LSTM-Based Stock Price Forecasting

Group 13 – SYS 5185

Aahil Jivani (8729441)

Alex Rodriguez (300452235)

***Abstract*— This project develops a TensorFlow-based LSTM model to forecast Tesla's daily closing price using technical indicators (RSI, MACD, SMAs, EMAs) and VWAP. The unidirectional LSTM, with a dense output layer and mean squared error loss, predicts tomorrow's (t+1) closing price. An 80% training and 20% test split yields an R^2 near 0.93 and a mean absolute percentage error (MAE) under 7%. Hyperparameter tuning via KerasTuner optimizes layer sizes, dropout rates, and learning rates. These results confirm that an LSTM architecture for improving stock price prediction is achievable and provides a scalable framework for other financial time series.

Keywords—LSTM, Tesla, Stock Forecasting, TensorFlow

I. Introduction

Time series forecasting in financial markets exhibit non-linear, volatile, and stochastic traits. Traditional machine learning methods, such as linear regression, logistic regression, and technical analysis, are not able to capture the non-linearity present in financial data. Random Forest and Support Vector Regression (SVR) show improvements over the classical statistical models, although fall short in modeling sequential and time-dependent relationships. [1] Long Short-Term Memory (LSTM) networks are specifically designed to retain long-term dependencies and learn from historical sequences, making them a suitable choice for financial time series prediction. Prior research, such as Fischer and Krauss [2], has demonstrated that LSTM models can outperform traditional machine learning methods in forecasting stock movements by better capturing temporal patterns. This project applies an LSTM-based model to predict short-term price movements of Tesla stock using a breadth of common technical indicators derived from historical market data. To enhance feature relevance and reduce redundancy, input features are selected based on feature importance analysis using a Random Forest model. The objective of this paper is to extend existing research on LSTMs through integrating them with statistically selected technical features that adapt to the characteristics of the stock market.

II. Background & Literature Review

Machine learning for stock price forecasting has been well researched, yet there are many methods that have yet to be explored to tackle the main challenges surrounding market volatility, nonlinearity, and shifting data distributions. Traditional Machine Learning (ML) methods as explored by Pashankar and Shendage [1] compared linear regression, random forest, and support vector regression, demonstrating that non-linear methods can capture richer patterns in financial data but may not perform as well with high-volatility equities. The findings suggest that architectures designed for sequential data could offer stronger predictive performance within the domain of predictive time series.

Fischer and Krauss [2] then showed that Long Short-Term Memory (LSTM) networks outperform memory-free algorithms for daily market prediction, due to the gating mechanism's ability to retain relevant historical context. Zhu and Laptev [3] extended deep forecasting frameworks to noisy, large-scale time series at Uber with forecasting uber rideshare demand. The model uses LSTM architecture enhanced with Bayesian dropout to estimate predictive confidence, enabling real-time anomaly detection in a noisy, fast-changing operational environment. Similarly, Balamohan et al. [4] addressed evolving datasets, indicating that improved gating mechanisms and data preprocessing can enhance LSTM's stability and accuracy in rapidly changing conditions.

Other refinements to deep models include convolutional–recurrent hybrids and attention layers, mentioned in Chalapathy and Chawla's survey [5]. However, Patel et al. [6] emphasized that increased architectural depth also demands careful hyperparameter tuning. Gülmez [7] had deployed an Artificial Rabbits Optimization (ARO) to optimize LSTM hyperparameters, achieving marked accuracy gains over default or manual searches.

Despite these noteworthy advances, there remains a gap in highly targeted, next-day predictions of volatile single stocks, such as Tesla. Existing literature demonstrates its focus on broader indices or aggregates multiple equities with performance measures using the Sharpe ratio, or rate of return, which are standard financial metric to assess portfolio performance. The use of systematic feature selection (e.g., using a Random Forest for technical indicator importance) is rarely performed, potentially leaving models prone to overfitting had they considered including more technical indicators as inputs. Consequently, a specialized pipeline that combines hyperparameter tuning, refined feature sets, and an LSTM fitted to Tesla's short-horizon forecasts promises both academic and practical value within the context of stock prediction.

III. Methodology

The objective of this study is to develop an LSTM-based framework that accurately forecasts Tesla's next-day closing price. To achieve this, there must be a statistically significant feature selection process, followed by thorough hyperparameter tuning. The process begins by gathering Tesla's daily stock data through an API with Yahoo Finance and add several technical indicators, including RSI, MACD, SMAs, EMAs, and VWAP. Since not all indicators carry equal predictive value, sklearn's Random Forest importance analysis is employed to isolate the subset of features that correlates with Tesla's future closing prices. This ensures the LSTM focuses on relevant signals for short-horizon prediction rather than reacting to noise in the data.

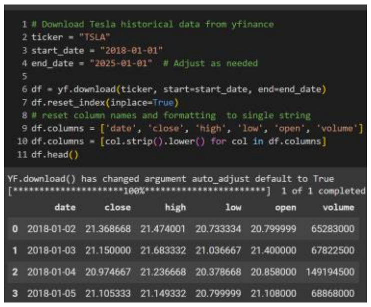
Once the feature set is finalized, the cleaned data is sequentially divided into training (80%) and testing (20%) subsets, this is a standard practice among train and test splits. The training partition is fed into a unidirectional LSTM architecture that processes a rolling window of recent days to predict the next day's closing price. Hyperparameters such as the number of LSTM units, dropout fractions, and optimizer settings are tuned with KerasTuner, using mean squared error to guide the adjustments. Early stopping will further prevent overfitting by halting training once validation loss reaches minimum threshold. The finalized model is evaluated on the hold-out test set using mean absolute percentage error (MAPE), root mean squared error (RMSE), and R² metrics to evaluate the performance of the model's ability to predict the observed data accurately.

IV. Implementation

The proposed model was implemented using a standard machine learning process which involved data extraction, feature engineering, data cleaning, feature evaluation, exploratory data analysis (EDA), and model development. In the following sections, these implementation steps will be discussed in further detail.

A. Data Extraction

Tesla stock data was acquired from yahoo finance using the yfinance library. This data scraping step allowed the following stock data to be extracted for Tesla from 2018 to 2024: 'date', 'close', 'high', 'low', 'open', and 'volume'. Where 'date; is the trading day, 'close', 'high', 'low', and 'open' are the tesla stock prices, and 'volume' is the number of shares trades during that day. The code for that data extraction is shown below.

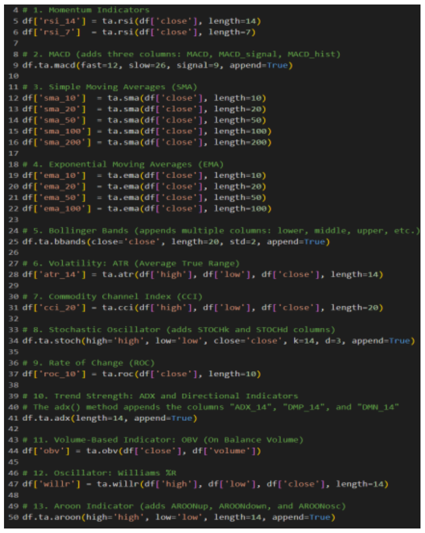


```
# Download Tesla historical data from yfinance
ticker = "TSLA"
start_date = "2018-01-01"
end_date = "2025-01-01"  # Adjust as needed

df = yf.download(ticker, start=start_date, end=end_date)
df.reset_index(inplace=True)
# reset column names and formatting  to single string
df.columns = ['date', 'close', 'high', 'low', 'open', 'volume']
df.columns = [col.strip().lower() for col in df.columns]
df.head()
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  1 of 1 completed
         date      close       high        low       open     volume
0  2018-01-02  21.368668  21.474001  20.733334  20.799999   65283000
1  2018-01-03  21.150000  21.683332  21.036667  21.400000   67822500
2  2018-01-04  20.974667  21.236668  20.378668  20.858000  149194500
3  2018-01-05  21.105333  21.149332  20.799999  21.108000   68868000
```

Figure 1: yfinance Data Extraction for Tesla Stcok

To make the dataset more robust and informative, technical indicators were imported using the pandas_ta library. The code for this extraction can be seen below.



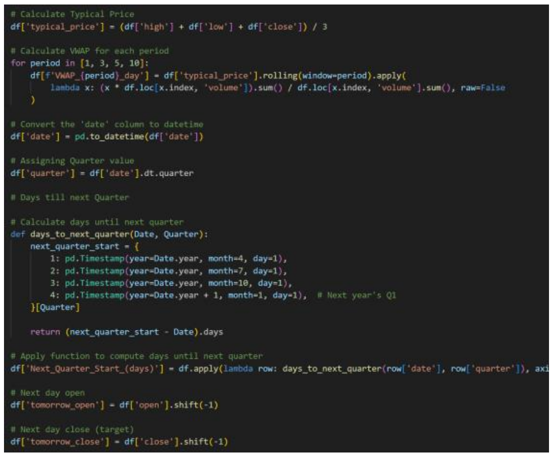
```
# 1. Momentum Indicators
df['rsi_14'] = ta.rsi(df['close'], length=14)
df['rsi_7']  = ta.rsi(df['close'], length=7)

# 2. MACD (adds three columns: MACD, MACD_signal, MACD_hist)
df.ta.macd(fast=12, slow=26, signal=9, append=True)

# 3. Simple Moving Averages (SMA)
df['sma_10']  = ta.sma(df['close'], length=10)
df['sma_20']  = ta.sma(df['close'], length=20)
df['sma_50']  = ta.sma(df['close'], length=50)
df['sma_100'] = ta.sma(df['close'], length=100)
df['sma_200'] = ta.sma(df['close'], length=200)

# 4. Exponential Moving Averages (EMA)
df['ema_10']  = ta.ema(df['close'], length=10)
df['ema_20']  = ta.ema(df['close'], length=20)
df['ema_50']  = ta.ema(df['close'], length=50)
df['ema_100'] = ta.ema(df['close'], length=100)

# 5. Bollinger Bands (appends multiple columns: lower, middle, upper, etc.)
df.ta.bbands(close='close', length=20, std=2, append=True)

# 6. Volatility: ATR (Average True Range)
df['atr_14'] = ta.atr(df['high'], df['low'], df['close'], length=14)

# 7. Commodity Channel Index (CCI)
df['cci_20'] = ta.cci(df['high'], df['low'], df['close'], length=20)

# 8. Stochastic Oscillator (adds STOCHk and STOCHd columns)
df.ta.stoch(high='high', low='low', close='close', k=14, d=3, append=True)

# 9. Rate of Change (ROC)
df['roc_10'] = ta.roc(df['close'], length=10)

# 10. Trend Strength: ADX and Directional Indicators
# The adx() method appends the columns "ADX_14", "DMP_14", and "DMN_14"
df.ta.adx(length=14, append=True)

# 11. Volume-Based Indicator: OBV (On Balance Volume)
df['obv'] = ta.obv(df['close'], df['volume'])

# 12. Oscillator: Williams %R
df['willr'] = ta.willr(df['high'], df['low'], df['close'], length=14)

# 13. Aroon Indicator (adds AROONup, AROONdown, and AROONosc)
df.ta.aroon(high='high', low='low', length=14, append=True)
```

Figure 2: Technical Indicators Import using pandas_ta

With these two data extractions, the raw dataset for tesla stock contains 38 features. From this data, additional features will be created and discussed in the next section.

B. Feature Engineering

Additional features were created based off the existing data to help the model make correlations and not solely rely on the technical indicators for patterns. The following features were created and added to the dataset: Typical Price, Volume-Weighted Average Price (VWAP), Quarter, Days till next Quarter, tomorrow open, and tomorrow close. The code for these feature additions can be seen below.



```
# Calculate Typical Price
df['typical_price'] = (df['high'] + df['low'] + df['close']) / 3

# Calculate VWAP for each period
for period in [1, 3, 5, 10]:
    df[f'VWAP_{period}_day'] = df['typical_price'].rolling(window=period).apply(
        lambda x: (x * df.loc[x.index, 'volume']).sum() / df.loc[x.index, 'volume'].sum(), raw=False
    )

# Convert the 'date' column to datetime
df['date'] = pd.to_datetime(df['date'])

# Assigning Quarter value
df['quarter'] = df['date'].dt.quarter

# Days till next Quarter

# Calculate days until next quarter
def days_to_next_quarter(Date, Quarter):
    next_quarter_start = {
        1: pd.Timestamp(year=Date.year, month=4, day=1),
        2: pd.Timestamp(year=Date.year, month=7, day=1),
        3: pd.Timestamp(year=Date.year, month=10, day=1),
        4: pd.Timestamp(year=Date.year + 1, month=1, day=1),  # Next year's Q1
    }[Quarter]

    return (next_quarter_start - Date).days

# Apply function to compute days until next quarter
df['Next_Quarter_Start_(days)'] = df.apply(lambda row: days_to_next_quarter(row['date'], row['quarter']), axi

# Next day open
df['tomorrow_open'] = df['open'].shift(-1)

# Next day close (target)
df['tomorrow_close'] = df['close'].shift(-1)
```

Figure 3: Feature Creation

Typical Price is an average of the high, low, and close price for the day. VWAP uses the Typical Price, volume and a set number of days to calculate a weighted average price based off the volume summation.

'Quarter' and 'Days till next Quarter' were created by categorizing the data based on the date and the date range of each Quarter. This allows any patterns related to stock price increases/decreases due to certain Quarters or nearing the next Quarter to be captured.

Tomorrow’s ‘open’ and ‘close’ were added to the dataset since the theoretical application of this model would result in the user running the model during the day to try and predict the next days close price. Therefore, the user would have access to the open price for that day. That means the user has access to all previous data and the current day’s open price. To translate that to the dataset, tomorrow_open was added to simulate this behavior. Tomorrow_close will be the target feature and was added so that the model predicted a day in advance instead of the “end of day” close price.

C. *Data Cleaning*

Next, since the proposed model is an LSTM, null values cannot be left in the dataset. A lot of the technical indicators imported use historical price data to make their calculations. Therefore, the beginning of the dataset contains a lot of null values since the technical indicators don’t have access to price data before 2018 but may need that data to do their calculations. Instead of making the null values zero, it was decided to remove the rows with null values. This allows the LSTM model to not become biased due to zero valued features.

The resulting deletion of rows removed about a year’s worth of data at the beginning of the dataset. Now the dataset is approximately from the beginning of 2019 to the end of 2024.

D. *Feature Importance & Selection*

The dataset contained 47 features in total, which can generate excess noise or add redundancy. The number of features was trimmed based off a Random Forest (RF) feature importance algorithm for the model to make predictions accurately. The following key features were excluded from examination: 'date', 'open', 'high', 'low', 'close', 'volume', 'tomorrow_open', 'tomorrow_close', 'typical_price', 'quarter', 'Next_Quarter_Start_(days)'. The 36 features were evaluated by their R^2 scores with threshold value of 0.001. 15 features achieved an R^2 value greater than threshold: 'VWAP_1_day', 'VWAP_3_day', 'VWAP_5_day', 'sma_200', 'ema_50', 'sma_50', 'ema_100', 'BBU_20_2.0', 'sma_20', 'obv', 'ema_10', 'sma_100', 'sma_10', 'BBM_20_2.0', and ‘VWAP_10_day'.

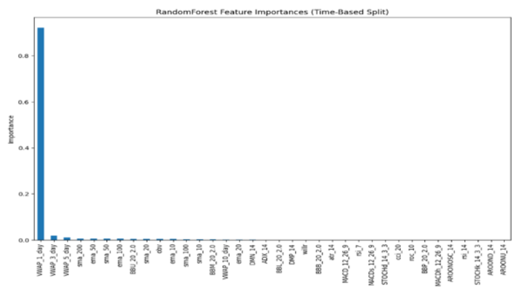
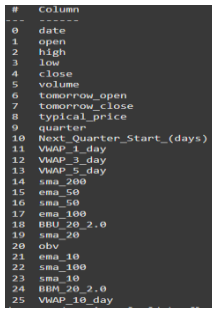


Figure 4: Feature Importance via Random Forest R^2 Score

The score for ‘VWAP_1_day’ surpassed the other features. Since VWAP is a volume weighted average of the High, Low, and Close price, within the timeframe of 1 day, this implies that VWAP_1_day is equivalent to the Typical Price. Unless the stock price is highly volatile from day to day, the Typical Price should be very close to tomorrow’s Close price. Thus, the Random Forest model must have concluded that the Typical Price was always very close to tomorrow’s Close price indicating a high R^2 score.

Excluding the low importance features resulted in a dataset of 26 features and 1561 rows. Those features were:

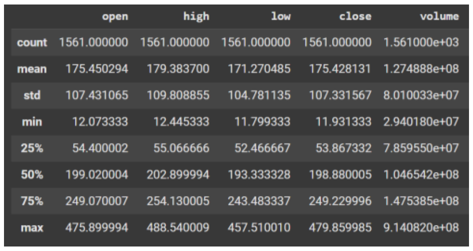


```
 #   Column
---  ------
 0   date
 1   open
 2   high
 3   low
 4   close
 5   volume
 6   tomorrow_open
 7   tomorrow_close
 8   typical_price
 9   quarter
 10  Next_Quarter_Start_(days)
 11  VWAP_1_day
 12  VWAP_3_day
 13  VWAP_5_day
 14  sma_200
 15  ema_50
 16  sma_50
 17  ema_100
 18  BBU_20_2.0
 19  sma_20
 20  obv
 21  ema_10
 22  sma_100
 23  sma_10
 24  BBM_20_2.0
 25  VWAP_10_day
```

Figure 5: Final Dataset Features

E. *Exploratory Data Analysis*

To further evaluate the dataset, exploratory data analysis was conducted to help determine possible trends and any clues as to how to optimally design the LSTM model. Firstly, summary statistics were conducted on the data set for ‘open’, ‘high’, ‘low’, ‘close’, and ‘volume’.



	open	high	low	close	volume
count	1561.000000	1561.000000	1561.000000	1561.000000	1.561000e+03
mean	175.450294	179.383700	171.270485	175.428131	1.274888e+08
std	107.431065	109.808855	104.781135	107.331567	8.010033e+07
min	12.073333	12.445333	11.799333	11.931333	2.940180e+07
25%	54.400002	55.066666	52.466667	53.867332	7.859550e+07
50%	199.020004	202.899994	193.333328	198.880005	1.046542e+08
75%	249.070007	254.130005	243.483337	249.229996	1.475385e+08
max	475.899994	488.540009	457.510010	479.859985	9.140820e+08

Figure 6: Summary Statistics

As seen above, the summary statistics don’t provide that much useful information due to Tesla stock having an aggressive increasing trend in its price. The only useful data is under Volume where we can see that the daily volume is consistent around the average but there are occasional high-volume days. There may be some trends around these outliers, which will be further explored.

Next, histograms of the features were generated to see the distribution of each feature. The results can be seen below:

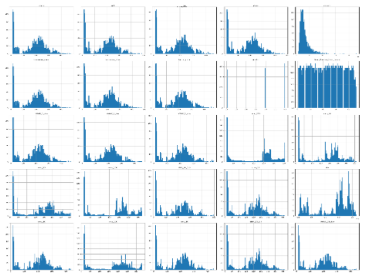


Figure 7: Histograms of the Features

Many of the features exhibited a normal distribution with a slight concentration near the low range. 'ema_50' exhibited a uniform distribution, which may suggest that it will not be that helpful in predicting the Close price. Other than 'ema_50' the rest of the features were not uniform so it was determined to leave all the features in the dataset as there were no features that would create a bias in the model.

To further explore the distribution of the features, box plots were generated. Evidently, volume was the only feature that was highly concentrated but contained some high valued outliers.

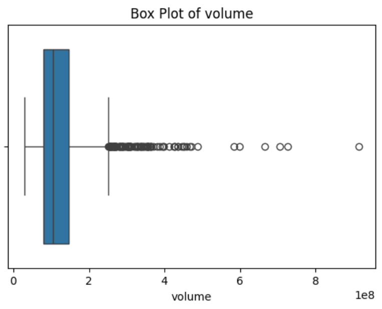


Figure 8: Box Plot of Volume

As seen above, these outliers may suggest drastic changes in the stock price which the model may be able to use to predict the change in the Close price. From appearance, most of the features have a large range. This is most likely due to Tesla stocks' increasing trend. To enhance model accuracy, the features will be normalized to counteract such a large range.

Next, Tesla's stock price was visualized alongside 'sma_50' and 'ema_50' to benchmark the technical indicators with the stock prices' momentum and trends.

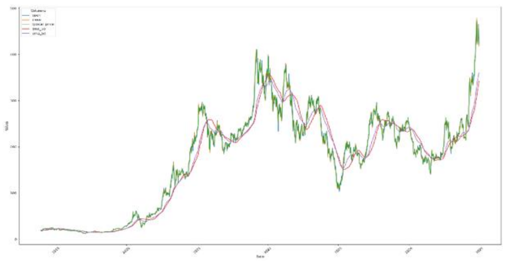


Figure 9: Tesla Stock Price compared to Sample Technical Indicators

The technical indicators shown above tend to fall under the price during increasing trends and fall over the price during decreasing trends. Although, the overall momentum and trend is captured in the technical indicators. This may suggest that the model may tend to over predict, or under predict, when it trains on the technical indicator data at first. To prevent this and make the model more accurate at predicting the close price, many [5] epochs will be used during training so that the model can adjust each round to lessen the over/under predictions potentially learned from the technical indicator data.

Lastly, correlation between features and volatility of the Close price were explored. For the purposes of this paper, only a heatmap and a graph of the rolling volatility will be shown below and discussed.

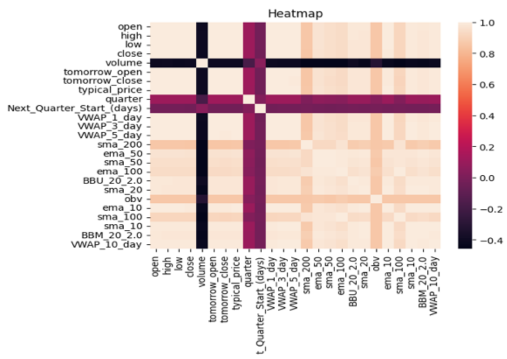


Figure 10: Heatmap of Features

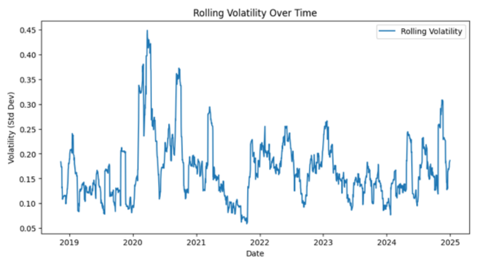


Figure 11: Rolling Volatility of Close Price

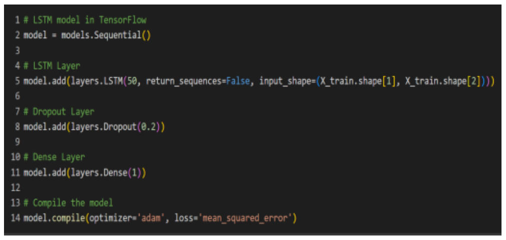
As seen in Figure 10, there is a high correlation between features. This may appear since most of the technical indicators use the stock price in their calculations. Surprisingly, volume is not highly correlated with any other features although VWAP uses volume in its calculation. To prevent redundancy and multicollinearity, a small rolling window was used due to most of the technical indicators short-term trends demonstrating difference from the other features. This allows for lower correlation between features and potentially provides more useful information to the model, instead of using a large rolling window where most feature characteristics remain similar in the long run.

The rolling volatility graph in Figure 11 shows that volatility is not constant. This benefits the LSTM model since volatility is captured in a lot of the momentum technical indicators and it can learn potential trends from these fluctuations. Overall, the EDA showed some insight into feature trends and characteristics, which helped in determining the model's parameters such as number of epochs and rolling window size.

F. LSTM Model

To prepare the data for training and testing, the data was normalized to ensure stable training while assisting the model with patterns rather than scale differences across features. Then the dataset was split sequentially using an 80/20 train-test ratio The dataset was split chronologically to preserve the temporal structure and ensure that the model learns from past data to predict future values realistically. As mentioned in the previous section, a small rolling window would have to be used. It was decided to make the rolling window of size 20. This translated to about a month’s worth of data since only weekdays are captured in stock data. This rolling window was large enough to show weekly trends but small enough to prevent averaging out or causing the technical indicators and features to show similar trends across the entire dataset.

The model’s architecture was simple to prevent complexity and overfitting while created using TensorFlow. Additionally, a 20% dropout layer was added after the LSTM layer to prevent overfitting. The model’s architecture can be seen below.



```
# LSTM model in TensorFlow
model = models.Sequential()

# LSTM Layer
model.add(layers.LSTM(50, return_sequences=False, input_shape=(X_train.shape[1], X_train.shape[2])))

# Dropout Layer
model.add(layers.Dropout(0.2))

# Dense Layer
model.add(layers.Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

Figure 12: LSTM Model Architecture

V. Results

In this section, the results of the LSTM model will be explored to show how well the model can predict the Close price of Tesla stock a day in advance.

A. *Training Results*

The model was trained using 50 epochs and a batch size of 32. A validation set was created from 10% of the training set to compare the loss. The train and validation loss were graphed out to evaluate the outcome.

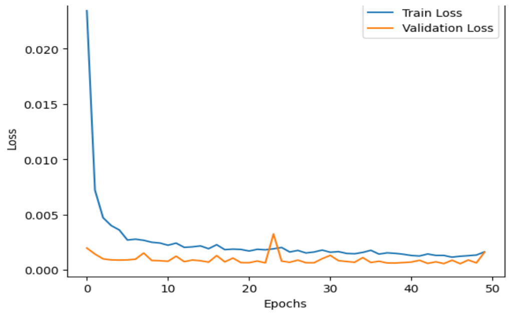


Figure 13: Model Training Results

As seen above, the model was able to fit to the training data well after 5 epochs or so. After 50 epochs, the training loss approached the validation loss, resulting in a fairly accurate model based off of the training data. More epochs could have been used to decrease the loss but it was determined that that may cause overfitting. Since Tesla’s stock price is a time series data type, it was best to try and minimize overfitting as much as possible due to its high volatility. Also, since stock data can drastically change due to external events, such as new reports or publicity, overfitting would just worsen the model since external events/news data is not incorporated in the dataset.

B. *Test Results*

The LSTM model was tested on the last 20% of the data set, which was from about the end of 2023 to the end of 2024. Several evaluation metrics were explored, including graphing the predicted Close price and actual Close price, mean error, RMSE, and R^2. A summary of the results can be found below.

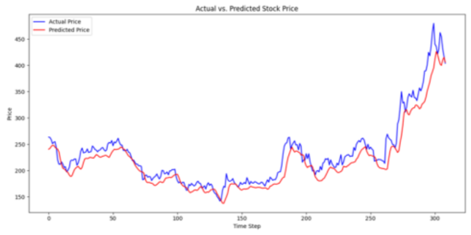
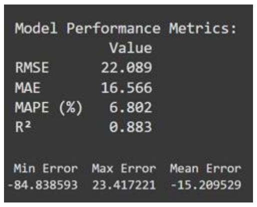


Figure 14: Predicted Close Price vs Actual



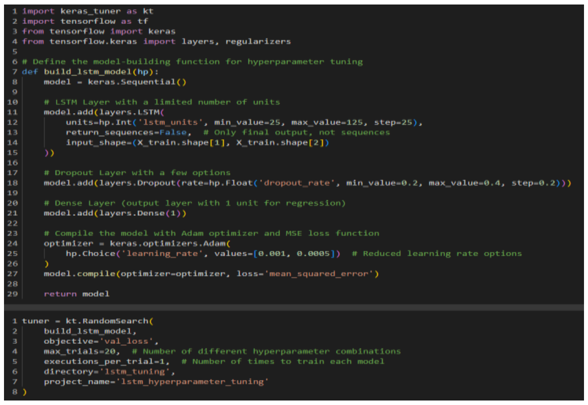
```
Model Performance Metrics:
            Value
RMSE       22.089
MAE        16.566
MAPE (%)    6.802
R²          0.883

 Min Error  Max Error  Mean Error
-84.838593  23.417221  -15.209529
```

Figure 15: LSTM Model Results

The model tended to underpredict the actual Close price and had an RMSE of 22.089 and an R^2 of 0.883. Its mean error was roughly -\$15. The model performed well but in terms of using this for a trading algorithm, this model wouldn’t be that helpful at this stage since being off by \$15 per day is quite a large amount, especially if the difference between the Open and Close price is smaller than \$15. Therefore, the model would have to be improved, which was done via hyperparameter tuning.

C. *Hyperparameter Tuning*

Since the LSTM model was created using TensorFlow, keras-tuner was used for hyperparameter tuning. The following parameters were tuned: LSTM units, dropout rate, and learning rate. The architecture of the LSTM model was kept the same so that comparison between the initial model and the best model could be made effectively. A sample of the tuner can be seen below.

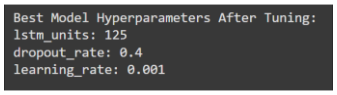


```
import keras_tuner as kt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers

# Define the model-building function for hyperparameter tuning
def build_lstm_model(hp):
    model = keras.Sequential()

    # LSTM Layer with a limited number of units
    model.add(layers.LSTM(
        units=hp.Int('lstm_units', min_value=25, max_value=125, step=25),
        return_sequences=False,  # Only final output, not sequences
        input_shape=(X_train.shape[1], X_train.shape[2])
    ))

    # Dropout Layer with a few options
    model.add(layers.Dropout(rate=hp.Float('dropout_rate', min_value=0.2, max_value=0.4, step=0.2)))

    # Dense Layer (output layer with 1 unit for regression)
    model.add(layers.Dense(1))

    # Compile the model with Adam optimizer and MSE loss function
    optimizer = keras.optimizers.Adam(
        hp.Choice('learning_rate', values=[0.001, 0.0005])  # Reduced learning rate options
    )
    model.compile(optimizer=optimizer, loss='mean_squared_error')

    return model
```

```
tuner = kt.RandomSearch(
    build_lstm_model,
    objective='val_loss',
    max_trials=20,  # Number of different hyperparameter combinations
    executions_per_trial=1,  # Number of times to train each model
    directory='lstm_tuning',
    project_name='lstm_hyperparameter_tuning'
)
```

Figure 16: Code sample of Hyperparameter Tuning

LSTM units of 25, 50, 75, 100, and 125; dropout rates of 0.2 and 0.4; and learning rates of 0.001 and 0.0005 were tested. An outcome of 20 different trials utilizing all possible combinations were found. The models were trained on the training set and tested on the validation set. The validation loss was used to choose the best model. After testing all 20 models, the model that had the lowest validation loss was:



```
Best Model Hyperparameters After Tuning:
lstm_units: 125
dropout_rate: 0.4
learning_rate: 0.001
```

Figure 17: Best LSTM Model Parameters

The hyperparameter tuned LSTM model uses more LSTM units than the initial model, 125 vs 50, uses a higher dropout rate, 0.4 vs 0.2, and uses the same learning rate of 0.001.

D. Final Model Results

The results of the final LSTM hyperparameter tuned model can be seen below:

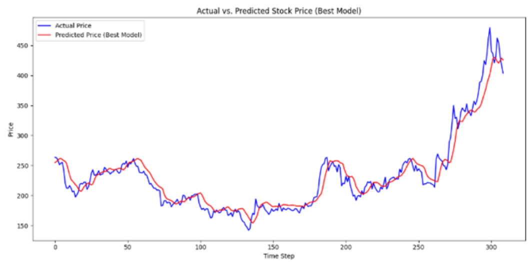
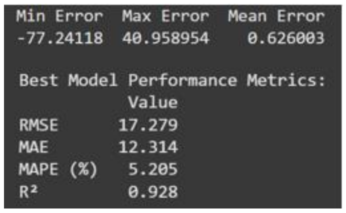


Figure 18: Final Model Predictions of Close Price vs Actual



```
Min Error  Max Error  Mean Error
-77.24118  40.958954    0.626003

 Best Model Performance Metrics:
               Value
RMSE          17.279
MAE           12.314
MAPE (%)       5.205
R²             0.928
```

Figure 19: Final Model Results

The final model performed significantly better than the initial model with a mean error of only \$0.63. The RSME was better with a value of 17.279 and the R^2 was higher with a value of 0.928. Overall, while only tuning 3 parameters and keeping the LSTM architecture simple, the final model was able to perform accurately and considerably better than the initial model. This final model could potentially be used in a trading algorithm since on average it was only off by a matter of cents.

VI. Conclusion

This project shows that a unilateral LSTM model with strategic feature selection and hyperparameter tuning can effectively model next-day stock price movements for Tesla, reducing mean error from about \$15 to \$0.63 while raising R^2 to around 0.93. By combining a Random Forest–based feature importance analysis with a hyperparameter search (across 25–125 LSTM units, dropout rates of 0.2–0.4, and learning rates of 0.0005–0.001), the final model shows significant improvement over the initial model. The model, however, is subject to limitations past January 2025 in determining an appropriate magnitude in direction of price trends due to recent company events. Future research regarding how best to incorporate additional data sources, such as real-time news or social media sentiment with Natural Language Processing (NLP) to capture text-based signals may enhance the accuracy of the model. Future research could integrate these NLP-based features within a deep reinforcement learning (DRL) framework, where an autonomous trading agent can make optimized buy/sell decisions in response to both technical indicators and sentiment data.

References

[1] S. Pashankar, J. Shendage and Pawar, *Journal of Advanced Zoology*, vol. 45, no. S-4, pp. 118-127, 2024.

[2] Fischer and Krauss, "Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions,," *European Journal of Operational Research*, pp. 1-16, 2017.

[3] N. Laptev and L. Zhu, "“Deep and Confident Prediction for Time Series at Uber,”," *IEEE International Conference on Data Mining Workshops (ICDMW)*, p. 103–110., 2017.

[4] S. Balamohan, V. Khanaa and K. Sivaraman, "Effective Stock Market Pricing Prediction Using Long Short Term Memory-Upgraded Model (LSTM-UP) on Evolving Data Sets," *SN Computer Science*, Vols. 4,, no. 658, 2023.

[5] R. Chalapathy and S. Chawla, "Deep Learning for Anomaly Detection: A Survey," *arXiv*, 2019.

[6] H. Patel, B. K. Bolla and S. E. D. Reddy, "Comparative Study of Predicting Stock Index Using Deep Learning Models," *Lecture Notes in Computer Science*, vol. 13754, p. 244–257, 2023.

[7] B. Gülmez, "Stock Price Prediction with Optimized Deep LSTM Network with Artificial Rabbits Optimization Algorithm," *Expert Systems with Applications*, vol. 227, 2023.