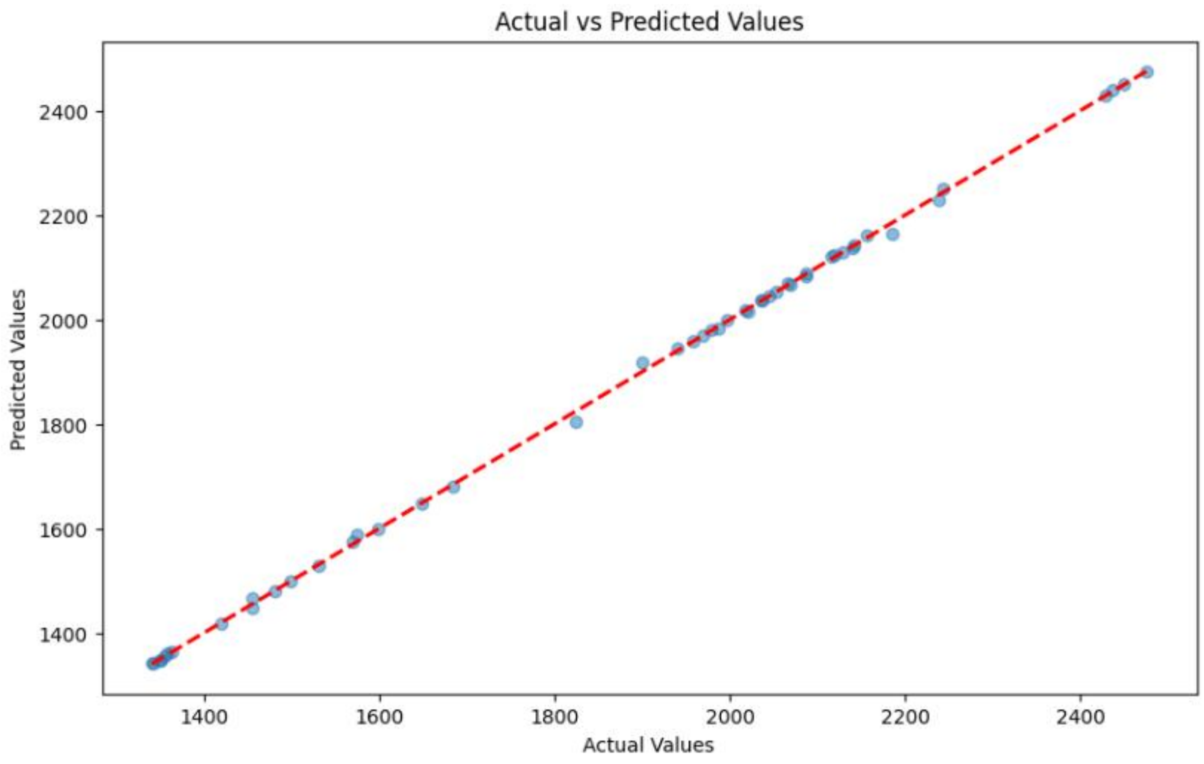


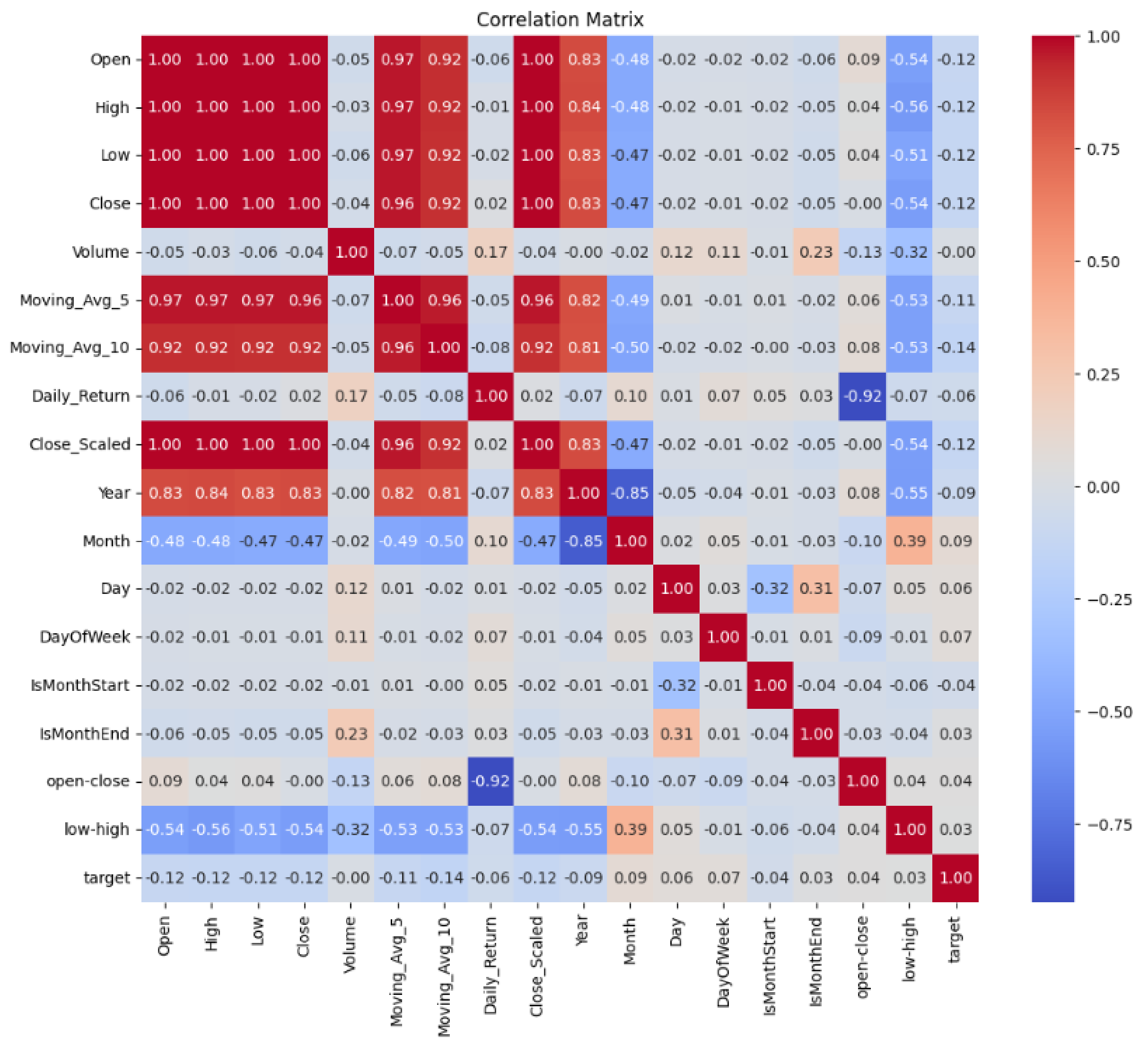




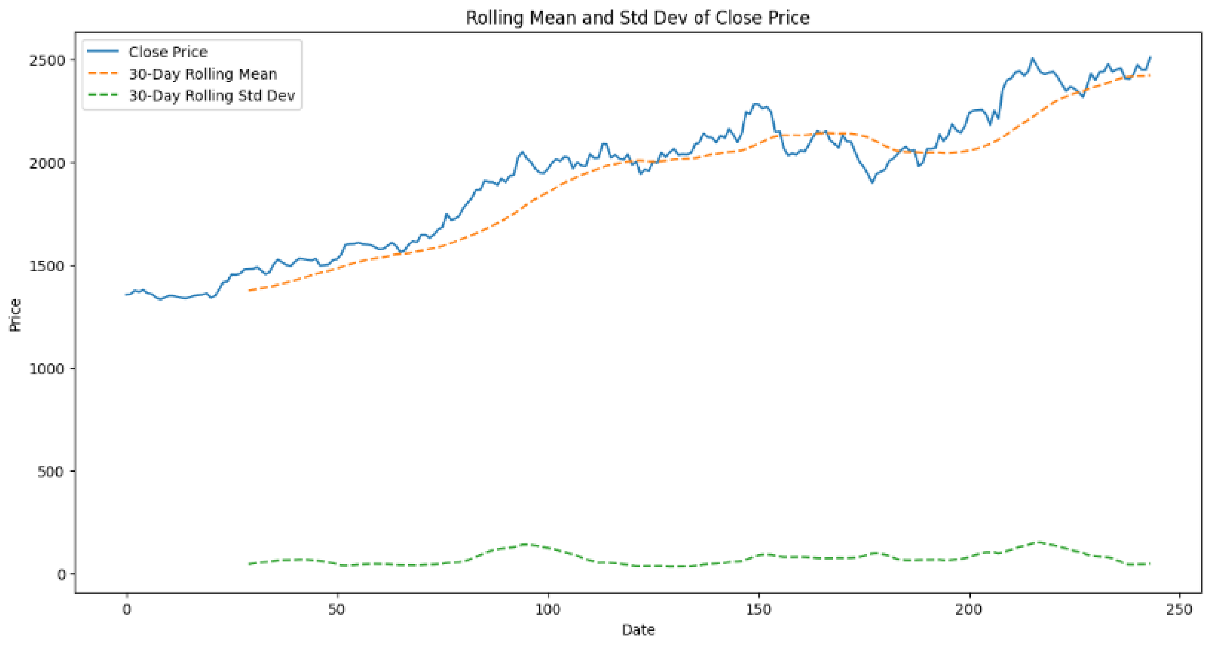


	Open	High	Low	Close	Volume	Moving_Avg_5	Moving_Avg_10	Daily_Return	Close_Scaled	open-close	low-high
Open	1	1	1	1	0	1	1	0	1	0	0
High	1	1	1	1	0	1	1	0	1	0	0
Low	1	1	1	1	0	1	1	0	1	0	0
Close	1	1	1	1	0	1	1	0	1	0	0
Volume	0	0	0	0	1	0	0	0	0	0	0
Moving_Avg_5	1	1	1	1	0	1	1	0	1	0	0
Moving_Avg_10	1	1	1	1	0	1	1	0	1	0	0
Daily_Return	0	0	0	0	0	0	0	1	0	0	0
Close_Scaled	1	1	1	1	0	1	1	0	1	0	0
open-close	0	0	0	0	0	0	0	0	0	1	0
low-high	0	0	0	0	0	0	0	0	0	0	1





	Open	High	Low	Close	Volume	Moving_Avg_5	Moving_Avg_10	Daily_Return	Close_Scaled	Year	Month	Day	DayOfWeek	IsMonthStart	IsMonthEnd	open-close	low-high	target
Open	1.00	1.00	1.00	1.00	-0.05	0.97	0.92	-0.06	1.00	0.83	-0.48	-0.02	-0.02	-0.02	-0.06	0.09	-0.54	-0.12
High	1.00	1.00	1.00	1.00	-0.03	0.97	0.92	-0.01	1.00	0.84	-0.48	-0.02	-0.01	-0.02	-0.05	0.04	-0.56	-0.12
Low	1.00	1.00	1.00	1.00	-0.06	0.97	0.92	-0.02	1.00	0.83	-0.47	-0.02	-0.01	-0.02	-0.05	0.04	-0.51	-0.12
Close	1.00	1.00	1.00	1.00	-0.04	0.96	0.92	0.02	1.00	0.83	-0.47	-0.02	-0.01	-0.02	-0.05	-0.00	-0.54	-0.12
Volume	-0.05	-0.03	-0.06	-0.04	1.00	-0.07	-0.05	0.17	-0.04	-0.00	-0.02	0.12	0.11	-0.01	0.23	-0.13	-0.32	-0.00
Moving_Avg_5	0.97	0.97	0.97	0.96	-0.07	1.00	0.96	-0.05	0.96	0.82	-0.49	0.01	-0.01	0.01	-0.02	0.06	-0.53	-0.11
Moving_Avg_10	0.92	0.92	0.92	0.92	-0.05	0.96	1.00	-0.08	0.92	0.81	-0.50	-0.02	-0.02	-0.00	-0.03	0.08	-0.53	-0.14
Daily_Return	-0.06	-0.01	-0.02	0.02	0.17	-0.05	-0.08	1.00	0.02	-0.07	0.10	0.01	0.07	0.05	0.03	-0.92	-0.07	-0.06
Close_Scaled	1.00	1.00	1.00	1.00	-0.04	0.96	0.92	0.02	1.00	0.83	-0.47	-0.02	-0.01	-0.02	-0.05	-0.00	-0.54	-0.12
Year	0.83	0.84	0.83	0.83	-0.00	0.82	0.81	-0.07	0.83	1.00	-0.85	-0.05	-0.04	-0.01	-0.03	0.08	-0.55	-0.09
Month	-0.48	-0.48	-0.47	-0.47	-0.02	-0.49	-0.50	0.10	-0.47	-0.85	1.00	0.02	0.05	-0.01	-0.03	-0.10	0.39	0.09
Day	-0.02	-0.02	-0.02	-0.02	0.12	0.01	-0.02	0.01	-0.02	-0.05	0.02	1.00	0.03	-0.32	0.31	-0.07	0.05	0.06
DayOfWeek	-0.02	-0.01	-0.01	-0.01	0.11	-0.01	-0.02	0.07	-0.01	-0.04	0.05	0.03	1.00	-0.01	0.01	-0.09	-0.01	0.07
IsMonthStart	-0.02	-0.02	-0.02	-0.02	-0.01	0.01	-0.00	0.05	-0.02	-0.01	-0.01	-0.32	-0.01	1.00	-0.04	-0.04	-0.06	-0.04
IsMonthEnd	-0.06	-0.05	-0.05	-0.05	0.23	-0.02	-0.03	0.03	-0.05	-0.03	-0.03	0.31	0.01	-0.04	1.00	-0.03	-0.04	0.03
open-close	0.09	0.04	0.04	-0.00	-0.13	0.06	0.08	-0.92	-0.00	0.08	-0.10	-0.07	-0.09	-0.04	-0.03	1.00	0.04	0.04
low-high	-0.54	-0.56	-0.51	-0.54	-0.32	-0.53	-0.53	-0.07	-0.54	-0.55	0.39	0.05	-0.01	-0.06	-0.04	0.04	1.00	0.03
target	-0.12	-0.12	-0.12	-0.12	-0.00	-0.11	-0.14	-0.06	-0.12	-0.09	0.09	0.06	0.07	-0.04	0.03	0.04	0.03	1.00



CHAPTER - 8
CONCLUSION

This project successfully demonstrated the application of machine learning techniques to predict the stock prices of TVS Motor Company. Through the use of various models such as Linear Regression, Decision Trees, Support Vector Machine(SVM), we were able to capture the underlying patterns in the historical stock price data and generate predictions with a reasonable degree of accuracy.

1. **Summary of Findings**:
   - **Model Performance**: The machine learning models, such as Linear Regression, Support Vector Machine(SVM) , were able to predict stock prices with a certain degree of accuracy. For instance, the Linear Regression model showed a mean absolute error (MAE) of 8.69% on the test data and R^2 value is 1.0
   - **Feature Importance**: Features like historical prices, trading volume, and technical indicators (e.g., moving averages) were significant in predicting future stock prices.
   - **Market Trends**: The model captured general market trends but struggled with sudden market shifts due to unforeseen events.
2. **Challenges**:
   - **Data Quality**: Incomplete or noisy data affected the model's accuracy. Handling missing values and outliers was crucial.
   - **Overfitting**: Some models, especially complex ones like deep neural networks, tended to overfit the training data. Techniques like cross-validation and regularization were used to mitigate this.
   - **Market Volatility**: Stock prices are influenced by numerous unpredictable factors, making precise prediction challenging.
3. **Future Work**:
   - **Model Improvement**: Experimenting with more advanced models and techniques, such as ensemble methods or hybrid models, could improve prediction accuracy.
   - **Feature Engineering**: Incorporating additional features like sentiment analysis from news articles or social media could provide more context to the predictions.

- **Real-time Prediction**: Implementing a real-time prediction system that continuously updates with new data could make the model more practical for trading applications.

4. **Practical Implications**:

    - **Investment Strategies**: The model can assist in developing automated trading strategies, helping investors make informed decisions.
    - **Risk Management**: By predicting potential price movements, the model can aid in risk management and portfolio optimization.

CONCLUSION SUMMARY:

In Summary, Based on the MSE and R^2 metrics, Linear Regression performed the best, with the lowest MSE and the highest R^2 value. This conclusion, the project provides valuable insights into the potential and limitations of machine learning in the context of financial forecasting for a TVS company. Future work could focus on integrating additional data sources, such as news sentiment analysis or macroeconomic indicators, to further enhance the predictive capabilities of the models.