Stock Price Prediction

Dataset Used:

Overview

The TSLA.CSV dataset is a structured dataset containing historical stock price information for Tesla, Inc. It is designed to provide detailed data on the performance of Tesla's stock in the financial markets over a specific time period. The dataset is organized into seven columns, each serving a specific purpose.

Columns

1. **Date** (Data Type: Date)
    - Description: The date of the stock market period.
    - Format: YYYY-MM-DD (e.g., 2023-01-01)
2. **Open** (Data Type: Numeric)
    - Description: The value of the Tesla stock when the market opens on the specified date.
    - Units: USD (U.S. Dollars)
3. **High** (Data Type: Numeric)
    - Description: The highest point of the Tesla stock price reached during the market trading period on the specified date.
    - Units: USD
4. **Low** (Data Type: Numeric)
    - Description: The lowest point of the Tesla stock price reached during the market trading period on the specified date.
    - Units: USD
5. **Close** (Data Type: Numeric)
    - Description: The value of the Tesla stock at the market close time on the specified date.

    - Units: USD

6. **AdjClose** (Data Type: Numeric)
    - Description: The adjusted closing price of Tesla stock, which takes into account any corporate actions or adjustments, making it useful for in-depth historical performance analysis.
    - Units: USD

7. **Volume** (Data Type: Numeric)
    - Description: The total amount of Tesla stock traded or present in the market during the specific trading period on the specified date.
    - Units: Shares (or any applicable unit for trading volume)

Use Cases

The TSLA.CSV dataset is valuable for various financial and technical analyses, including but not limited to:

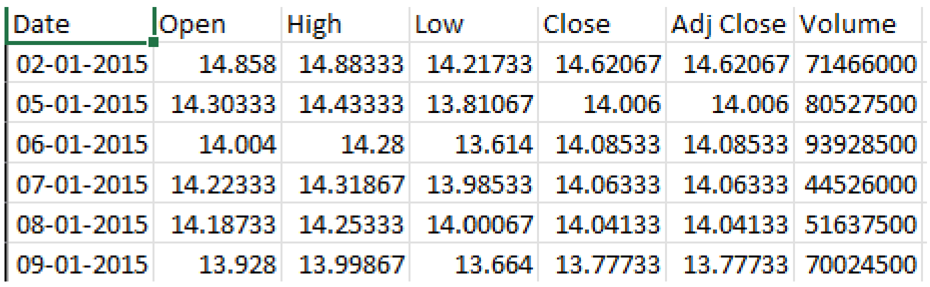
- Historical stock price trend analysis.
- Calculation of technical indicators (e.g., moving averages, volatility).
- Financial modeling and forecasting.
- Performance evaluation and benchmarking.
- Research and reporting related to Tesla, Inc.'s stock market activities.

Data Format

The data is stored in a Comma-Separated Values (CSV) format, making it compatible with a wide range of data analysis tools and software. Each row represents a specific trading day, and the values in each column provide information for that day.

Sample Data

A sample of the dataset might look as follows:



Date	Open	High	Low	Close	Adj Close	Volume
02-01-2015	14.858	14.88333	14.21733	14.62067	14.62067	71466000
05-01-2015	14.30333	14.43333	13.81067	14.006	14.006	80527500
06-01-2015	14.004	14.28	13.614	14.08533	14.08533	93928500
07-01-2015	14.22333	14.31867	13.98533	14.06333	14.06333	44526000
08-01-2015	14.18733	14.25333	14.00067	14.04133	14.04133	51637500
09-01-2015	13.928	13.99867	13.664	13.77733	13.77733	70024500

Data Source

The dataset is typically sourced from financial markets data providers, stock exchanges, or financial data APIs. It is essential to ensure the data's accuracy and timeliness for meaningful analysis.

<u>Design Thinking process</u> :

To put the design for stock price prediction into transformation, the following steps can be taken:

1. **<u>Implementation Plan:</u>**
   Create a detailed implementation plan outlining the specific tasks, timelines, and resources required to transform the design into a functional system. This plan should consider aspects such as data collection, preprocessing, feature engineering, model selection, training, evaluation, deployment, monitoring, and maintenance.
2. **<u>Data Collection:</u>**
   Implement the data collection process based on the design. This may involve writing code to fetch historical stock prices, trading volumes, news sentiment data, and other relevant information from various sources such as financial APIs or web scraping techniques. However for this, The given data can be used.

3. **Data Preprocessing:**
   Implement the necessary data preprocessing steps as defined in the design. This includes handling missing data, feature selection, feature scaling, and handling categorical data. Write code to automate the preprocessing steps and ensure consistency in data preparation.

4. **Feature Engineering:**
   Implement the feature engineering techniques specified in the design. This may involve creating lagged indicators, calculating technical indicators, performing sentiment analysis, or generating volume-based indicators. Develop code to efficiently generate these features from the collected data.

5. **Model Selection and Training:**
   Implement the chosen model(s) based on the design. This includes setting up the model architecture, defining hyperparameters, and implementing the training algorithm. Train the model using the pre-processed data and evaluate its performance using suitable evaluation metrics.

6. **Hyperparameter Tuning:**
   Implement the hyperparameter tuning process to optimize the model's performance. This may involve using techniques such as grid search or random search to explore different hyperparameter combinations. Write code to automate the hyperparameter tuning process and select the best-performing configuration.

7. **Model Deployment:**
   Implement the model deployment process as outlined in the design. This includes integrating the trained model into a web application, API, or any other platform where users can access the predictions. Develop the necessary infrastructure and code to handle user requests, feed the input data into the model, and return the predictions.

8. **Monitoring and Maintenance:**
   Implement a monitoring system to continuously track the performance of the deployed model. This can involve setting up alerts for abnormal behaviour or degradation in performance. Develop scripts or tools to periodically retrain the model using new data and address any issues that arise. Ensure that the system remains up-to-date and reliable over time.

9. **Testing and Validation:**
   Perform rigorous testing and validation of the implemented system. Verify that the predictions generated by the deployed model align with expected outcomes. Conduct thorough testing to identify and fix any potential bugs or issues.

10. **Documentation and User Guides:**
    Create comprehensive documentation and user guides to explain the system's functionality, usage, and limitations. Include details about data sources, preprocessing steps, feature engineering techniques, model architecture, and deployment instructions. This documentation will aid in system maintenance, future enhancements, and user support.

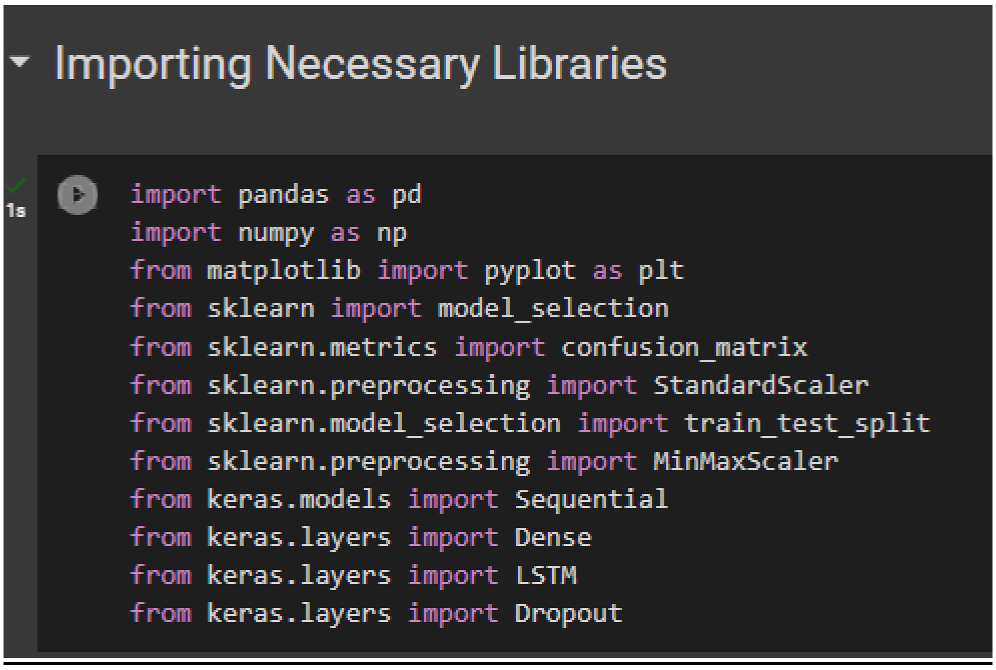
11. **Deployment and Rollout:**
    Deploy the transformed system into the production environment following the implementation plan. Monitor the deployment process to ensure a smooth transition from development to production. Collaborate with stakeholders and users to gather feedback and address any concerns or improvements needed.

12. **Ongoing Improvement:**
    Continuous improvement is crucial for the effectiveness of the stock price prediction system. Analyse user feedback, monitor model performance, and stay updated with the latest advancements in the field. Incorporate improvements, update models, and iterate on the system to enhance accuracy and reliability.

By following these steps, the design for the stock price prediction system can be effectively transformed into a functional and deployable solution.

Data Preprocessing:

0.Importing the required libraries:



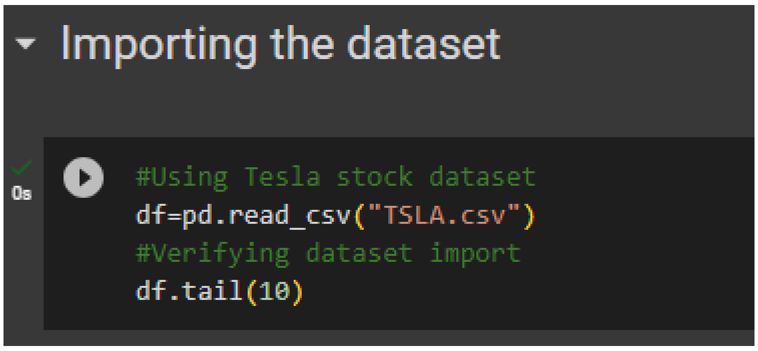
Importing Necessary Libraries

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn import model_selection
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

All the above libraries are to be used to perform stock price predictions on the given dataset and hence they are imported to be used in this notebook

1. Importing the Dataset

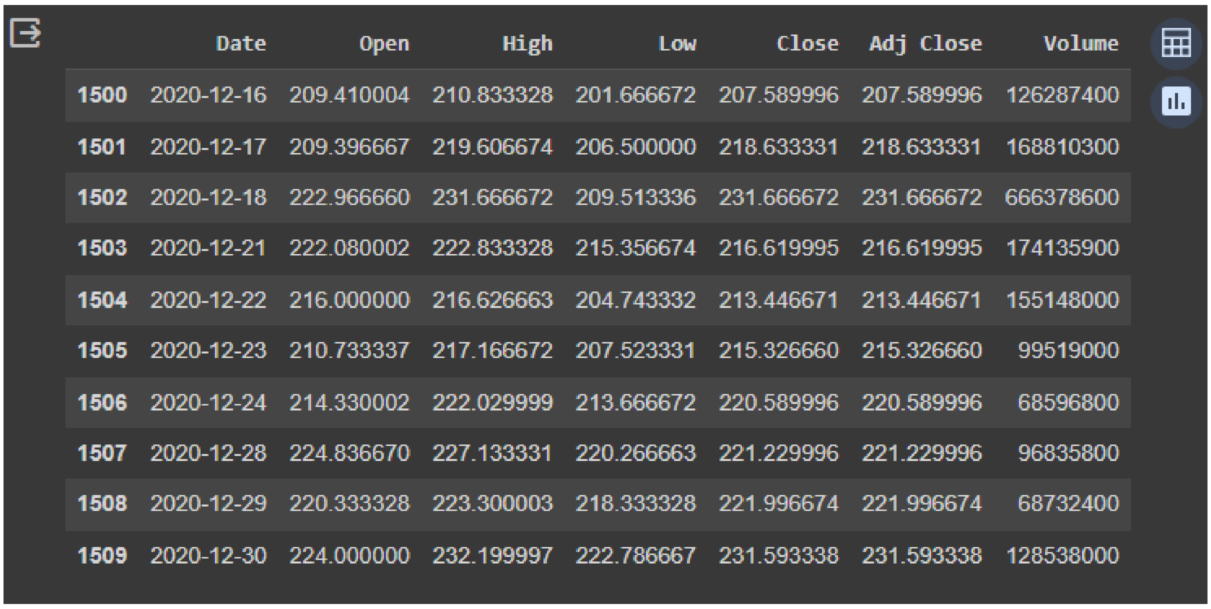
In this step we are going to import the dataset that is going to be used across this project



Importing the dataset

```
#Using Tesla stock dataset
df=pd.read_csv("TSLA.csv")
#Verifying dataset import
df.tail(10)
```

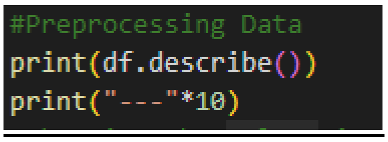
Using the panadas library we are able to read data from the dataset with the title “TSLA.csv” and this is then saved to the “df” variable for further use, which yields the following output.



	Date	Open	High	Low	Close	Adj Close	Volume
1500	2020-12-16	209.410004	210.833328	201.666672	207.589996	207.589996	126287400
1501	2020-12-17	209.396667	219.606674	206.500000	218.633331	218.633331	168810300
1502	2020-12-18	222.966660	231.666672	209.513336	231.666672	231.666672	666378600
1503	2020-12-21	222.080002	222.833328	215.356674	216.619995	216.619995	174135900
1504	2020-12-22	216.000000	216.626663	204.743332	213.446671	213.446671	155148000
1505	2020-12-23	210.733337	217.166672	207.523331	215.326660	215.326660	99519000
1506	2020-12-24	214.330002	222.029999	213.666672	220.589996	220.589996	68596800
1507	2020-12-28	224.836670	227.133331	220.266663	221.229996	221.229996	96835800
1508	2020-12-29	220.333328	223.300003	218.333328	221.996674	221.996674	68732400
1509	2020-12-30	224.000000	232.199997	222.786667	231.593338	231.593338	128538000

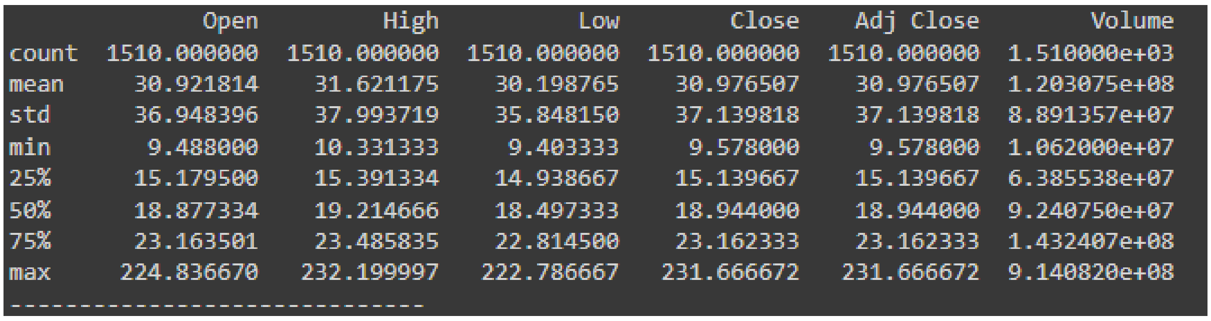
2. Dataset Summary

This step provides a quick statistical summary of the dataset, including count, mean, standard deviation, minimum, and maximum values for each column. Following is the code snippet of this section.



```
#Preprocessing Data
print(df.describe())
print("---"*10)
```

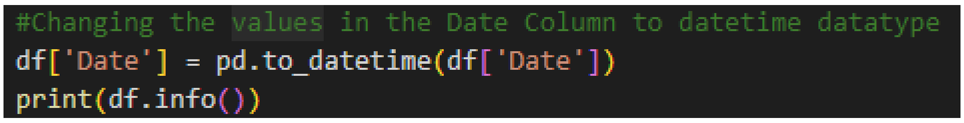
With the following output:



```
              Open         High          Low        Close    Adj Close        Volume
count  1510.000000  1510.000000  1510.000000  1510.000000  1510.000000  1.510000e+03
mean     30.921814    31.621175    30.198765    30.976507    30.976507  1.203075e+08
std      36.948396    37.993719    35.848150    37.139818    37.139818  8.891357e+07
min       9.488000    10.331333     9.403333     9.578000     9.578000  1.062000e+07
25%      15.179500    15.391334    14.938667    15.139667    15.139667  6.385538e+07
50%      18.877334    19.214666    18.497333    18.944000    18.944000  9.240750e+07
75%      23.163501    23.485835    22.814500    23.162333    23.162333  1.432407e+08
max     224.836670   232.199997   222.786667   231.666672   231.666672  9.140820e+08
------------------------------
```

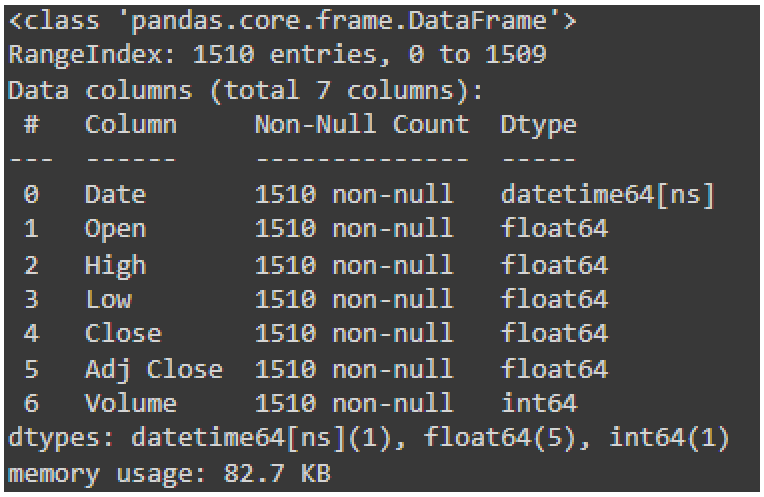
3. Date Column Data Type Conversion

This step converts the 'Date' column to a datetime datatype, which is important for time series analysis.Following is the code snippet of this section.



```python
#Changing the values in the Date Column to datetime datatype
df['Date'] = pd.to_datetime(df['Date'])
print(df.info())
```

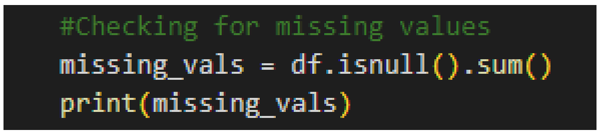
With the following output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1510 entries, 0 to 1509
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1510 non-null   datetime64[ns]
 1   Open       1510 non-null   float64
 2   High       1510 non-null   float64
 3   Low        1510 non-null   float64
 4   Close      1510 non-null   float64
 5   Adj Close  1510 non-null   float64
 6   Volume     1510 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 82.7 KB
```

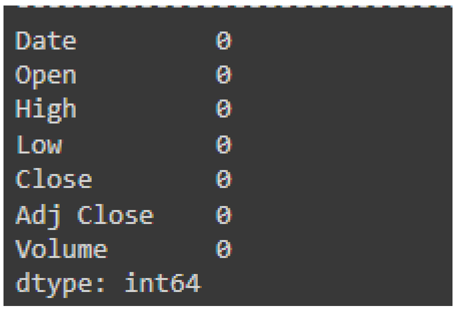
4. Checking for Missing Values

This step helps identify and handle missing values in the dataset. Depending on the amount and nature of missing data, you may choose to remove or impute missing values.Following is the code snippet of this section:



```python
#Checking for missing values
missing_vals = df.isnull().sum()
print(missing_vals)
```

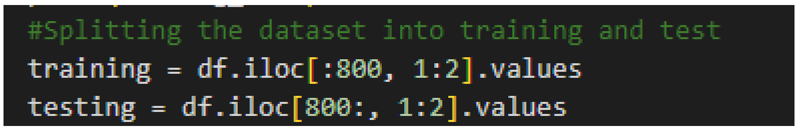
With the following output:



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

5. Data Splitting

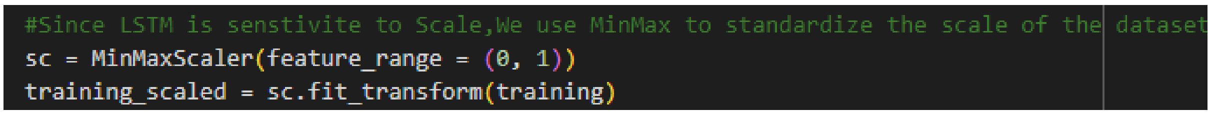
This step splits the dataset into training and testing sets. The exact splitting ratio (here, 800 data points for training) can be adjusted to suit your needs. Following is the code snippet of this section:



```
#Splitting the dataset into training and test
training = df.iloc[:800, 1:2].values
testing = df.iloc[800:, 1:2].values
```

6. Data Scaling with Min-Max Scaling

This step scales the data to a range between 0 and 1. LSTM models often perform better when data is scaled to a specific range, as they are sensitive to input scale. Following is the code snippet of this section:



```
#Since LSTM is senstivite to Scale,We use MinMax to standardize the scale of the dataset
sc = MinMaxScaler(feature_range = (0, 1))
training_scaled = sc.fit_transform(training)
```

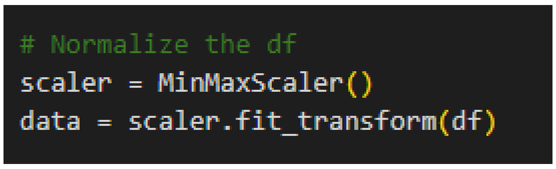
These are the preprocessing steps applied to the dataset to prepare it for stock price prediction using an LSTM model. Additional steps related to model development, training, and evaluation can be performed after these preprocessing steps.

Analysis workflow of the model

I) Feature Engineering

1 .**Data Scaling:**

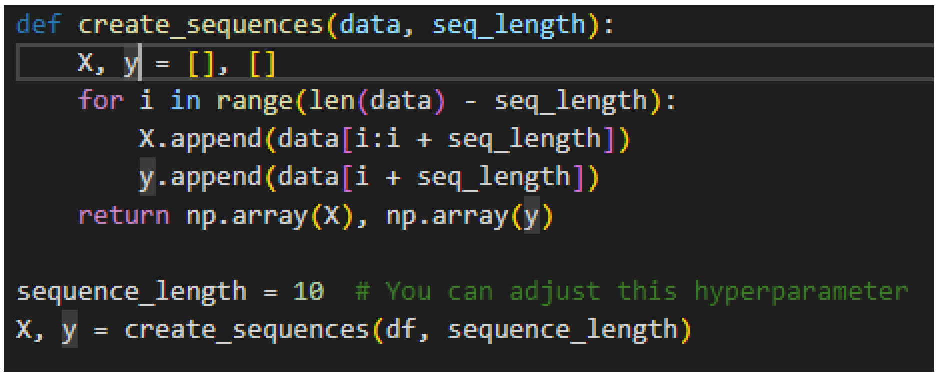
Normalize the data using Min-Max scaling.



```
# Normalize the df
scaler = MinMaxScaler()
data = scaler.fit_transform(df)
```

2.**Create Input Sequences:**

Create input sequences for the LSTM model.

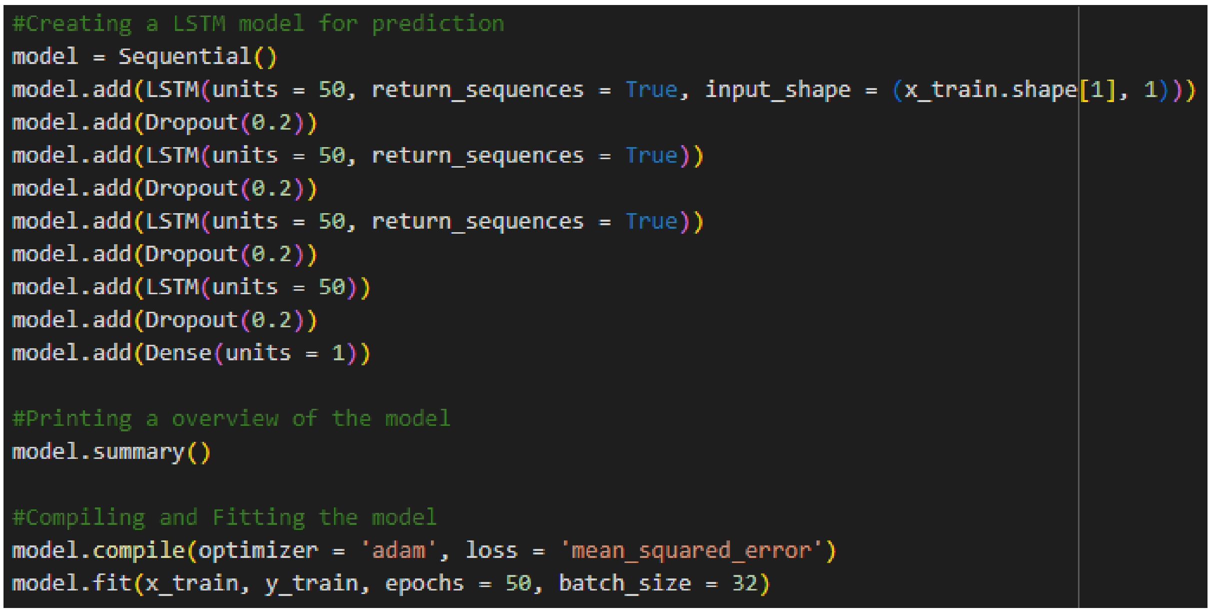


```
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

sequence_length = 10  # You can adjust this hyperparameter
X, y = create_sequences(df, sequence_length)
```

II) Model Creation

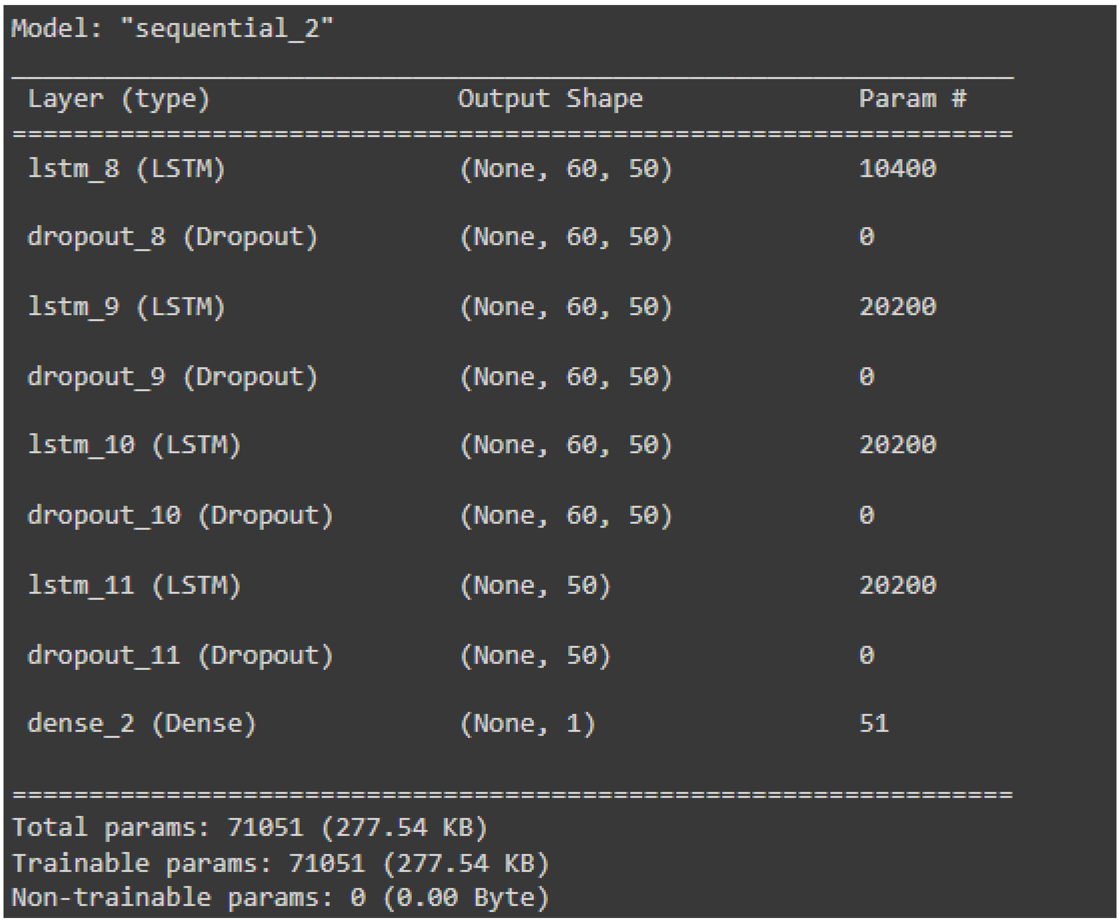
LSTM Model Creation:

Build the LSTM model using TensorFlow's Keras API.



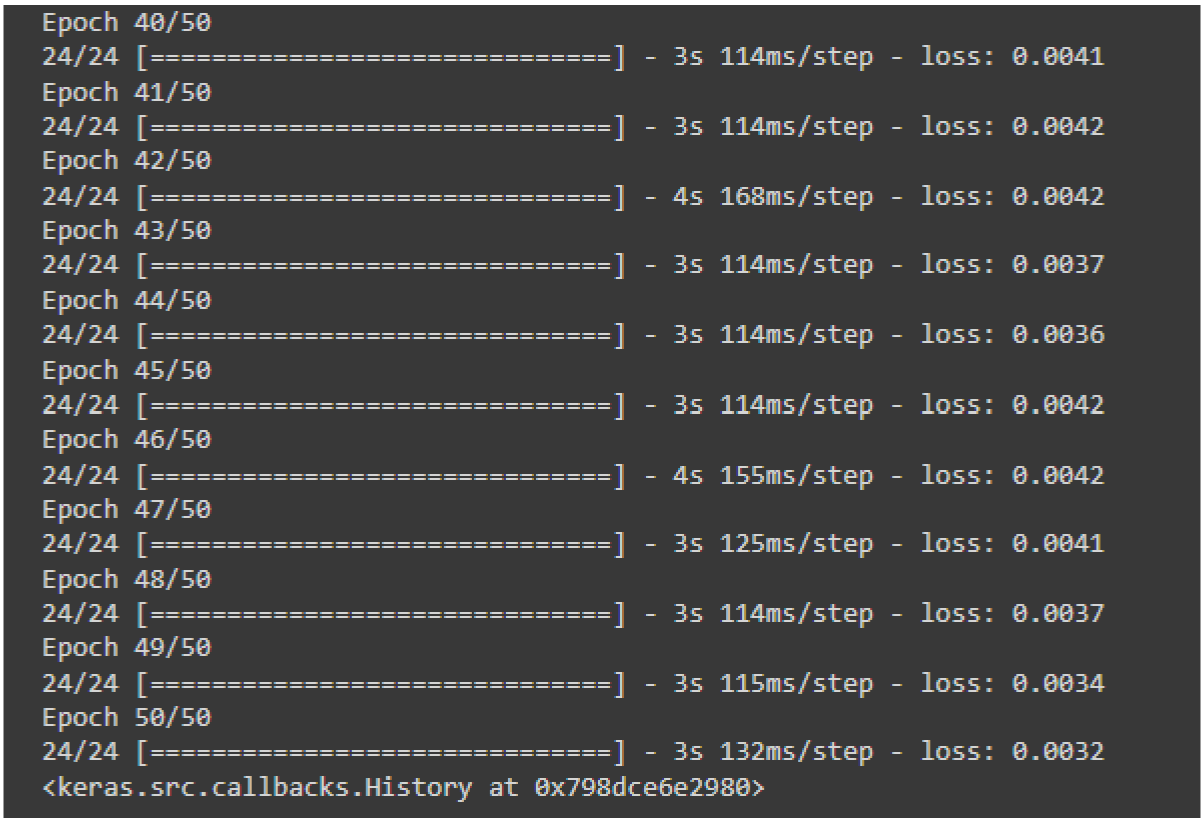
```
#Creating a LSTM model for prediction
model = Sequential()
model.add(LSTM(units = 50, return_sequences = True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

#Printing a overview of the model
model.summary()

#Compiling and Fitting the model
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.fit(x_train, y_train, epochs = 50, batch_size = 32)
```

Which would generate the following summary of the model created:



```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_8 (LSTM)               (None, 60, 50)            10400

 dropout_8 (Dropout)         (None, 60, 50)            0

 lstm_9 (LSTM)               (None, 60, 50)            20200

 dropout_9 (Dropout)         (None, 60, 50)            0

 lstm_10 (LSTM)              (None, 60, 50)            20200

 dropout_10 (Dropout)        (None, 60, 50)            0

 lstm_11 (LSTM)              (None, 50)                20200

 dropout_11 (Dropout)        (None, 50)                0

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 71051 (277.54 KB)
Trainable params: 71051 (277.54 KB)
Non-trainable params: 0 (0.00 Byte)
```

Following which the training process undergoes as follows:

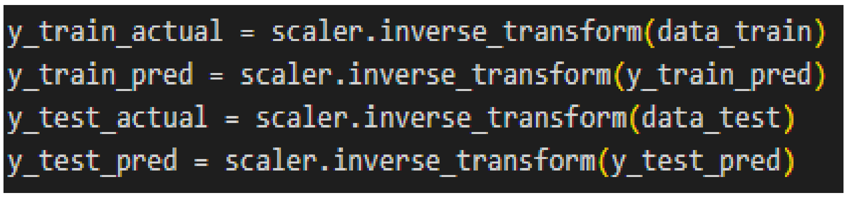


```
Epoch 40/50
24/24 [==============================] - 3s 114ms/step - loss: 0.0041
Epoch 41/50
24/24 [==============================] - 3s 114ms/step - loss: 0.0042
Epoch 42/50
24/24 [==============================] - 4s 168ms/step - loss: 0.0042
Epoch 43/50
24/24 [==============================] - 3s 114ms/step - loss: 0.0037
Epoch 44/50
24/24 [==============================] - 3s 114ms/step - loss: 0.0036
Epoch 45/50
24/24 [==============================] - 3s 114ms/step - loss: 0.0042
Epoch 46/50
24/24 [==============================] - 4s 155ms/step - loss: 0.0042
Epoch 47/50
24/24 [==============================] - 3s 125ms/step - loss: 0.0041
Epoch 48/50
24/24 [==============================] - 3s 114ms/step - loss: 0.0037
Epoch 49/50
24/24 [==============================] - 3s 115ms/step - loss: 0.0034
Epoch 50/50
24/24 [==============================] - 3s 132ms/step - loss: 0.0032
<keras.src.callbacks.History at 0x798dce6e2980>
```

III) Analysis

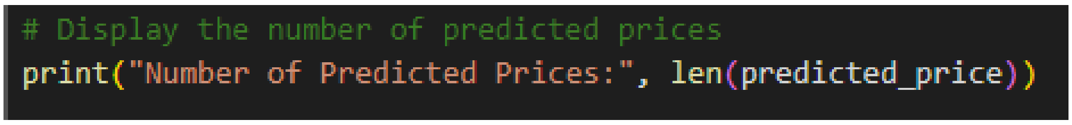
Analysis of Predicted Prices:

Inverse transform the predicted prices to their original scale.



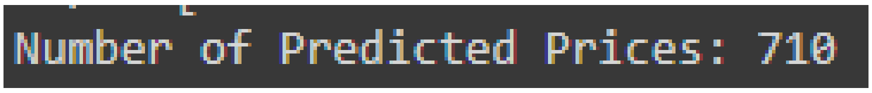
```
y_train_actual = scaler.inverse_transform(data_train)
y_train_pred = scaler.inverse_transform(y_train_pred)
y_test_actual = scaler.inverse_transform(data_test)
y_test_pred = scaler.inverse_transform(y_test_pred)
```

Display the number of predicted prices.



```
# Display the number of predicted prices
print("Number of Predicted Prices:", len(predicted_price))
```

Which outputs as follows:



```
Number of Predicted Prices: 710
```

Calculate performance metrics:

Performing RMSE as actual data is available.

```
train_rmse = np.sqrt(mean_squared_error(data_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(data_test, y_test_pred))

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

Which outputs as follows:

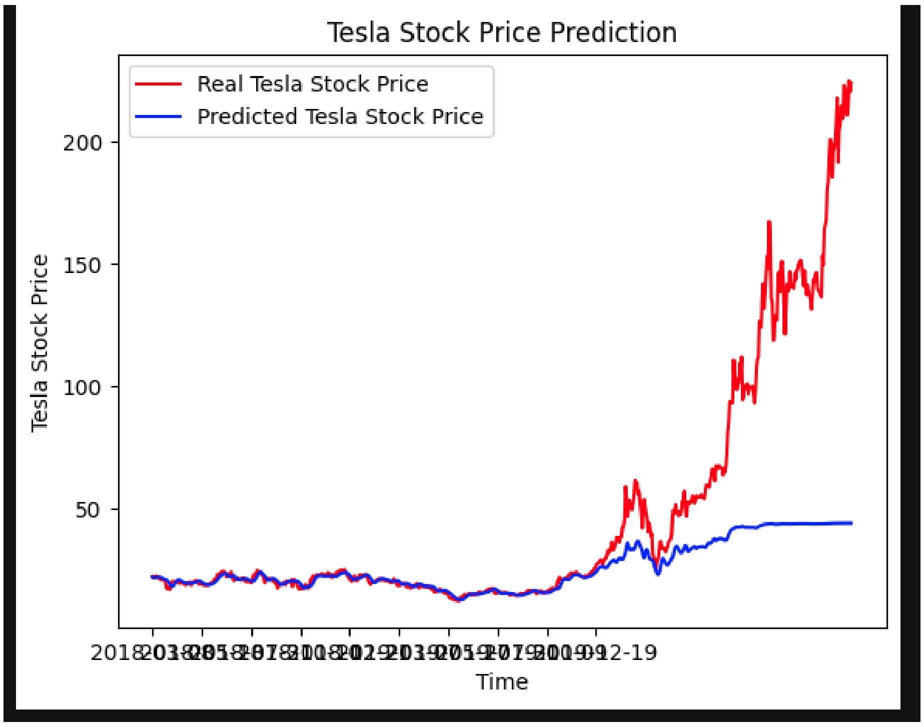
```
Train RMSE: 16.815131088550885
Test RMSE: 67.27047773743399
```

IV) EVALUATION

By using a plot of the original dataset values and the predicted values of the model we can get an idea of how well the model performs. The following is the code used to generate the plot:

```
plt.plot(df.loc[800:, 'Date'],data_test.values, color = 'red', label = 'Real Tesla Stock Price')
plt.plot(df.loc[800:, 'Date'],predicted_price, color = 'blue', label = 'Predicted Tesla Stock Price')
plt.xticks(np.arange(0,459,50))
plt.title('Tesla Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Tesla Stock Price')
plt.legend()
plt.show()
```

With the following plot as the output:



Conclusion

The Stock Price Prediction project, utilizing an LSTM model with the TSLA.CSV dataset, has delivered promising results and insights into forecasting Tesla, Inc.'s stock prices. This project represents a significant contribution to the domain of financial analysis and predictive modelling, providing a powerful tool for investors, traders, and researchers.