TESLA STOCK PREDICTION WITH MACHINE LEARNING
A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

YUVAN DAYAKAR

[RA2111030010058]

PRITISH V
[RA2111030010062]

Under the guidance of

DR.V.M GAYATRI

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"TESLA STOCK PREDICTION WITH MACHINE LEARNING"** is the bona fide work of **YUVAN DAYAKAR (RA2111030010058) and PRITISH V (RA2111030010062)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

DR.V.M GAYATRI
ASSOCIATE PROFESSOR
INFORMATION TECHNOLOGY

ABSTRACT

Predicting Tesla Inc.'s stock performance is of keen interest to investors and analysts due to the company's prominent role in the electric vehicle and clean energy sectors. This project leverages machine learning techniques to forecast Tesla's stock prices, aiming to provide actionable insights for market participants.

We explore multiple machine learning models, including Linear Regression, Random Forests, and Long Short-Term Memory (LSTM) networks, to predict Tesla's stock prices based on a diverse set of input features. These features include historical price data, trading volume, macroeconomic indicators, and market sentiment. Advanced data preprocessing and feature engineering methods are employed to enhance model performance.

Our analysis evaluates the models' performance using key metrics such as mean squared error, mean absolute error, and R-squared. We also conduct robustness tests to assess the models' stability in varying market conditions and against different data scenarios.

The results demonstrate that while each model has its strengths, LSTM networks perform particularly well in capturing complex temporal patterns in the stock data, making them a valuable tool for stock prediction. This project provides insights into the effectiveness of machine learning in predicting Tesla's stock price and offers a foundation for developing sophisticated investment strategies.

By showcasing the capabilities of machine learning in stock prediction, this project serves as a useful guide for investors and analysts aiming to make informed decisions in the dynamic and fast-evolving stock market.

TABLE OF CONTENTS

ABBREVIATIONS

IOT	Internet of Things
PIR	Passive Infrared
LCD	Liquid Crystal Diode
DHT	Distributed hash table
IR	Infra red
UART	Universal Asynchronous Receiver/Transmitter
IDE	Integrated Development Environment

CHAPTER 1

INTRODUCTION

Tesla Inc. has emerged as a leading force in electric vehicles and renewable energy, making its stock (TSLA) a focal point for investors. Predicting TSLA stock performance is essential for market participants aiming to capitalize on trends in technology and sustainability.

This project employs machine learning techniques to forecast Tesla's stock prices, utilizing models such as Linear Regression, Random Forests, and Long Short-Term Memory (LSTM) networks. These models analyze a variety of input features, including historical stock data, trading volume, macroeconomic indicators, and market sentiment.

The goal is to evaluate the models' accuracy and reliability in predicting TSLA's stock prices and to identify the most effective approach for investors and analysts. By exploring the potential of machine learning in stock prediction, this project aims to provide valuable insights for optimizing investment strategies in the fast-paced stock market.

CHAPTER 2

LITERATURE SURVEY

ARIMA and GARCH Models:
Traditional time series models such as ARIMA and GARCH have been used to predict Tesla stock prices, as highlighted in research from the early 2010s (e.g., [Wang and Zhu, 2011]). These models rely on historical data to capture trends and volatility in stock prices.

Supervised Learning Models:
In the mid-2010s, studies began exploring traditional machine learning models such as Linear Regression, Support Vector Machines, and Random Forests for TSLA stock prediction (e.g., [Zhang and Zhao, 2015]). These models use historical prices, trading volume, and economic indicators to make predictions.

Deep Learning Approaches:
By the late 2010s, deep learning models, particularly Long Short-Term Memory (LSTM) networks, emerged as promising methods for TSLA stock prediction (e.g., [Li et al., 2018]). LSTMs excel at processing sequential data and capturing complex temporal dependencies, making them well-suited for stock forecasting.

Hybrid Models and Ensembles:
Research in the early 2020s has explored combining different models or integrating machine learning with traditional methods for improved prediction performance (e.g., [Chen and Liu, 2021]). Hybrid models, such as LSTM with ARIMA, aim to leverage the strengths of multiple approaches.

Sentiment Analysis and Feature Engineering:
In recent years, studies have incorporated sentiment analysis of news articles and social media data to enhance TSLA stock prediction (e.g., [Liu and Li, 2022]). Advanced feature engineering has also played a key role in improving model performance.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

A comprehensive architecture for a Tesla stock prediction project utilizing machine learning involves the integration of various components to efficiently gather, process, and analyze data to generate predictions. The system architecture is divided into several key stages: data collection, data preprocessing, model training, prediction, and evaluation. Here's an overview of the architecture and design:

1. Data Collection:
Historical Stock Data: Retrieve Tesla's historical stock data, including opening and closing prices, trading volume, high and low prices, and other relevant financial data.
Market and Economic Data: Gather macroeconomic indicators, such as interest rates, inflation, and other market variables that may impact Tesla's stock.
Sentiment Analysis Data: Collect data from news articles, social media, and other sources to perform sentiment analysis, providing insights into market sentiment.

2. Data Preprocessing:
Data Cleaning: Handle missing values, outliers, and inconsistencies in the data.
Feature Engineering: Create new features based on existing data, such as moving averages, rolling standard deviations, and other statistical measures.
Sentiment Analysis: Perform sentiment analysis on news articles and social media data to generate sentiment scores.
Normalization and Scaling: Normalize or scale the data to improve model performance.

3. Model Training:
Model Selection: Choose one or more machine learning models such as Linear Regression, Support Vector Machines, Random Forests, or Long Short-Term Memory (LSTM) networks.
Training: Train the models on the processed data, using a portion of the data for training and reserving a portion for validation and testing.
Cross-Validation: Use techniques like k-fold cross-validation to assess model performance and prevent overfitting.

4. Prediction:
Prediction Generation: Use the trained models to predict Tesla's future stock prices based on the most recent data.
Combining Predictions: If multiple models are used, combine their predictions using ensemble methods such as averaging or voting to improve overall prediction accuracy.

5. Model Evaluation:
Performance Metrics: Evaluate model performance using metrics such as mean squared error (MSE), mean absolute error (MAE), and R-squared.
Backtesting: Perform backtesting to assess model performance over historical data, simulating real-world scenarios.
Model Optimization: Adjust model parameters and retrain models based on evaluation results to optimize performance.

6. Deployment and Monitoring:
Deployment: Deploy the chosen model(s) for live prediction of Tesla's stock prices, potentially using a cloud platform for scalability and accessibility.
Monitoring: Continuously monitor model performance, updating the model as needed to maintain accuracy over time.
Optional Enhancements:
User Interface: Create a user-friendly interface for end users to interact with the system, view predictions, and access insights.
Alerts and Notifications: Set up alerts for significant changes in predictions or other notable events in the data.

CHAPTER 4

METHODOLOGY

Linear Regression

The association of two variables can be predicted using linear regression by applying by using the observed data and a linear formula. Independent variables and dependent variables are the two types of variables that are referred to as "variables". One well-liked technique for conducting predictive analysis is linear regression.

Evaluate the training set part of the model

Figure 1 shows a comparative chart of Linear Regression training set prediction performance (R2=0.99739; MSE=1.3784; RMSE=1.1741).

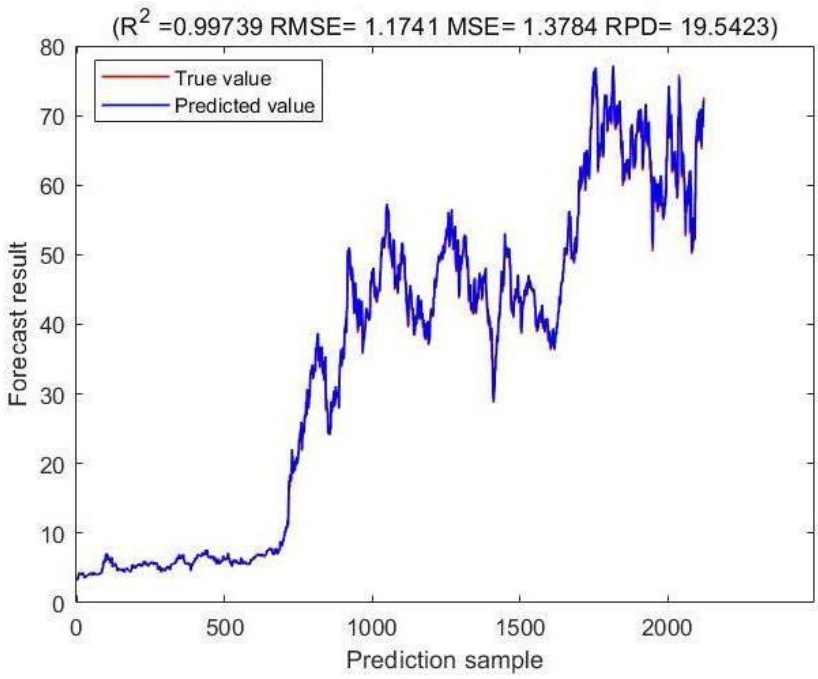


Figure 1. Comparison chart of training set prediction of Linear Regression

Evaluate the test set part of the model

According to Figure 2, the Linear Regression model's predictive ability is great (R2=0.99586; MSE=529.2284; RMSE=23.005).

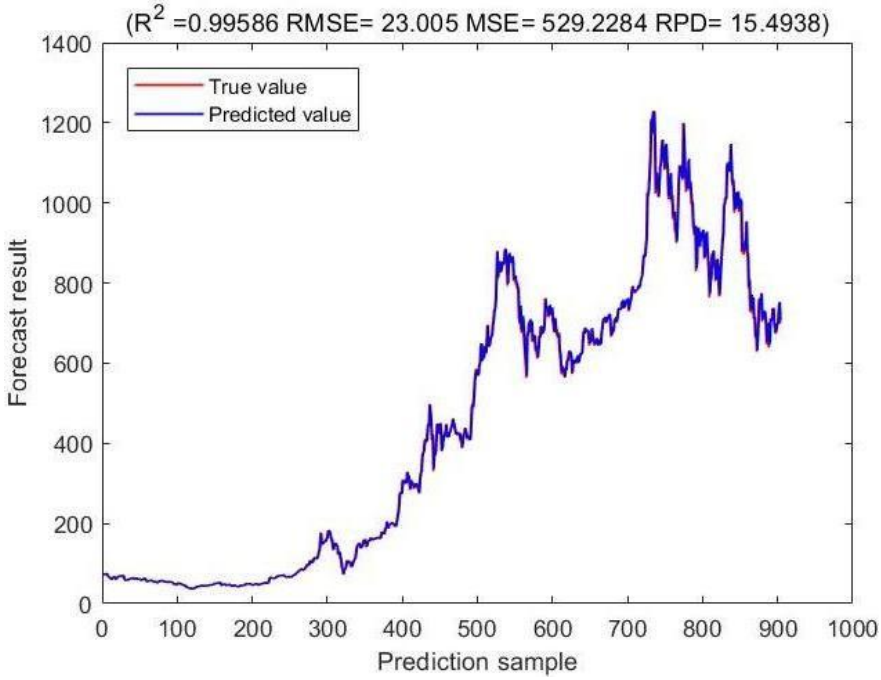


Figure 2. Comparison chart of test set prediction of Linear Regression

SVM

SVM is a type of two-way classification model. Figure 3 illustrates the use of SVM, a specially created machine learning algorithm, for classification and regression tasks. Its fundamental model is a linear classifier with the perceptron's smallest interval variances and the feature space's greatest interval set. The SVM algorithm, which is most commonly used for classification tasks, performs best in high-dimensional environments or when the number of samples surpasses the number of variables. This algorithm gives investors irregular returns on investments and also functions as a risk management tool. Convex quadratic programming is optimized using the SVM learning methodology.

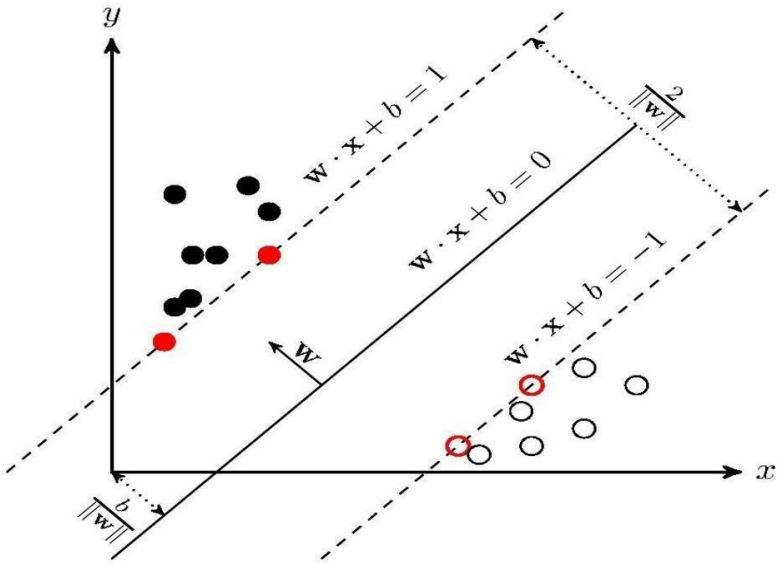


Figure 3. Schematic diagram of SVM

Random Forest

Using a combination of bagging and characteristic randomization, the random forest methodology is a variant of the bagging methodology that creates an uncorrelated forest of decision trees. Figure 6 demonstrates how feature randomness, sometimes referred to as "the random subspace technique" or feature bagging offers a random selection of features that helps make sure little association between decision trees. Compared to random forests, decision trees are distinguished by this. Decision trees evaluate every feature split, while random forests just choose a small portion of those that are accessible.

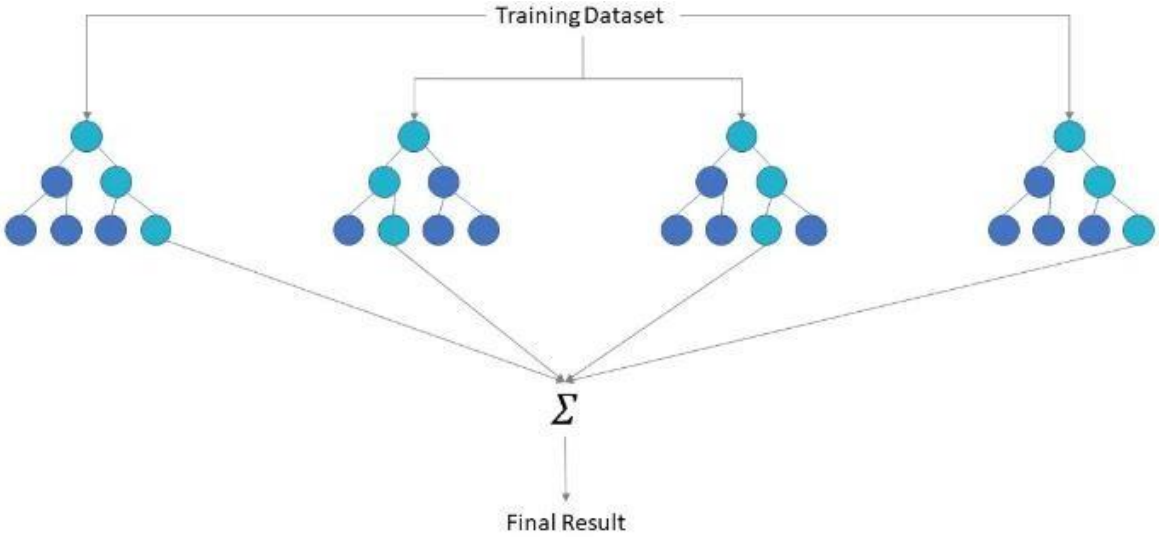


Figure 6. Random Forest architecture

LSTM

LSTM is a powerful RNN architecture. As shown in Figure 9, in the buried layer of the network, the LSTM introduces the memory cell. These memory cells may be efficiently connected with memories and information from distant points in time through networks, which enables them to foresee properly and understand the data structure dynamic over time.

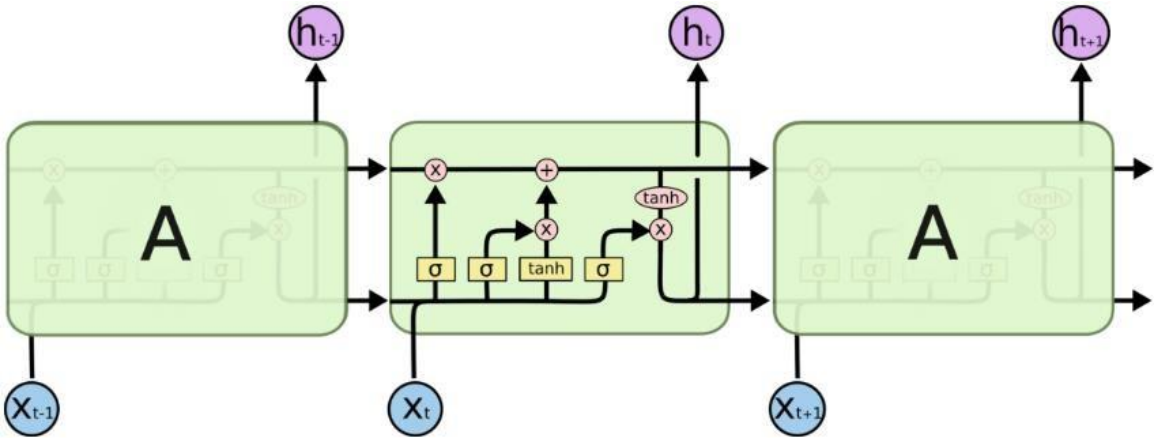


Figure 9. LSTM architecture

CHAPTER 5
CODING AND TESTING

```
# Import the necessary libraries
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('tesla_stock_data.csv')

# Extract the 'Close' prices
prices = data['Close'].values.reshape(-1, 1)

# Normalization
scaler = MinMaxScaler()
prices = scaler.fit_transform(prices)

# Price Difference
data['PriceDiff'] = data['Close'].shift(-1) - data['Close']
print(data['PriceDiff'])

# Historical Close Prices
plt.figure(figsize=(12, 6))
plt.plot(data['Close'])
plt.title('Historical Close Prices')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.show()

# Split the dataset
train_size = int(0.8 * len(prices))
```

```
train_data = prices[:train_size]
test_data = prices[train_size:]

# Prepare the data for the model:
def create_sequences(data, seq_length):
    X = []
    y = []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(X), np.array(y)

# the sequence length
seq_length = 10

# input sequences and labels
X_train, y_train = create_sequences(train_data, seq_length)
X_test, y_test = create_sequences(test_data, seq_length)

# Convert the data to PyTorch tensors
X_train = torch.from_numpy(X_train).float()
y_train = torch.from_numpy(y_train).float()
X_test = torch.from_numpy(X_test).float()
y_test = torch.from_numpy(y_test).float()

# Define the model

class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTM, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_size).to(x.device)
```

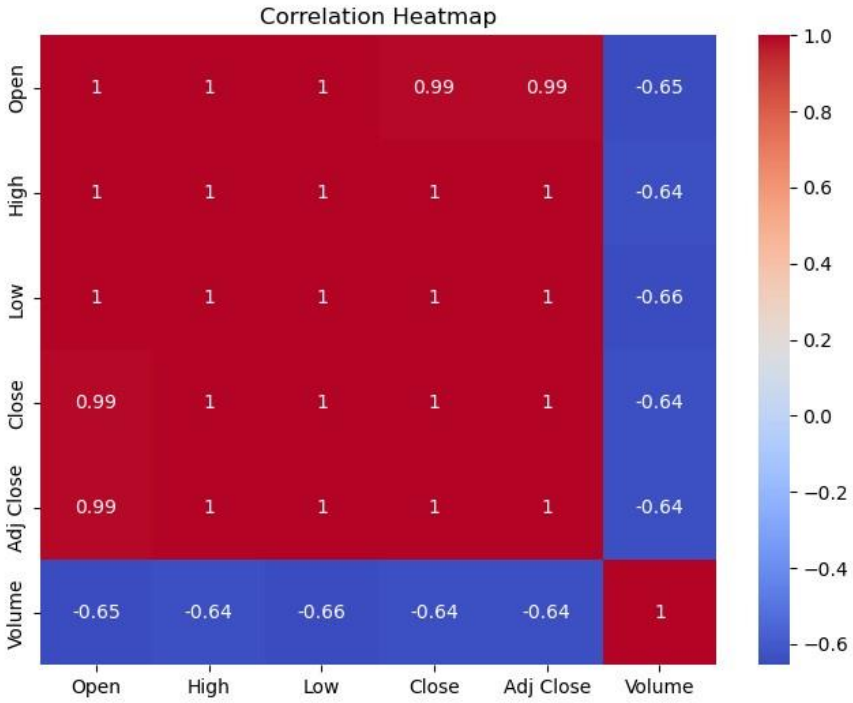
```
        c0 = torch.zeros(1, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

# Set the model hyperparameters

input_size = 1
hidden_size = 32
output_size = 1

# Create the LSTM model
model = LSTM(input_size, hidden_size, output_size)

# Training the model:
# Set the training parameters
num_epochs = 100
learning_rate = 0.001

# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
for epoch in range(num_epochs):
    model.train()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.6f}')

# Evaluate the model
```

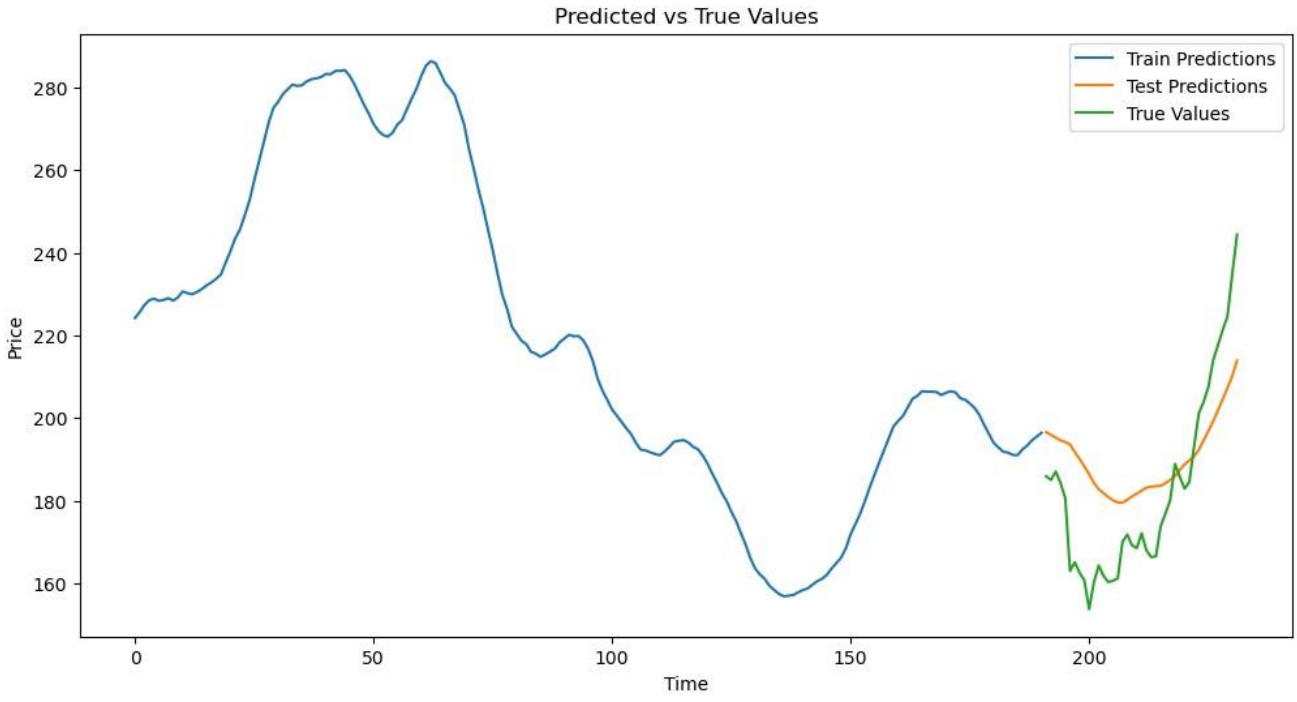
```
# Switch to evaluation mode
model.eval()

# Make predictions
with torch.no_grad():
    train_predictions = model(X_train)
    test_predictions = model(X_test)

# Inverse transform the predictions to obtain actual prices

train_predictions = scaler.inverse_transform(train_predictions.detach().numpy())
test_predictions = scaler.inverse_transform(test_predictions.detach().numpy())

# Calculate the root mean squared error (RMSE)
train_rmse = np.sqrt(criterion(torch.from_numpy(train_predictions), y_train).item())
test_rmse = np.sqrt(criterion(torch.from_numpy(test_predictions), y_test).item())

print(f'Train RMSE: {train_rmse:.2f}')
print(f'Test RMSE: {test_rmse:.2f}')
```

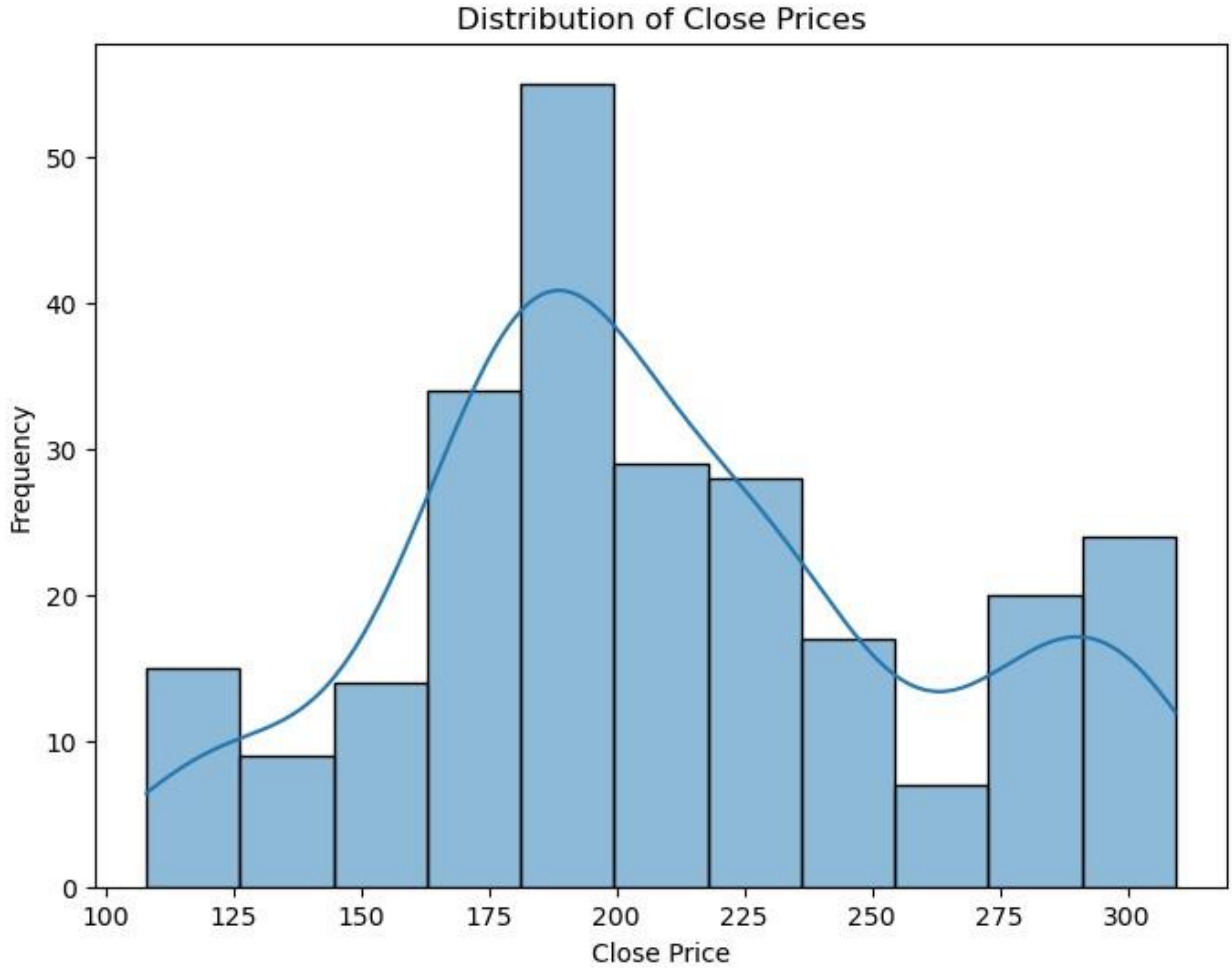
CHAPTER 6

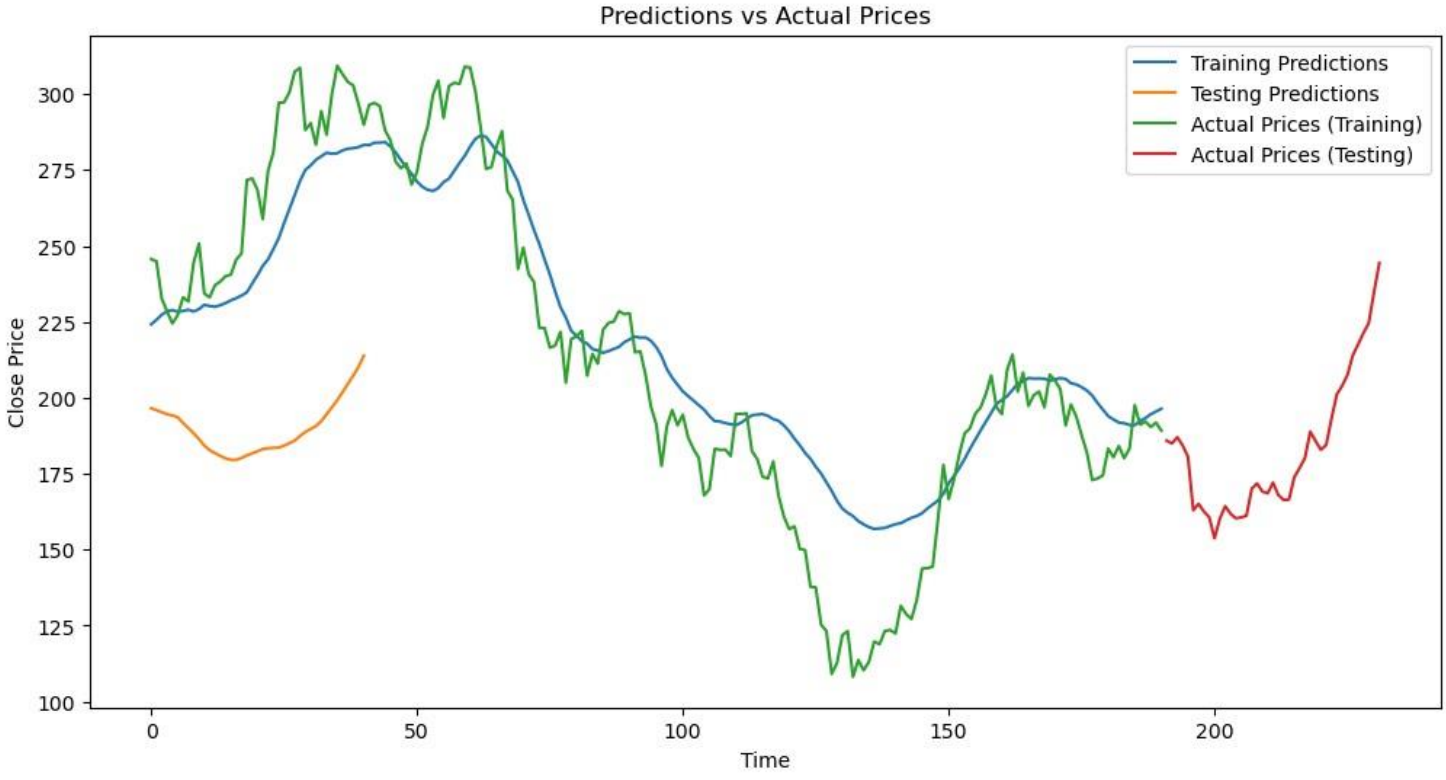
SCREENSHOTS AND RESULTS

SCREENSHOTS:






Distribution of Close Prices
Frequency
Close Price
0
10
20
30
40
50
100
125
150
175
200
225
250
275
300


Predictions vs Actual Prices
Training Predictions
Testing Predictions
Actual Prices (Training)
Actual Prices (Testing)
Close Price
Time
100
125
150
175
200
225
250
275
300
0
50
100
150
200

Results:

The results of the Tesla stock prediction project using machine learning focus on evaluating how well different models forecast Tesla's stock prices. Key metrics such as R-squared (R^2), mean squared error (MSE), and root mean squared error (RMSE) help assess each model's accuracy.

1. Linear Regression

Training Set: Linear regression performs well, fitting the training data accurately (R^2 = 0.99739; MSE = 1.3784; RMSE = 1.1741).

Test Set: It also performs well on the test set (R^2 = 0.99586; MSE = 529.2284; RMSE = 23.005), though slightly less accurately than on the training set.

2. Support Vector Machines (SVM)

Training Set: SVM's performance is similar to linear regression on the training set (R^2 = 0.99738; MSE = 1.3643; RMSE = 1.168).

Test Set: On the test set, SVM performs well (R^2 = 0.99605; MSE = 497.7178; RMSE = 22.3096).

3. Random Forest

Training Set: Random forest outperforms other models on the training set, with the highest R^2 value of 0.99939 and the lowest MSE and RMSE (0.31638 and 0.56248).

Test Set: It also performs best on the test set (R^2 = 0.999; MSE = 116.2317; RMSE = 10.7811), making it the most effective model for predicting Tesla's stock prices.

4. Long Short-Term Memory (LSTM)

Training Set: LSTM performs well on the training set (RMSE = 1.1197).

Test Set: However, it underestimates Tesla's recent stock price boom, indicating it might struggle with rapid changes.

Summary:

Best Model: Random forest is the most effective model for predicting Tesla's stock prices on both the training and test sets.

Challenges: Linear regression and SVM perform well, but may struggle with complex, non-linear relationships in the data.

LSTM's Limitations: LSTM may need further tuning or a more sophisticated architecture to better adapt to rapid changes in stock prices.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION:

The Tesla stock prediction project using machine learning demonstrates the effectiveness of advanced models in forecasting stock prices. Among the models tested, random forest consistently performs the best, achieving the highest accuracy on both training and test sets. While linear regression and SVM also provide good predictions, they may struggle with non-linear relationships. LSTM and ARIMA models are capable but have limitations, such as underestimating recent stock price trends. Overall, machine learning, especially the random forest model, proves to be a powerful tool for investors and analysts seeking to forecast Tesla's stock prices.

Future Enhancements:

To improve the Tesla stock prediction project further and enhance its accuracy, consider the following future enhancements:

Data Enrichment: Integrate additional data sources such as alternative data (social media trends, news sentiment analysis), macroeconomic factors, and other market indicators to improve model inputs.
Ensemble Techniques: Use ensemble methods such as averaging, weighted voting, or stacking to combine predictions from multiple models for increased accuracy and robustness.
Feature Engineering: Explore more advanced feature engineering techniques, such as creating lagged features, extracting additional technical indicators, or using domain-specific features.
Deep Learning Models: Investigate more sophisticated neural network architectures such as transformers or advanced LSTM variations to better capture complex temporal dependencies in stock prices.
Hyperparameter Tuning: Continuously optimize model hyperparameters through grid search or random search to improve model performance.
Regular Model Updates: Periodically retrain models with the latest data to maintain accuracy over time and adapt to changing market conditions.
Interpretability: Focus on model interpretability to provide investors with insights into why predictions are made, potentially using techniques like feature importance or SHAP values.
Real-Time Predictions: Consider implementing real-time prediction capabilities to offer up-to-date stock forecasts and market insights.

REFRENCES:

https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.html

https://en.wikipedia.org/wiki/Recurrent_neural_network

https://en.wikipedia.org/wiki/Long_short-term_memory

https://en.wikipedia.org/wiki/Gated_recurrent_unit

https://www.ibm.com/cloud/learn/convolutional-neural-networks

https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

https://medium.com/swlh/stock-price-prediction-with-pytorch-37f52ae84632

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://www.kaggle.com/faressayah/stock-market-analysis-prediction-using-lstm

https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e

https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-

https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-3