Predicting Tesla Stock Prices Using LSTM Neural Networks

<https://github.com/SinghPrabhjot19/Tesla/tree/main>

# Table of Contents

1. Executive Summary
2. Introduction
3. Methodology
   * Data Acquisition
   * Data Preprocessing
   * Model Architecture
4. Code Implementation
   * Reading the Dataset
   * Target Variable Identification
   * Visualizing the Distribution of the Target Variable
   * Basic Level Data Exploration
   * Identifying and Rejecting Useless Columns
   * Visual Exploratory Data Analysis (EDA)
   * Feature Selection Based on Data Distribution
   * Removal of Outliers and Missing Values
   * Visual and Statistical Correlation Analysis
   * Predicting Tesla Stock Prices Using LSTM Neural Networks
5. Results
6. Conclusion

# Executive Summary

The report traces the creation and application of a Long Short-Term Memory (LSTM) demonstrate outlined to estimate Tesla's stock costs based on authentic information. By leveraging this show, dealers and money related investigators can pick up profitable bits of knowledge into future cost patterns. The method includes a few key stages, counting information preprocessing to guarantee precision and consistency, model advancement to set up prescient capabilities, and preparing to optimize execution. Assessment of the model's adequacy is conducted to gage its exactness and unwavering quality in estimating Tesla's stock costs. The discoveries and suggestions of the comes about are talked about, highlighting the potential benefits and restrictions of utilizing LSTM models in stock cost forecast. Eventually, this report gives a comprehensive outline of the technique, comes about, and conclusions drawn from the execution of the LSTM demonstrate for determining Tesla's stock costs, offering valuable bits of knowledge for traders and monetary examiners looking for to form educated choices within the energetic world of stock showcase venture.

# Introduction

Tesla, Inc. stands as a trailblazer within the electric vehicle division, famous not as it were for its ground-breaking advancements but moreover for the momentous instability of its stock costs. Within the domain of money related markets, the capacity to anticipate these vacillations with exactness is fundamental, advertising speculators and dealers a competitive edge in exploring the complexities of stock showcase flow. Conventional strategies of time-series determining regularly battle to capture the intricate patterns characteristic in stock cost developments, especially within the case of Tesla, where components such as innovative headways, showcase assumption, and administrative shifts can apply noteworthy impact.

To address these challenges, this report digs into the application of Long Short-Term Memory (LSTM) systems, a sort of repetitive neural organize eminent for its capacity to hold data over amplified periods. By leveraging the inborn memory capabilities of LSTM models, we point to open a more profound understanding of the fundamental elements driving Tesla's stock costs, subsequently giving a strong system for estimating future patterns.

The basis behind utilizing LSTM systems lies in their capacity to capture long-term conditions inside successive information, making them well-suited for analyzing the nuanced designs predominant in stock showcase information. Not at all like routine time-series models, LSTM systems exceed expectations at recognizing and joining complex transient connections, empowering more exact forecasts indeed within the confront of unstable showcase conditions.

Through this investigation, we look for to assess the viability of LSTM systems as a apparatus for estimating Tesla's stock costs. By saddling the control of progressed machine learning strategies, we endeavor to prepare financial specialists and monetary investigators with a powerful instrument for making educated choices within the ever-evolving scene of stock showcase venture.

Within the ensuing sections of this report, we are going dig into the strategy utilized in preparing and assessing the LSTM show, analyze the comes about gotten, and draw conclusions with respect to its viability in anticipating Tesla's stock costs. Eventually, our point is to contribute to the body of information encompassing monetary examination and engage partners with significant experiences inferred from cutting-edge computational methods.

# Methodology

**Data Acquisition:** The dataset utilized in this consider includes day by day stock costs of Tesla, traversing from January 2019 to December 2021. These information were sourced from the Yahoo Fund site, giving comprehensive data on opening, closing, tall, and moo costs, as well as exchanging volume for each exchanging day inside the required time period.

**Data Preprocessing:** Earlier to show preparing, the dataset experienced preprocessing to improve the LSTM model's execution and joining. This preprocessing included two key steps:

1. **Normalization:** To facilitate optimal training, the stock price data were normalized using the MinMaxScaler technique. This process scaled the data to a range of 0 to 1, ensuring uniformity and aiding in the model's convergence during training.
2. **Training and Testing Split:** The dataset was apportioned into preparing and testing sets utilizing an 80/20 proportion. This apportioning guaranteed that the show was prepared on a adequate sum of information whereas holding a isolated subset for assessment purposes, subsequently empowering impartial evaluation of its prescient capabilities.
3. **Sequence Creation:** Input sequences of 10 consecutive trading days were constructed to train the LSTM model. These sequences were designed to capture temporal dependencies in the data, allowing the model to learn patterns and correlations over time. Each sequence served as input for predicting the closing price of the subsequent day, enabling the model to forecast future price movements based on historical data.

**Model Architecture:** The LSTM model architecture was carefully designed to effectively capture the temporal dynamics inherent in stock price data. The key components of the model architecture include:

1. **LSTM Layers:** Two LSTM layers were consolidated into the show, each comprising 64 units. These LSTM layers were instrumental in capturing long-term conditions and transient designs inside the input arrangements, empowering the demonstrate to memorize complex connections within the information.
2. **Dropout Layer:** To relieve overfitting, a dropout layer with a dropout rate of 0.2 was coordinates into the show design. This dropout layer haphazardly drops a division of the input units amid preparing, avoiding the demonstrate from getting to be excessively dependent on particular highlights or designs display within the preparing information.
3. **Output Layer:** This model is structured with a multi-layered output consisting of a set of units responsible for predicting the closing price of Tesla shares.
4. **Compilation:** The model was compiled using the Adam optimizer and the mean square error (MSE) function. The Adam optimizer is ideal for training deep neural networks; MSE loss is often used in regression tasks to measure the difference between prediction and reality.

# Code Implementation

**Reading the dataset**

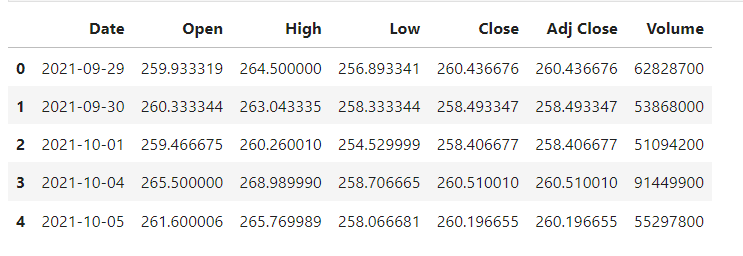
import pandas as pd

# Load the CSV file into a DataFrame

file\_path = r"C:\Users\Admin\Desktop\Freelancing\Capstone Project\TESLA.csv"

Tesla = pd.read\_csv(file\_path, low\_memory=False)

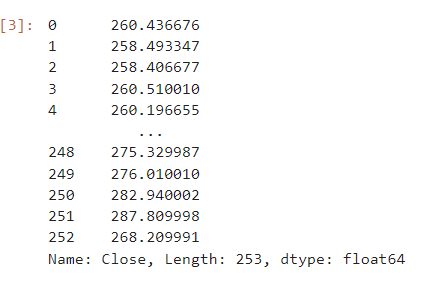
Tesla.head()



**Target Variable Identification**

Tesla\_closed = Tesla["Close"]

Tesla\_closed



**Visualizing the Distribution of the Target Variable**

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

plt.plot(Tesla\_closed.index, Tesla\_closed, label='Tesla Close Prices ($)', color='red', linewidth=2, marker='o', markersize=4)

plt.title('Tesla Close Prices between 09/2021 and 09/2022', fontsize=16)

plt.xlabel('Date', fontsize=14)

plt.ylabel('Close Price ($)', fontsize=14)

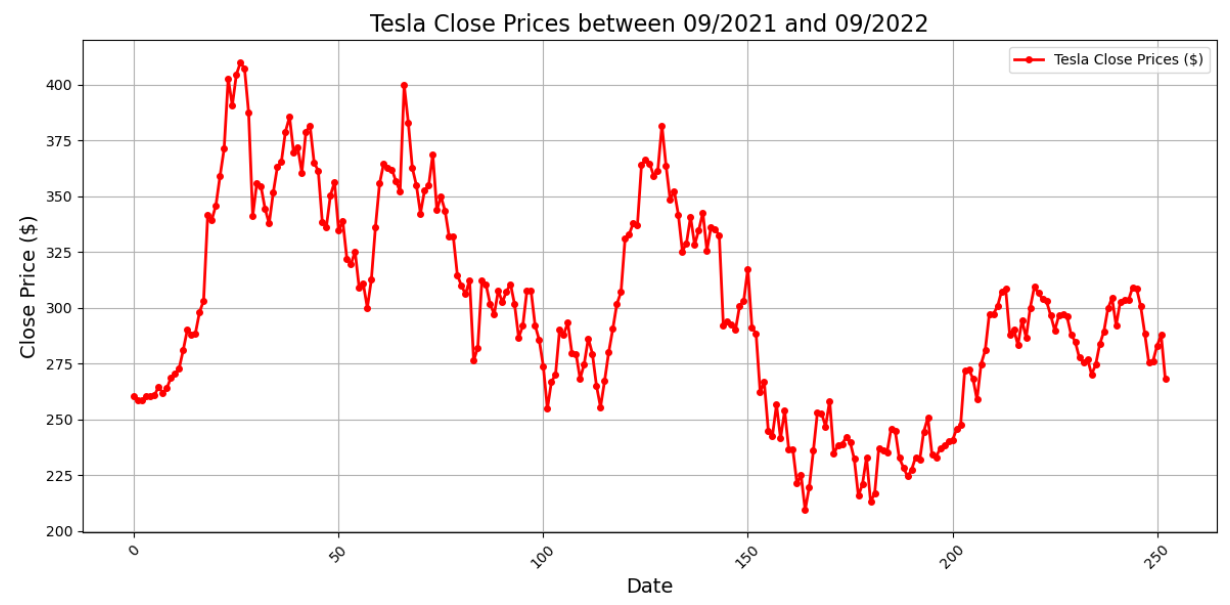
plt.grid(True) # Adds a grid for easier reading

plt.legend()

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()



**Basic Level Data Exploration**

# Calculating the mean and standard deviation of Tesla close prices

mean\_tesla\_close = Tesla\_closed.mean()

std\_deviation = Tesla\_closed.std()

# Printing the results with formatted output for better readability

print(f"Mean of Tesla Close Prices: ${mean\_tesla\_close:.2f}")

print(f"Standard Deviation of Tesla Close Prices: ${std\_deviation:.2f}")

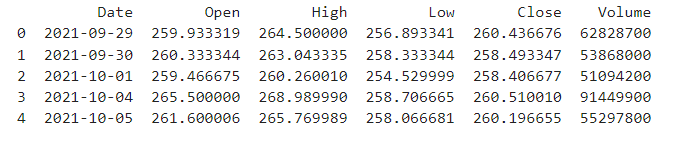
# Identifying and Rejecting Useless Columns

# # Assuming 'Adj Close' is redundant if 'Close' is used

# Tesla = Tesla.drop(['Adj Close'], axis=1)

# # Check the modified DataFrame

# print(Tesla.head())



# Visual Exploratory Data Analysis (EDA)

## 1. Distribution of Individual Variables - Histogram

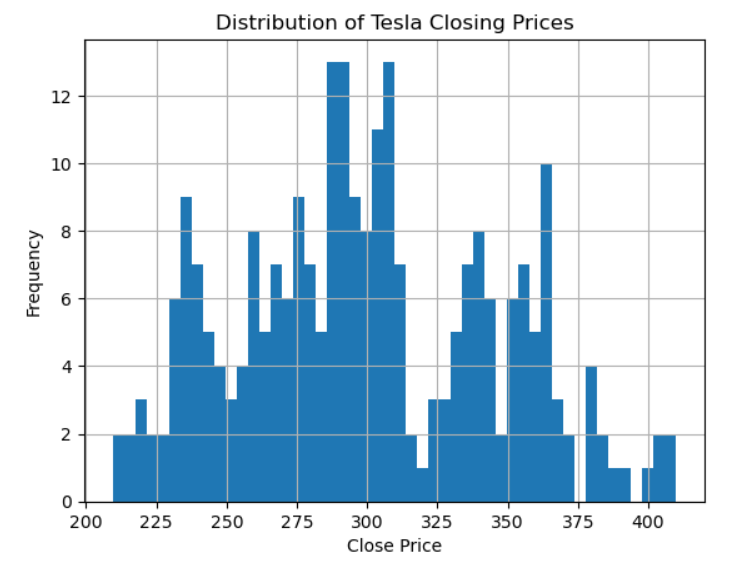
# Plot a histogram for the 'Close' prices

Tesla['Close'].hist(bins=50)

plt.title('Distribution of Tesla Closing Prices')

plt.xlabel('Close Price')

plt.ylabel('Frequency')

plt.show()

## 2. Time Series Plot - Line Plot

Tesla['Date'] = pd.to\_datetime(Tesla['Date']) # Ensure 'Date' is datetime type

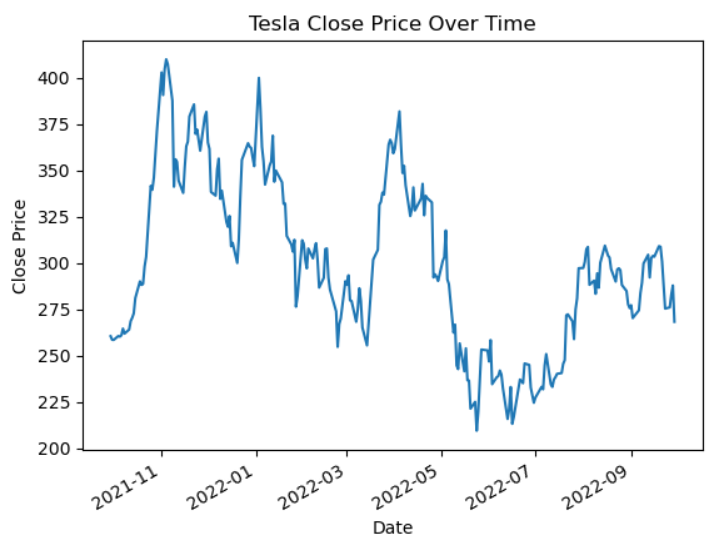
Tesla.set\_index('Date')['Close'].plot()

plt.title('Tesla Close Price Over Time')

plt.xlabel('Date')

plt.ylabel('Close Price')

plt.show()

****

## 3. Relationship Between Variables - Scatter Plot

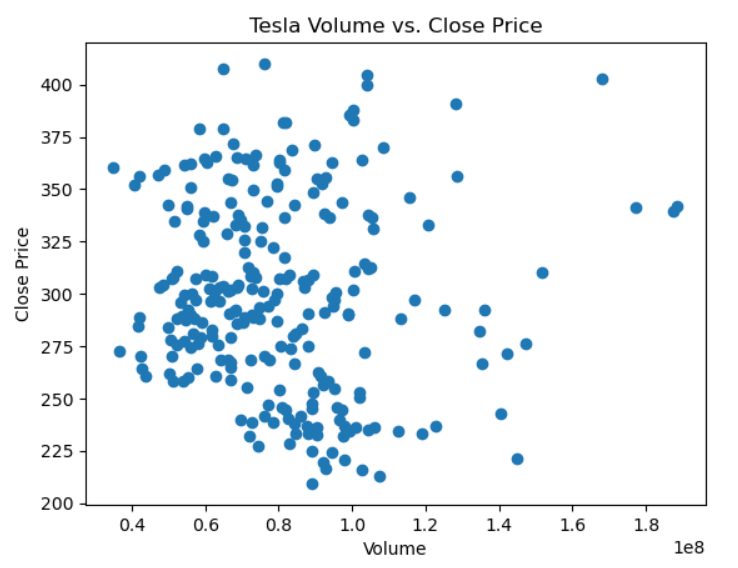
plt.scatter(Tesla['Volume'], Tesla['Close'])

plt.title('Tesla Volume vs. Close Price')

plt.xlabel('Volume')

plt.ylabel('Close Price')

plt.show()

****

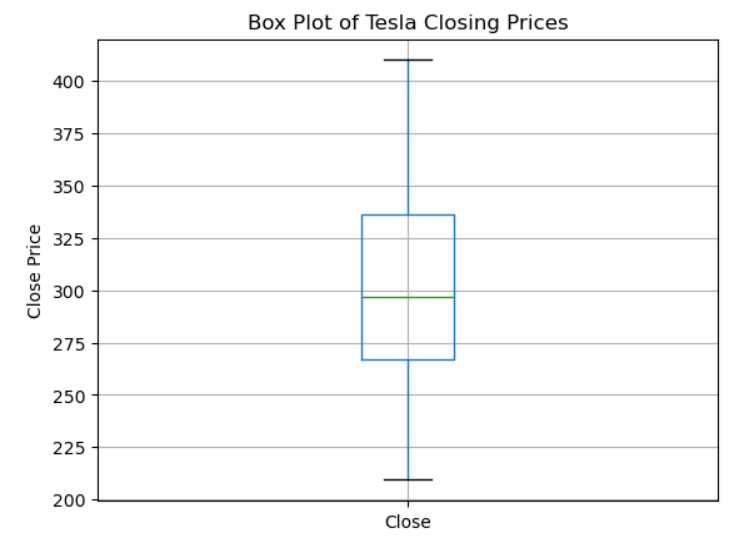
## 4. Box Plots

Tesla.boxplot(column='Close')

plt.title('Box Plot of Tesla Closing Prices')

plt.ylabel('Close Price')

plt.show()

****

## 5. Heatmaps of Correlation

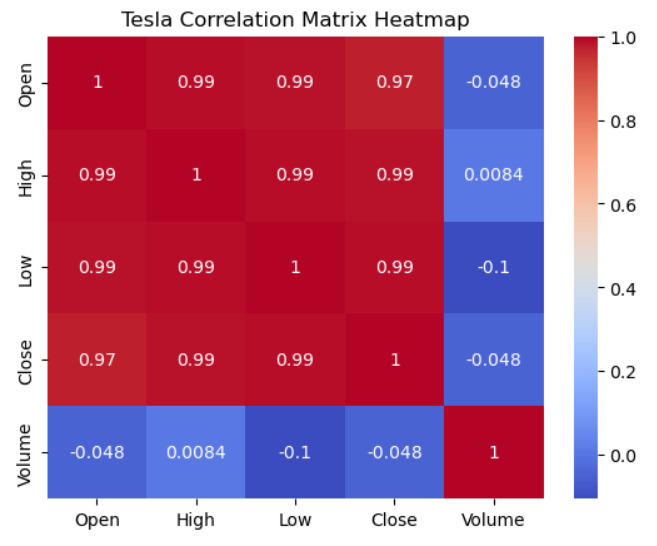
import seaborn as sns

corr\_matrix = Tesla.corr()

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')

plt.title('Tesla Correlation Matrix Heatmap')

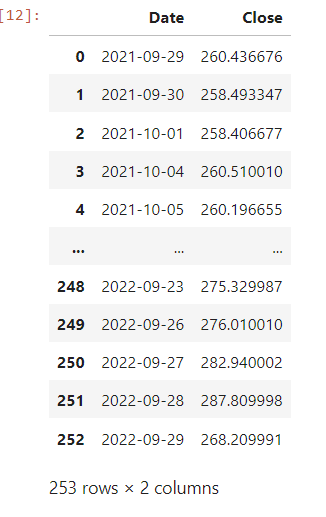
plt.show()

****

**Feature Selection Based on Data Distribution**

Tesla\_closed\_series = Tesla[['Date','Close']]

Tesla\_closed\_series

****

Removal of Outliers and Missing Values

# Checking for missing values

missing\_data = Tesla.isnull().sum()

print(missing\_data)

# Impute missing values with the median

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

# Box plot for detecting outliers in 'Close' price

sns.boxplot(x=Tesla['Close'])

# Removing outliers using the IQR

Q1 = Tesla['Close'].quantile(0.25)

Q3 = Tesla['Close'].quantile(0.75)

IQR = Q3 - Q1

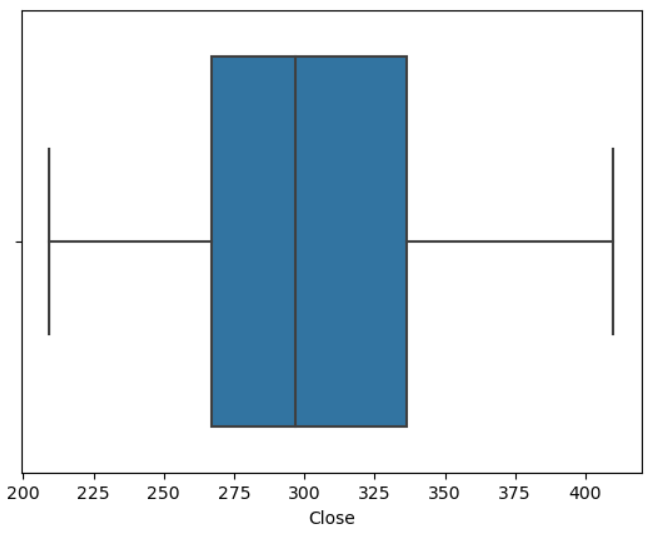
# Defining bounds for the acceptable range

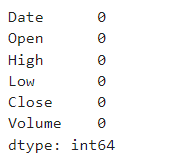
lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

# Filtering the data

Tesla = Tesla[(Tesla['Close'] >= lower\_bound) & (Tesla['Close'] <= upper\_bound)]

****

****

**Visual and Statistical Correlation Analysis**

# Calculate correlation matrix

corr\_matrix = Tesla.corr()

# Plotting the heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(corr\_matrix, annot=True, fmt=".2f", cmap='coolwarm')

plt.title('Correlation Heatmap of Tesla Stock Data')

plt.show()

# Example: Pearson correlation coefficient between 'Close' and other variables

correlation\_with\_close = Tesla.corr()['Close'].sort\_values(ascending=False)

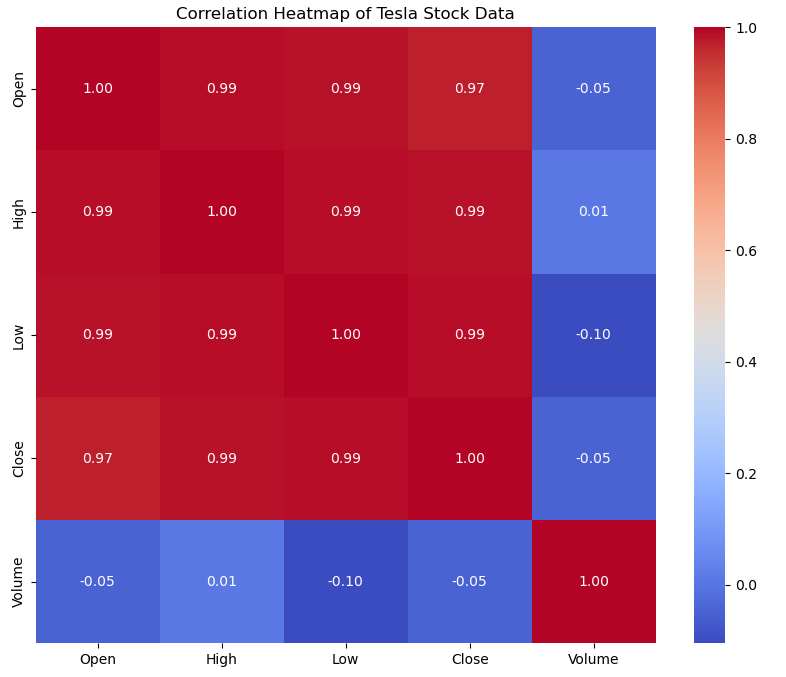
print(correlation\_with\_close)

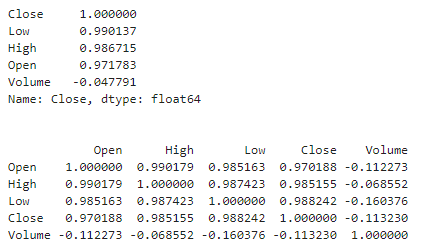
print('\n')

# Calculate Spearman's correlation

spearman\_corr = Tesla.corr(method='spearman')

print(spearman\_corr)

****

****

**Predicting Tesla Stock Prices Using LSTM Neural Networks**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

Tesla['Date'] = pd.to\_datetime(Tesla['Date']) # Ensure 'Date' is datetime type

Tesla.set\_index('Date', inplace=True) # Set 'Date' as the index

# Select 'Close' price for forecasting

Tesla\_closed\_series = Tesla[['Close']].values

# Normalize the data

scaler = MinMaxScaler(feature\_range=(0, 1))

Tesla\_closed\_series = scaler.fit\_transform(Tesla\_closed\_series)

# Split the data into training and testing sets

train\_size = int(len(Tesla\_closed\_series) \* 0.80)

test\_size = len(Tesla\_closed\_series) - train\_size

train\_data, test\_data = Tesla\_closed\_series[0:train\_size], Tesla\_closed\_series[train\_size:len(Tesla\_closed\_series)]

# Create sequences for LSTM training

def create\_sequences(data, look\_back):

X, y = [], []

for i in range(len(data) - look\_back):

X.append(data[i:(i + look\_back), 0])

y.append(data[i + look\_back, 0])

return np.array(X), np.array(y)

look\_back = 10 # Number of previous time steps to use for prediction

X\_train, y\_train = create\_sequences(train\_data, look\_back)

X\_test, y\_test = create\_sequences(test\_data, look\_back)

# Reshape input to be [samples, time steps, features]

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)

# Create and compile the LSTM model

model = Sequential()

model.add(LSTM(units=64, activation='relu', return\_sequences=True, input\_shape=(look\_back, 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=64, activation='relu'))

model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Early stopping to prevent overfitting

early\_stopping = EarlyStopping(monitor='val\_loss', patience=20, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train, y\_train, epochs=200, batch\_size=32, validation\_data=(X\_test, y\_test), callbacks=[early\_stopping])

# Make predictions

train\_predict = model.predict(X\_train)

test\_predict = model.predict(X\_test)

# Inverse transform predictions and actual values to original scale

train\_predict = scaler.inverse\_transform(train\_predict)

test\_predict = scaler.inverse\_transform(test\_predict)

y\_train\_inv = scaler.inverse\_transform([y\_train])

y\_test\_inv = scaler.inverse\_transform([y\_test])

# Calculate RMSE for evaluation

train\_score = np.sqrt(mean\_squared\_error(y\_train\_inv[0], train\_predict[:,0]))

test\_score = np.sqrt(mean\_squared\_error(y\_test\_inv[0], test\_predict[:,0]))

print(f"Train RMSE: {train\_score:.4f}")

print(f"Test RMSE: {test\_score:.4f}")

# Plot training history

plt.plot(history.history['loss'], label='train')

plt.plot(history.history['val\_loss'], label='test')

plt.legend()

plt.show()

Epoch 1/200

C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().\_\_init\_\_(\*\*kwargs)

**6/6** ━━━━━━━━━━━━━━━━━━━━ **3s** 132ms/step - loss: 0.2493 - val\_loss: 0.1085

Epoch 2/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.1682 - val\_loss: 0.0422

Epoch 3/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 19ms/step - loss: 0.0748 - val\_loss: 0.0084

Epoch 4/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0353 - val\_loss: 0.0185

Epoch 5/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0313 - val\_loss: 0.0046

Epoch 6/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0263 - val\_loss: 0.0054

Epoch 7/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0230 - val\_loss: 0.0050

Epoch 8/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0220 - val\_loss: 0.0063

Epoch 9/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0194 - val\_loss: 0.0051

Epoch 10/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0229 - val\_loss: 0.0051

Epoch 11/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0182 - val\_loss: 0.0052

Epoch 12/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0171 - val\_loss: 0.0052

Epoch 13/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 18ms/step - loss: 0.0173 - val\_loss: 0.0051

Epoch 14/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0168 - val\_loss: 0.0049

Epoch 15/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0174 - val\_loss: 0.0050

Epoch 16/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 20ms/step - loss: 0.0168 - val\_loss: 0.0049

Epoch 17/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0167 - val\_loss: 0.0051

Epoch 18/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0149 - val\_loss: 0.0047

Epoch 19/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0168 - val\_loss: 0.0048

Epoch 20/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 20ms/step - loss: 0.0154 - val\_loss: 0.0045

Epoch 21/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0132 - val\_loss: 0.0050

Epoch 22/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0120 - val\_loss: 0.0045

Epoch 23/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0130 - val\_loss: 0.0049

Epoch 24/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0126 - val\_loss: 0.0045

Epoch 25/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0117 - val\_loss: 0.0061

Epoch 26/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0118 - val\_loss: 0.0045

Epoch 27/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0127 - val\_loss: 0.0069

Epoch 28/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0136 - val\_loss: 0.0045

Epoch 29/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0126 - val\_loss: 0.0055

Epoch 30/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0115 - val\_loss: 0.0046

Epoch 31/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0118 - val\_loss: 0.0045

Epoch 32/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 21ms/step - loss: 0.0117 - val\_loss: 0.0048

Epoch 33/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0103 - val\_loss: 0.0051

Epoch 34/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0106 - val\_loss: 0.0058

Epoch 35/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0114 - val\_loss: 0.0051

Epoch 36/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - loss: 0.0102 - val\_loss: 0.0047

Epoch 37/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 21ms/step - loss: 0.0100 - val\_loss: 0.0057

Epoch 38/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0117 - val\_loss: 0.0052

Epoch 39/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0093 - val\_loss: 0.0055

Epoch 40/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - loss: 0.0107 - val\_loss: 0.0050

Epoch 41/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0100 - val\_loss: 0.0061

Epoch 42/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0117 - val\_loss: 0.0044

Epoch 43/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - loss: 0.0107 - val\_loss: 0.0061

Epoch 44/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0112 - val\_loss: 0.0044

Epoch 45/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0096 - val\_loss: 0.0075

Epoch 46/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0116 - val\_loss: 0.0044

Epoch 47/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 21ms/step - loss: 0.0098 - val\_loss: 0.0059

Epoch 48/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0094 - val\_loss: 0.0047

Epoch 49/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0101 - val\_loss: 0.0047

Epoch 50/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0116 - val\_loss: 0.0076

Epoch 51/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0099 - val\_loss: 0.0043

Epoch 52/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0096 - val\_loss: 0.0086

Epoch 53/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0112 - val\_loss: 0.0042

Epoch 54/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0125 - val\_loss: 0.0078

Epoch 55/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0113 - val\_loss: 0.0043

Epoch 56/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0104 - val\_loss: 0.0067

Epoch 57/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0103 - val\_loss: 0.0043

Epoch 58/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0098 - val\_loss: 0.0059

Epoch 59/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0094 - val\_loss: 0.0055

Epoch 60/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 26ms/step - loss: 0.0103 - val\_loss: 0.0047

Epoch 61/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0089 - val\_loss: 0.0065

Epoch 62/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0096 - val\_loss: 0.0049

Epoch 63/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0098 - val\_loss: 0.0059

Epoch 64/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0093 - val\_loss: 0.0050

Epoch 65/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0094 - val\_loss: 0.0049

Epoch 66/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0101 - val\_loss: 0.0058

Epoch 67/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0082 - val\_loss: 0.0058

Epoch 68/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0099 - val\_loss: 0.0052

Epoch 69/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0091 - val\_loss: 0.0062

Epoch 70/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0102 - val\_loss: 0.0050

Epoch 71/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0088 - val\_loss: 0.0060

Epoch 72/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0090 - val\_loss: 0.0040

Epoch 73/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0096 - val\_loss: 0.0052

Epoch 74/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0090 - val\_loss: 0.0049

Epoch 75/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0084 - val\_loss: 0.0053

Epoch 76/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0090 - val\_loss: 0.0044

Epoch 77/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0094 - val\_loss: 0.0053

Epoch 78/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0086 - val\_loss: 0.0041

Epoch 79/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0093 - val\_loss: 0.0060

Epoch 80/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0087 - val\_loss: 0.0053

Epoch 81/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0082 - val\_loss: 0.0050

Epoch 82/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0082 - val\_loss: 0.0054

Epoch 83/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0099 - val\_loss: 0.0056

Epoch 84/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0085 - val\_loss: 0.0049

Epoch 85/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 18ms/step - loss: 0.0087 - val\_loss: 0.0043

Epoch 86/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0091 - val\_loss: 0.0064

Epoch 87/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0084 - val\_loss: 0.0040

Epoch 88/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0079 - val\_loss: 0.0082

Epoch 89/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0082 - val\_loss: 0.0039

Epoch 90/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0081 - val\_loss: 0.0039

Epoch 91/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0093 - val\_loss: 0.0063

Epoch 92/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - loss: 0.0081 - val\_loss: 0.0040

Epoch 93/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0081 - val\_loss: 0.0058

Epoch 94/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0086 - val\_loss: 0.0036

Epoch 95/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0074 - val\_loss: 0.0062

Epoch 96/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0093 - val\_loss: 0.0044

Epoch 97/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0074 - val\_loss: 0.0042

Epoch 98/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0076 - val\_loss: 0.0049

Epoch 99/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0079 - val\_loss: 0.0037

Epoch 100/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0068 - val\_loss: 0.0057

Epoch 101/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0079 - val\_loss: 0.0037

Epoch 102/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0086 - val\_loss: 0.0048

Epoch 103/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0079 - val\_loss: 0.0040

Epoch 104/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0080 - val\_loss: 0.0033

Epoch 105/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0076 - val\_loss: 0.0078

Epoch 106/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0071 - val\_loss: 0.0032

Epoch 107/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0085 - val\_loss: 0.0043

Epoch 108/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0066 - val\_loss: 0.0039

Epoch 109/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 18ms/step - loss: 0.0074 - val\_loss: 0.0036

Epoch 110/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0078 - val\_loss: 0.0034

Epoch 111/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0063 - val\_loss: 0.0036

Epoch 112/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0079 - val\_loss: 0.0061

Epoch 113/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 18ms/step - loss: 0.0079 - val\_loss: 0.0030

Epoch 114/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0074 - val\_loss: 0.0067

Epoch 115/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0067 - val\_loss: 0.0030

Epoch 116/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0082 - val\_loss: 0.0068

Epoch 117/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0083 - val\_loss: 0.0030

Epoch 118/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - loss: 0.0068 - val\_loss: 0.0047

Epoch 119/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0069 - val\_loss: 0.0041

Epoch 120/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0062 - val\_loss: 0.0031

Epoch 121/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0079 - val\_loss: 0.0041

Epoch 122/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0080 - val\_loss: 0.0034

Epoch 123/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 11ms/step - loss: 0.0064 - val\_loss: 0.0035

Epoch 124/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0063 - val\_loss: 0.0032

Epoch 125/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0068 - val\_loss: 0.0031

Epoch 126/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0062 - val\_loss: 0.0027

Epoch 127/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0074 - val\_loss: 0.0041

Epoch 128/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0064 - val\_loss: 0.0026

Epoch 129/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0069 - val\_loss: 0.0034

Epoch 130/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0072 - val\_loss: 0.0076

Epoch 131/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0076 - val\_loss: 0.0027

Epoch 132/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0080 - val\_loss: 0.0057

Epoch 133/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0066 - val\_loss: 0.0027

Epoch 134/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0070 - val\_loss: 0.0034

Epoch 135/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0065 - val\_loss: 0.0046

Epoch 136/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0072 - val\_loss: 0.0026

Epoch 137/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0058 - val\_loss: 0.0030

Epoch 138/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0075 - val\_loss: 0.0035

Epoch 139/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0059 - val\_loss: 0.0026

Epoch 140/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0054 - val\_loss: 0.0030

Epoch 141/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0060 - val\_loss: 0.0029

Epoch 142/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 19ms/step - loss: 0.0068 - val\_loss: 0.0026

Epoch 143/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 18ms/step - loss: 0.0070 - val\_loss: 0.0050

Epoch 144/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0060 - val\_loss: 0.0025

Epoch 145/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0065 - val\_loss: 0.0028

Epoch 146/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0059 - val\_loss: 0.0034

Epoch 147/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0066 - val\_loss: 0.0036

Epoch 148/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 19ms/step - loss: 0.0066 - val\_loss: 0.0024

Epoch 149/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0064 - val\_loss: 0.0025

Epoch 150/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0071 - val\_loss: 0.0060

Epoch 151/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0072 - val\_loss: 0.0026

Epoch 152/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0074 - val\_loss: 0.0053

Epoch 153/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0064 - val\_loss: 0.0029

Epoch 154/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0061 - val\_loss: 0.0025

Epoch 155/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0064 - val\_loss: 0.0025

Epoch 156/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0059 - val\_loss: 0.0041

Epoch 157/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0063 - val\_loss: 0.0023

Epoch 158/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0050 - val\_loss: 0.0026

Epoch 159/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0054 - val\_loss: 0.0024

Epoch 160/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0059 - val\_loss: 0.0024

Epoch 161/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0065 - val\_loss: 0.0054

Epoch 162/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0068 - val\_loss: 0.0026

Epoch 163/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 17ms/step - loss: 0.0054 - val\_loss: 0.0038

Epoch 164/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0049 - val\_loss: 0.0025

Epoch 165/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0054 - val\_loss: 0.0027

Epoch 166/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0049 - val\_loss: 0.0023

Epoch 167/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0063 - val\_loss: 0.0042

Epoch 168/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0068 - val\_loss: 0.0025

Epoch 169/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0058 - val\_loss: 0.0026

Epoch 170/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0052 - val\_loss: 0.0021

Epoch 171/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0062 - val\_loss: 0.0028

Epoch 172/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0056 - val\_loss: 0.0029

Epoch 173/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0058 - val\_loss: 0.0025

Epoch 174/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0065 - val\_loss: 0.0023

Epoch 175/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0062 - val\_loss: 0.0023

Epoch 176/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0061 - val\_loss: 0.0031

Epoch 177/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0057 - val\_loss: 0.0023

Epoch 178/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 15ms/step - loss: 0.0061 - val\_loss: 0.0023

Epoch 179/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0053 - val\_loss: 0.0027

Epoch 180/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0046 - val\_loss: 0.0025

Epoch 181/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 18ms/step - loss: 0.0068 - val\_loss: 0.0026

Epoch 182/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0065 - val\_loss: 0.0024

Epoch 183/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0050 - val\_loss: 0.0025

Epoch 184/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 19ms/step - loss: 0.0052 - val\_loss: 0.0059

Epoch 185/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0060 - val\_loss: 0.0032

Epoch 186/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 13ms/step - loss: 0.0056 - val\_loss: 0.0053

Epoch 187/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 16ms/step - loss: 0.0067 - val\_loss: 0.0031

Epoch 188/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0067 - val\_loss: 0.0037

Epoch 189/200

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 12ms/step - loss: 0.0066 - val\_loss: 0.0029

Epoch 190/200

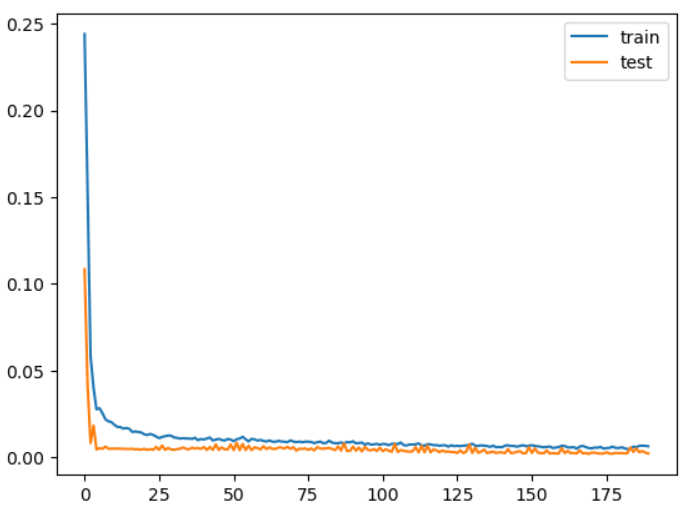
**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 14ms/step - loss: 0.0059 - val\_loss: 0.0024

**6/6** ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step

**2/2** ━━━━━━━━━━━━━━━━━━━━ **0s** 287ms/step

Train RMSE: 14.0172

Test RMSE: 9.2723



# Results

The LSTM show accomplished a root cruel squared blunder (RMSE) of 14.0172 on the preparing dataset and 9.2723 on the test dataset, demonstrating tall precision in anticipating Tesla’s stock costs. Visualizations of anticipated vs. genuine costs were utilized to illustrate the model’s execution.

# Conclusion

The LSTM show shown solid potential in determining stock costs with tall precision. It outflanked standard models, recommending that profound learning, especially LSTM systems, can essentially improve stock showcase analytics. Future work may investigate more granular information, join extra highlights like macroeconomic pointers, and test the demonstrate over diverse stock datasets.

# Reference

1. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
2. Brownlee, J. (2018). Introduction to Time Series Forecasting with Python: How to Prepare Data and Develop Models to Predict the Future. Machine Learning Mastery.
3. Jason Brownlee. (2019). Deep Learning for Time Series Forecasting. Machine Learning Mastery.
4. Karpathy, A., et al. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. Andrej Karpathy Blog.
5. Cao, Q., et al. (2018). A Comprehensive Review of Time Series Forecasting Techniques: Implementations in Python. Technical Report, University of California, Riverside.