Project Title	**TCS Stock Data - Live and Latest**
language	Machine learning, python, SQL, Excel
Tools	VS code, Jupyter notebook
Domain	Data Analyst
Project Difficulties level	Advance

Dataset : Dataset is available in the given link. You can download it at your convenience.

[Click here to download data set]

About Dataset

Tata Consultancy Services (TCS) is an Indian multinational information technology (IT) services and consulting company headquartered in Mumbai, Maharashtra, India with its largest campus located in Chennai, Tamil Nadu, India. As of February 2021, TCS is the largest IT services company in the world by market capitalisation ($200 billion). It is a subsidiary of the Tata Group and operates in 149 locations across 46 countries.

TCS is the second largest Indian company by market capitalisation and is among the most valuable IT services brands worldwide.In 2015, TCS was ranked 64th overall in the Forbes World's Most Innovative Companies ranking, making it both the highest-ranked IT services company and the top Indian company. As of 2018, it is ranked eleventh on the Fortune India 500 list.In April 2018, TCS became the first Indian IT company to reach $100 billion in market capitalisation and second Indian company ever (after Reliance Industries achieved it in 2007) after its market capitalisation stood at ₹6.793 trillion (equivalent to ₹7.3 trillion or US$100 billion in 2019) on the Bombay Stock Exchange.

In 2016–2017, parent company Tata Sons owned 72.05% of TCS and more than 70% of Tata Sons' dividends were generated by TCS. In March 2018, Tata Sons decided to sell stocks of TCS worth $1.25 billion in a bulk deal.As of 15 September 2021, TCS has recorded a market capitalisation of US$200 billion, making it the first Indian IT firm to do so.

Here's a complete outline for a **Machine Learning Project on TCS Stock Data Analysis**. This project includes **data preprocessing, exploratory data analysis (EDA), and visualization**, along with machine learning **modeling to predict stock prices** based on historical data.

Project Title: TCS Stock Data Analysis and Prediction

Objective

Analyze the historical data of TCS stock to gain insights into stock behavior, identify trends, and forecast future stock prices.

Dataset Columns Explanation

1. **Date** - Date of trading data.
2. **Open** - Opening stock price on that day.
3. **High** - Highest stock price of the day.
4. **Low** - Lowest stock price of the day.
5. **Close** - Closing stock price of the day.
6. **Volume** - Number of shares traded.
7. **Dividends** - Dividends paid on the stock.
8. **Stock Splits** - Number of stock splits.

Step-by-Step Project Outline

Step 1: Import Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import datetime
```

Step 2: Load the Dataset

```python
data = pd.read_csv('TCS_stock_data.csv')  # Replace with the
correct path to your dataset
data['Date'] = pd.to_datetime(data['Date'])
data.sort_values(by='Date', inplace=True)
data.head()
```

Step 3: Data Preprocessing

- Check for null values and handle them.
- Convert necessary columns to numeric if needed.
- Check for any outliers in the data, especially in `Volume` and `Close` price.

```python
# Check for null values
print(data.isnull().sum())

# Convert numeric columns if required
data['Open'] = pd.to_numeric(data['Open'], errors='coerce')
data['High'] = pd.to_numeric(data['High'], errors='coerce')
data['Low'] = pd.to_numeric(data['Low'], errors='coerce')
data['Close'] = pd.to_numeric(data['Close'], errors='coerce')

# Fill any remaining NaN values
data.fillna(method='ffill', inplace=True)
```

Step 4: Exploratory Data Analysis (EDA)

- **Price Trends**: Visualize the Open, Close, High, and Low prices over time.
- **Volume Analysis**: Analyze trading volumes.
- **Moving Averages**: Calculate moving averages for trend analysis.

```
# Plotting Close price over time
plt.figure(figsize=(14, 7))
plt.plot(data['Date'], data['Close'], color='blue',
label='Close Price')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('TCS Stock Close Price Over Time')
plt.legend()
plt.show()

# Calculating 50-day and 200-day moving averages
data['MA50'] = data['Close'].rolling(window=50).mean()
data['MA200'] = data['Close'].rolling(window=200).mean()

# Plot with Moving Averages
plt.figure(figsize=(14, 7))
plt.plot(data['Date'], data['Close'], label='Close Price')
plt.plot(data['Date'], data['MA50'], label='50-Day MA')
plt.plot(data['Date'], data['MA200'], label='200-Day MA')
```

```
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('TCS Stock Price with Moving Averages')
plt.legend()
plt.show()
```

Step 5: Feature Engineering

- Extract features like **Year**, **Month**, **Day**, **Day of Week** from Date.
- Create lag features (e.g., previous day’s close, previous day’s high/low).

```
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['Day'] = data['Date'].dt.day
data['Day_of_Week'] = data['Date'].dt.dayofweek

# Lag Features
data['Prev_Close'] = data['Close'].shift(1)
data.dropna(inplace=True)  # Drop rows with NaN values from
shifting
```

Step 6: Model Building and Prediction

- Use **Linear Regression** to predict the **Close** price based on features.
- **Train/Test Split** for model evaluation.

```
# Feature selection
X = data[['Open', 'High', 'Low', 'Volume', 'Prev_Close',
'Day_of_Week', 'Month']]
y = data['Close']

# Split into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation
print("Mean Squared Error:", mean_squared_error(y_test,
y_pred))
print("R-Squared Score:", r2_score(y_test, y_pred))
```

Step 7: Visualize Model Performance

- Plot predicted vs. actual values.
- Scatter plot to observe prediction accuracy.

```
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred, color='blue', label='Predicted vs
Actual')
plt.xlabel('Actual Close Price')
plt.ylabel('Predicted Close Price')
plt.title('Actual vs Predicted Close Price')
plt.legend()
plt.show()
```

Step 8: Save the Model (Optional)

- Save the trained model for future use.

```
import pickle
with open('TCS_Stock_Predictor.pkl', 'wb') as file:
    pickle.dump(model, file)
```

Step 9: Future Work & Interpretation

- Test different models (Random Forest, XGBoost).
- Use hyperparameter tuning for optimization.
- Explore time-series models like ARIMA for better predictions based on temporal data.

Explanation Summary

This project covers EDA, visualization, feature engineering, and prediction modeling for TCS stock prices:

1. **EDA** provides insights into the stock's historical patterns.
2. **Moving Averages** help smooth out price trends.
3. **Linear Regression** is used to predict closing prices.
4. **Evaluation metrics** help validate the model’s accuracy, giving insight into its reliability.

Sample code and output

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import mean_absolute_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tqdm import tqdm
```

```python
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"


/kaggle/input/tcs-stock-data-live-and-latest/TCS_stock_action.csv
/kaggle/input/tcs-stock-data-live-and-latest/TCS_stock_history.csv
```

```
/kaggle/input/tcs-stock-data-live-and-latest/TCS_stock_info.csv
```

**Loading Data

In [2]:

```
df=pd.read_csv('/kaggle/input/tcs-stock-data-live-and-latest/TCS_stock_history.csv')
df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2002-08-12	28.794172	29.742206	28.794172	29.519140	212976	0.0	0.0
1	2002-08-13	29.556316	30.030333	28.905705	29.119476	153576	0.0	0.0
2	2002-0	29.184	29.184	26.563	27.111	822	0.0	0.0

	8-14	536	536	503	877	776		
3	2002-08-15	27.111877	27.111877	27.111877	27.111877	0	0.0	0.0
4	2002-08-16	26.972458	28.255089	26.582090	27.046812	811856	0.0	0.0

In [3]:

```
df.columns
```

Out[3]:

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume',
'Dividends',
       'Stock Splits'],

      dtype='object')
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4463 entries, 0 to 4462
```

```
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          4463 non-null   object
 1   Open          4463 non-null   float64
 2   High          4463 non-null   float64
 3   Low           4463 non-null   float64
 4   Close         4463 non-null   float64
 5   Volume        4463 non-null   int64
 6   Dividends     4463 non-null   float64
 7   Stock Splits  4463 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 279.1+ KB
```

In [5]:

```
df['Date'] = pd.to_datetime(df['Date'])
df = df.sort_values(by='Date')
```

In [6]:

```
df.describe()
```

Out[6]:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
count	4463.000000	4463.000000	4463.000000	4463.000000	4.463000e+03	4463.000000	4463.000000
mean	866.936239	876.675013	856.653850	866.537398	3.537876e+06	0.071533	0.001344
std	829.905368	838.267104	821.233477	829.611313	3.273531e+06	0.965401	0.051842
min	24.146938	27.102587	24.146938	26.377609	0.000000e+00	0.000000	0.000000
25%	188.951782	191.571816	185.979417	188.594620	1.860959e+06	0.000000	0.000000
50%	530.907530	534.751639	525.616849	529.713257	2.757742e+06	0.000000	0.000000

75%	1156.462421	1165.815854	1143.622800	1154.784851	4.278625e+06	0.000000	0.000000
max	3930.000000	3981.750000	3892.100098	3954.550049	8.806715e+07	40.000000	2.000000

**Correlation Of Features

In [7]:

```
corel=df.corr()
corel
```

Out[7]:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Open	1.000000	0.999888	0.999892	0.999787	-0.153362	0.059743	-0.006715
High	0.999	1.000	0.999	0.999	-0.150	0.060	-0.0065

	888	000	867	914	918	044	97
Low	0.999892	0.999867	1.000000	0.999901	-0.154962	0.059916	-0.006622
Close	0.999787	0.999914	0.999901	1.000000	-0.152844	0.060179	-0.006635
Volume	-0.153362	-0.150918	-0.154962	-0.152844	1.000000	-0.010332	0.004752
Dividends	0.059743	0.060044	0.059916	0.060179	-0.010332	1.000000	0.142493
Stock Splits	-0.006715	-0.006597	-0.006622	-0.006635	0.004752	0.142493	1.000000

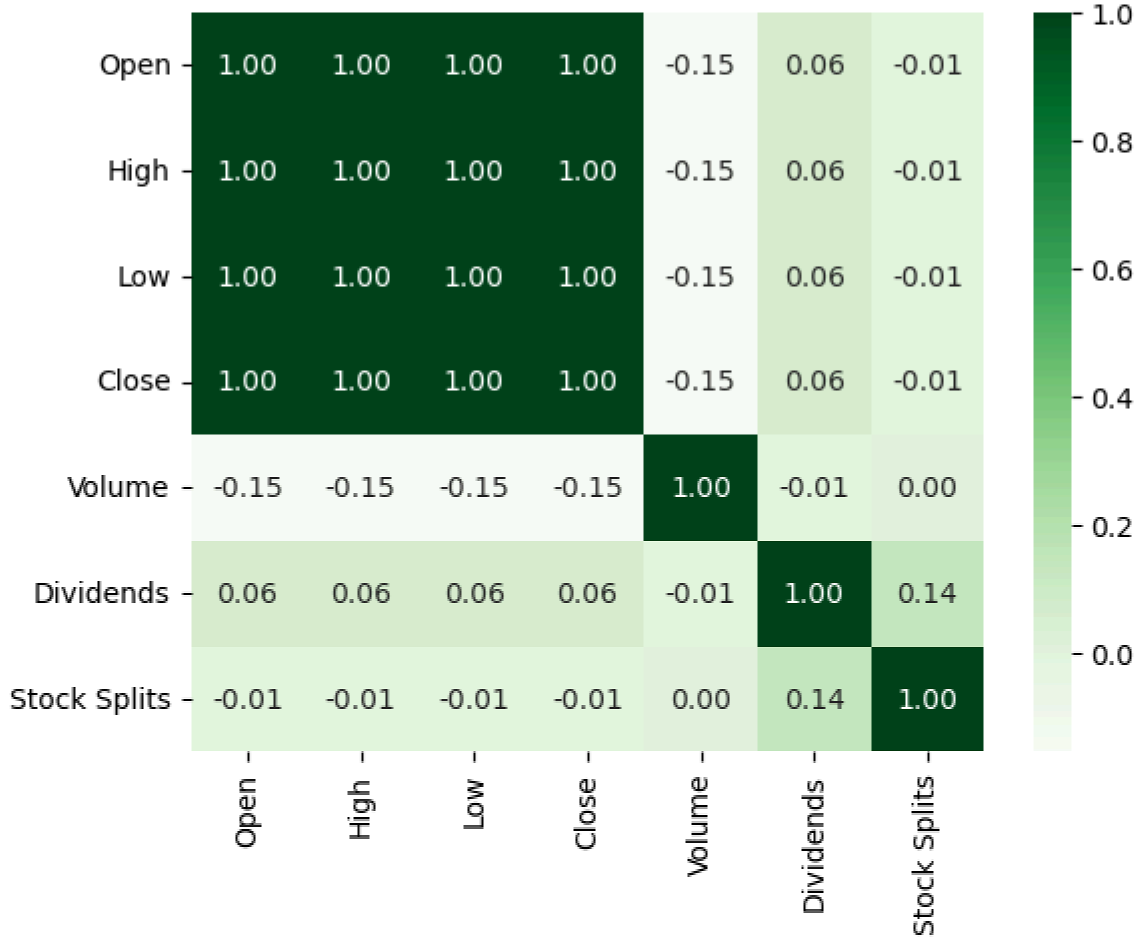
In [8]:

```
# Correlation of features with the target variable (Close Price)
correlation_with_close =
df.corr()['Close'].sort_values(ascending=False)
print(correlation_with_close)
```

```
Close             1.000000
High              0.999914
Low               0.999901
Open              0.999787
Dividends         0.060179
Stock Splits     -0.006635
Volume           -0.152844
Name: Close, dtype: float64
```

In [9]:

```
sns.heatmap(corel,annot= True,cmap= "Greens",fmt=".2f")
plt.show()
```



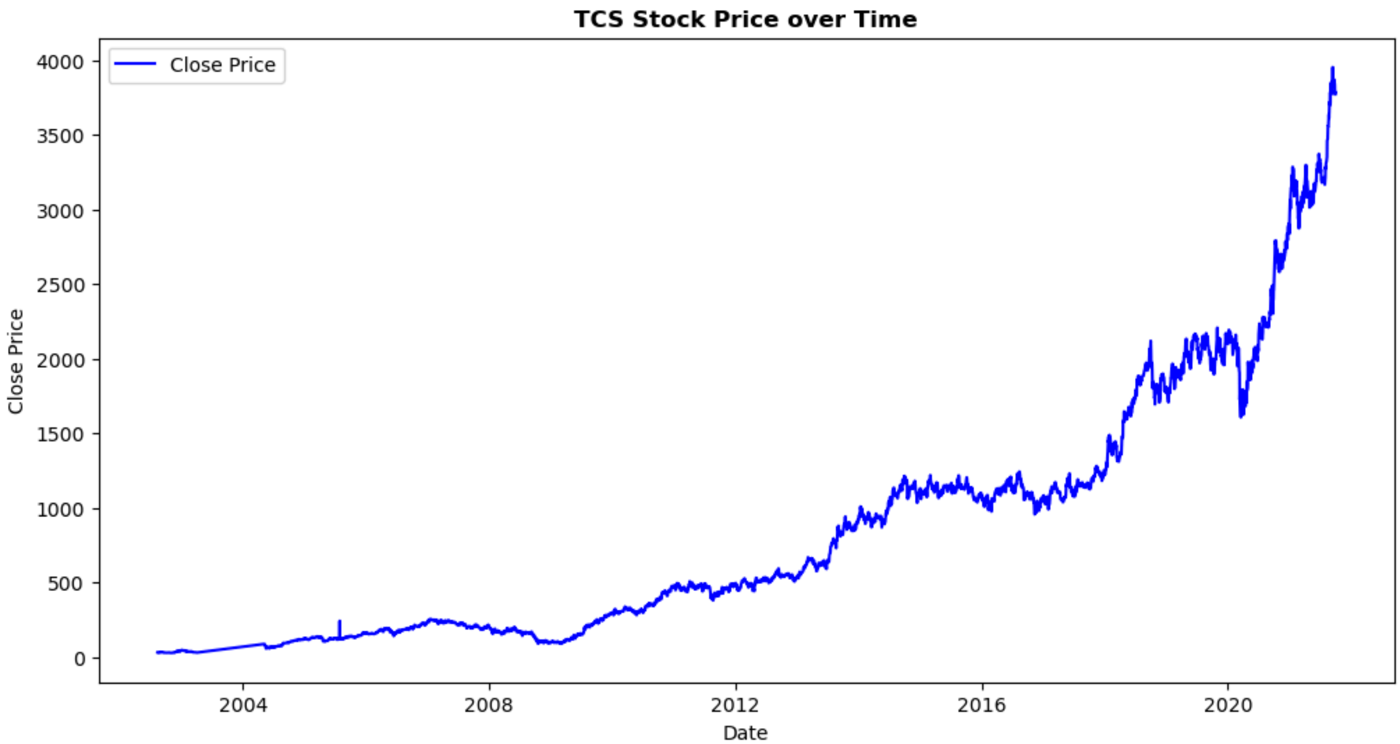
**Exploratory Data Analysis

Time series of stock prices

In [10]:

```
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Close Price',
color='b')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('TCS Stock Price over Time',weight = "bold")
plt.legend()
```

```
plt.show()
```

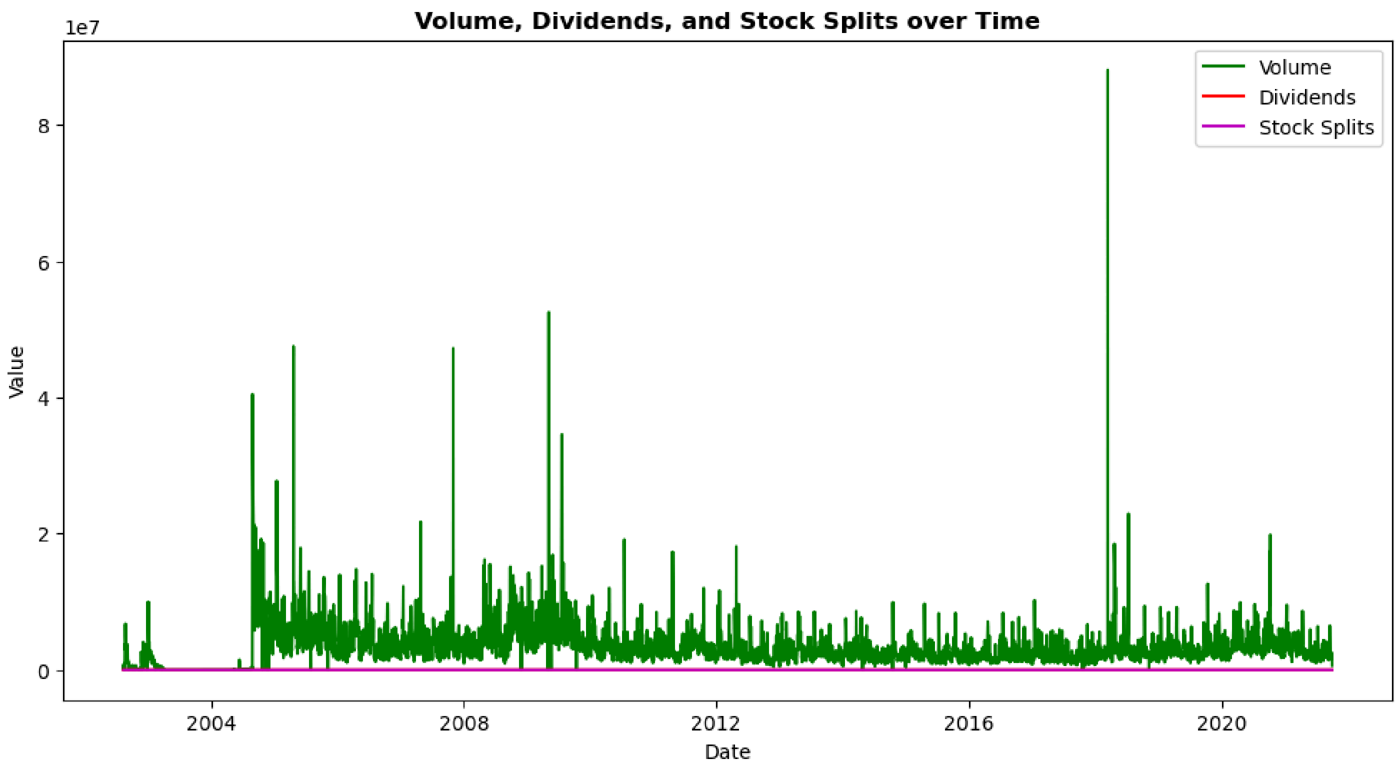


Volume, Dividends, Stock Splits

In [11]:

```
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Volume'], label='Volume', color='g')
plt.plot(df['Date'], df['Dividends'], label='Dividends',
color='r')
plt.plot(df['Date'], df['Stock Splits'], label='Stock Splits',
color='m')
plt.xlabel('Date')
plt.ylabel('Value')
```

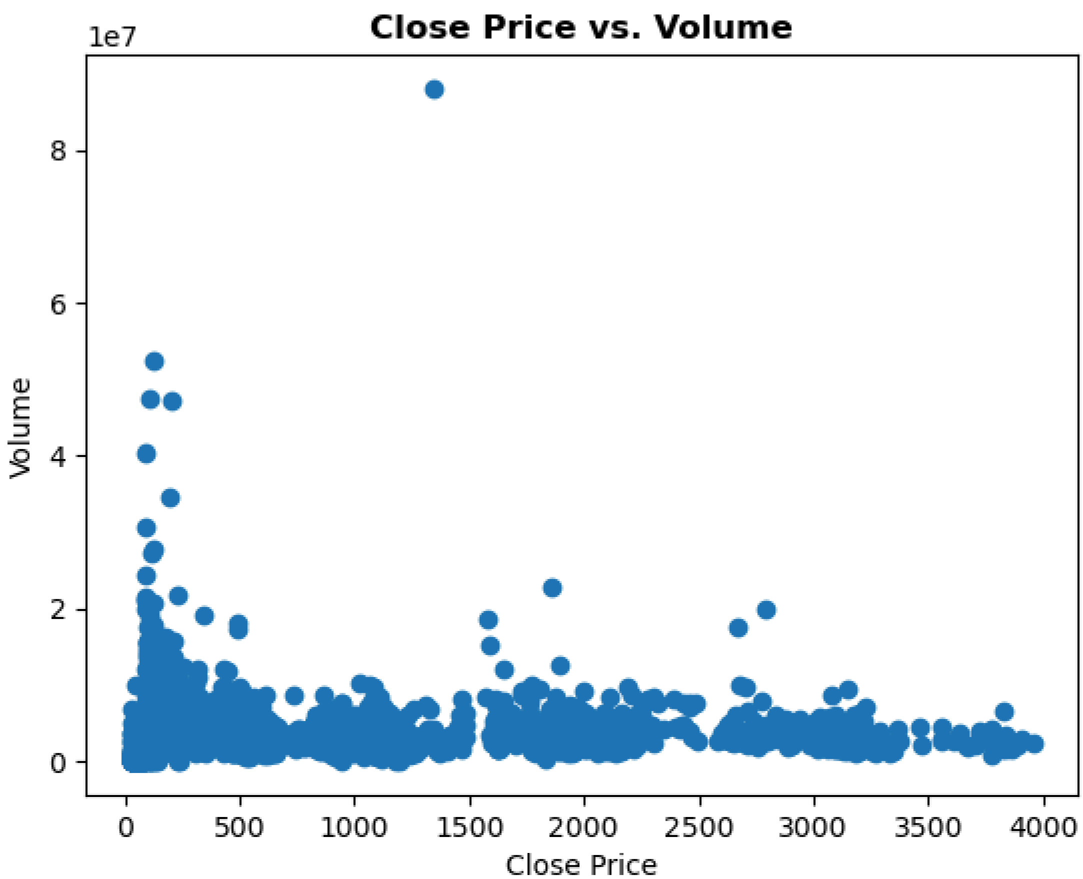
```
plt.title('Volume, Dividends, and Stock Splits over
Time',weight = "bold")
plt.legend()
plt.show()
```



Close vs Volume

In [12]:

```
plt.scatter(df['Close'], df['Volume'])
plt.xlabel('Close Price')
plt.ylabel('Volume')
plt.title('Close Price vs. Volume',weight= "bold")
plt.show()
```

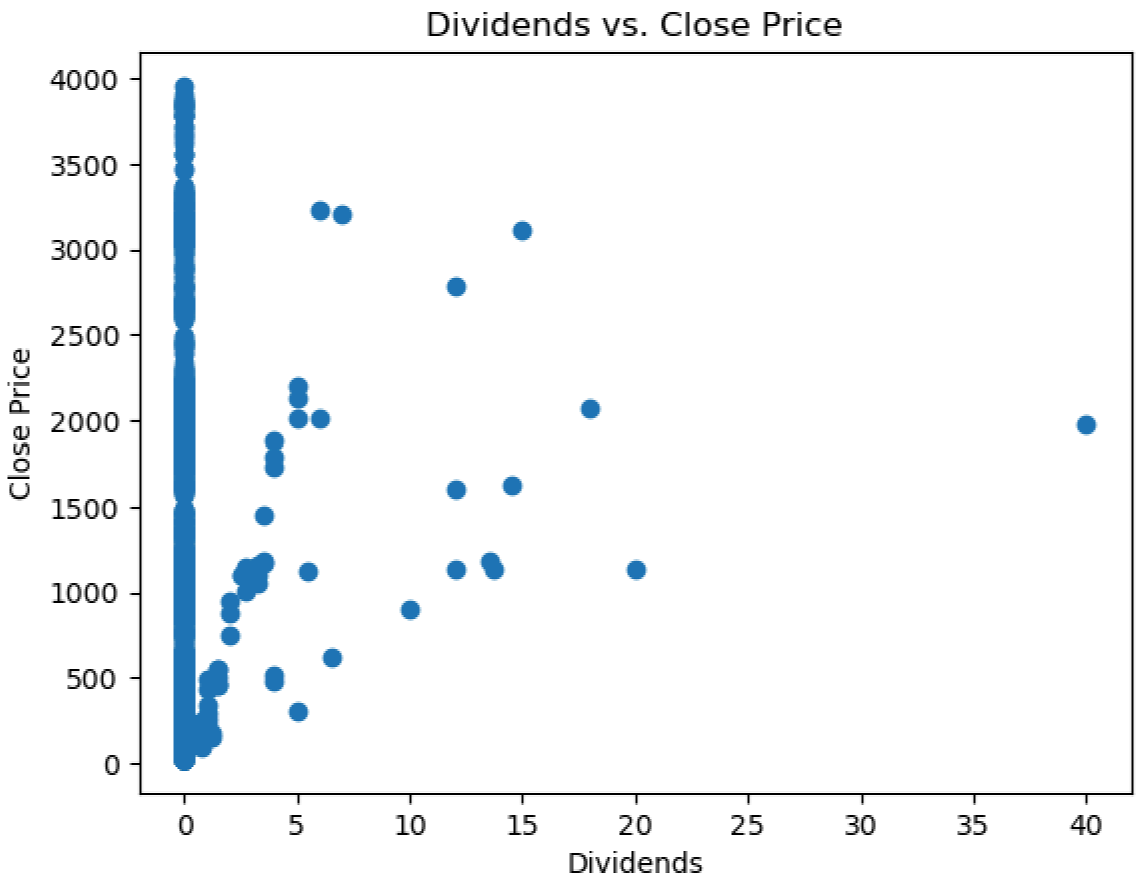


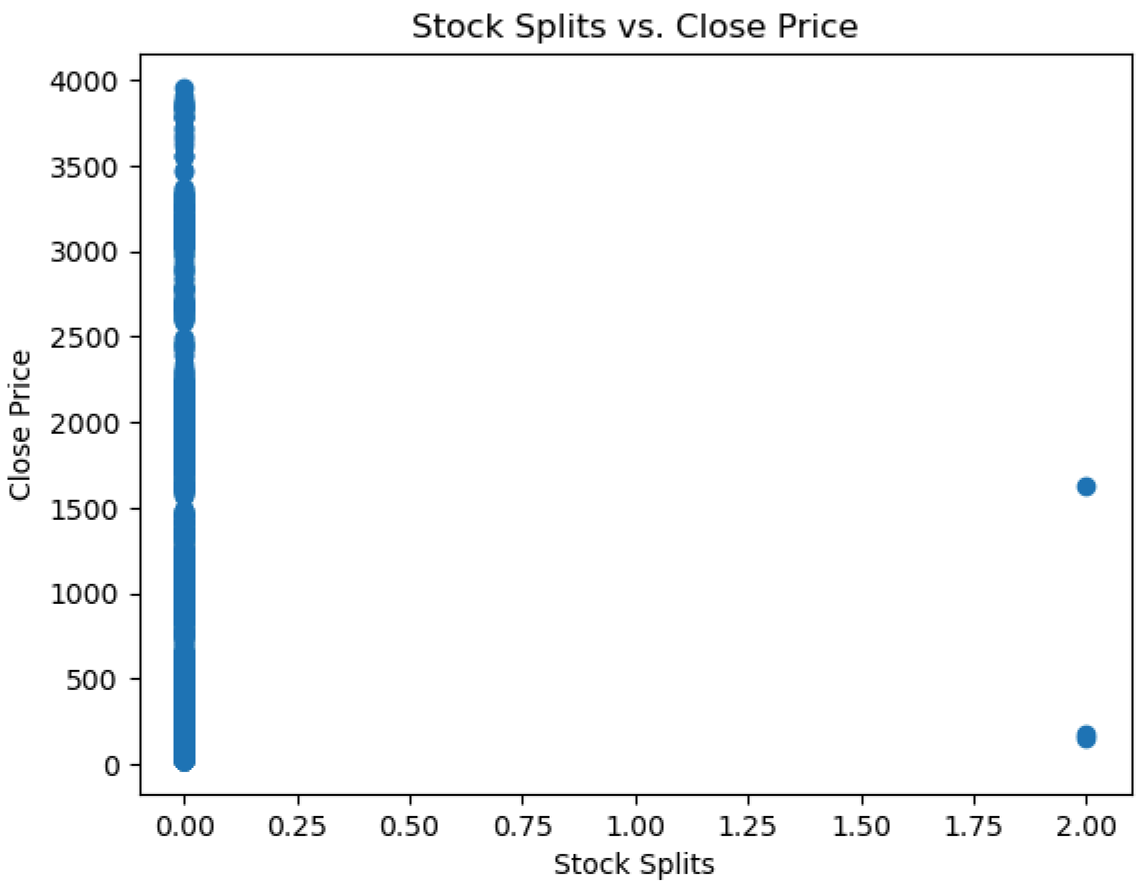
Dividends and Stock Splits

In [13]:

```
# Dividends vs. Close Price
plt.scatter(df['Dividends'], df['Close'])
plt.xlabel('Dividends')
plt.ylabel('Close Price')
plt.title('Dividends vs. Close Price')
plt.show()
```

```python
# Stock Splits vs. Close Price
plt.scatter(df['Stock Splits'], df['Close'])
plt.xlabel('Stock Splits')
plt.ylabel('Close Price')
plt.title('Stock Splits vs. Close Price')
plt.show()
```





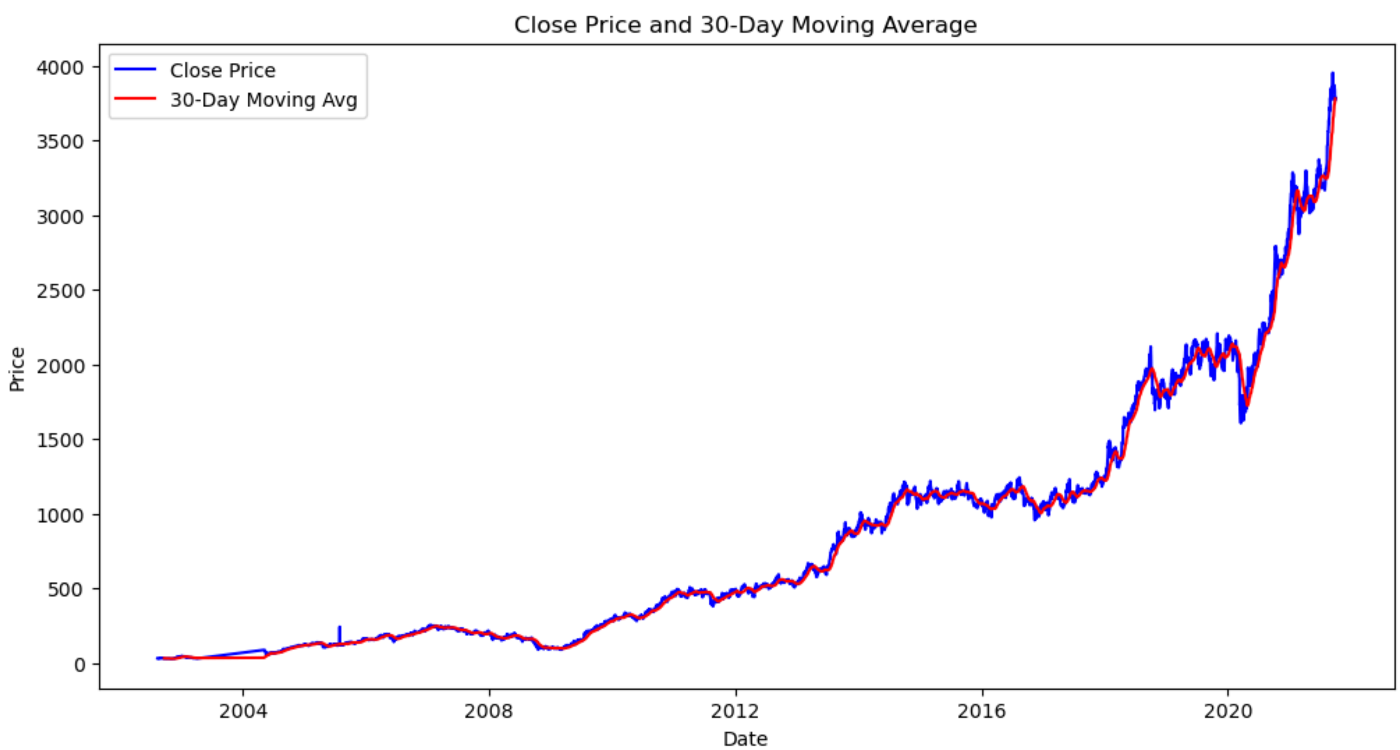
Moving Averages

In [14]:

```
df['30-Day Moving Avg'] = df['Close'].rolling(window=30).mean()


# Plot Close price and moving average
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Close Price',
color='b')
plt.plot(df['Date'], df['30-Day Moving Avg'], label='30-Day
Moving Avg', color='r')
plt.xlabel('Date')
```

```python
plt.ylabel('Price')
plt.title('Close Price and 30-Day Moving Average')
plt.legend()
plt.show()
```



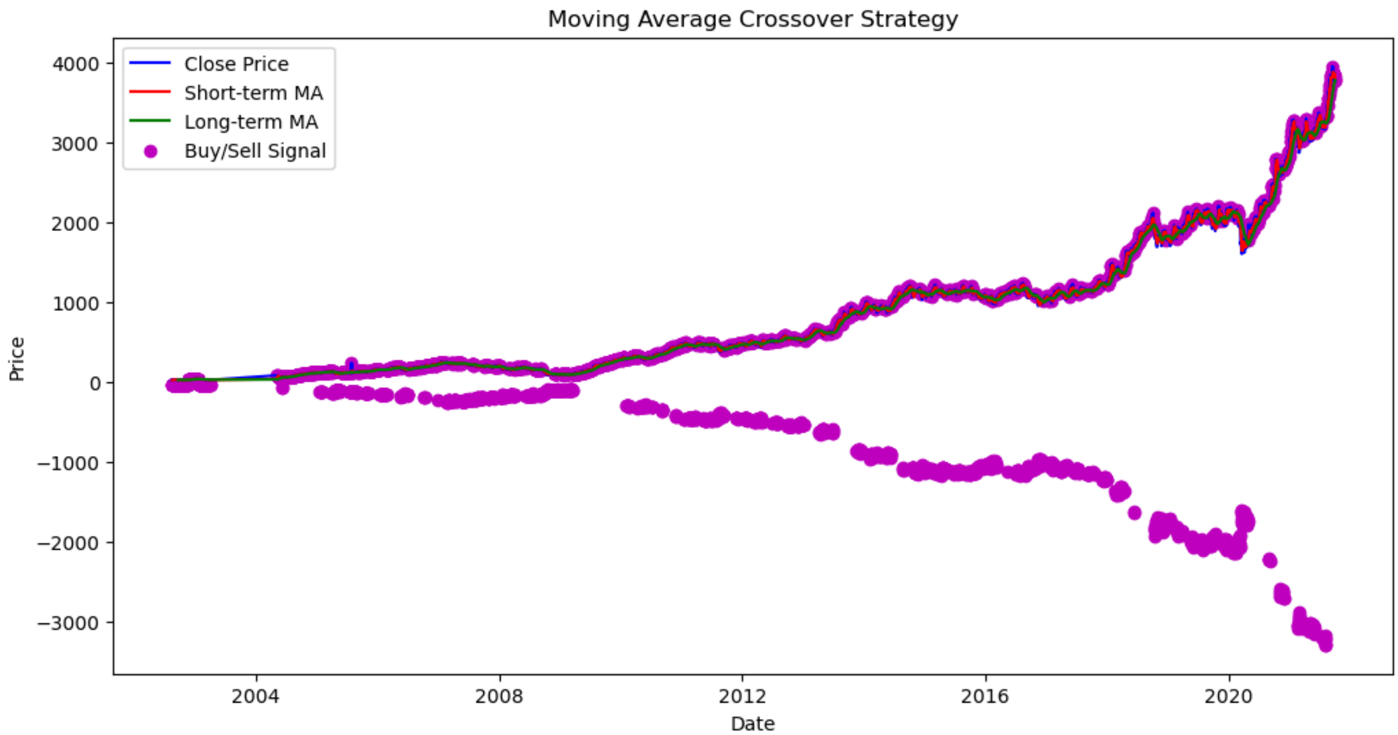
Moving Average Crossover Strategy

In [15]:

```python
df['Short_MA'] = df['Close'].rolling(window=5).mean()
df['Long_MA'] = df['Close'].rolling(window=30).mean()

# Creating a trading signals based on moving average crossovers
df['Signal'] = np.where(df['Short_MA'] > df['Long_MA'], 1, -1)
```

```
# Plot the strategy signals
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Close Price',
color='b')
plt.plot(df['Date'], df['Short_MA'], label='Short-term MA',
color='r')
plt.plot(df['Date'], df['Long_MA'], label='Long-term MA',
color='g')
plt.scatter(df['Date'], df['Close'] * df['Signal'],
label='Buy/Sell Signal', marker='o', color='m')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Moving Average Crossover Strategy')
plt.legend()
plt.show()
```

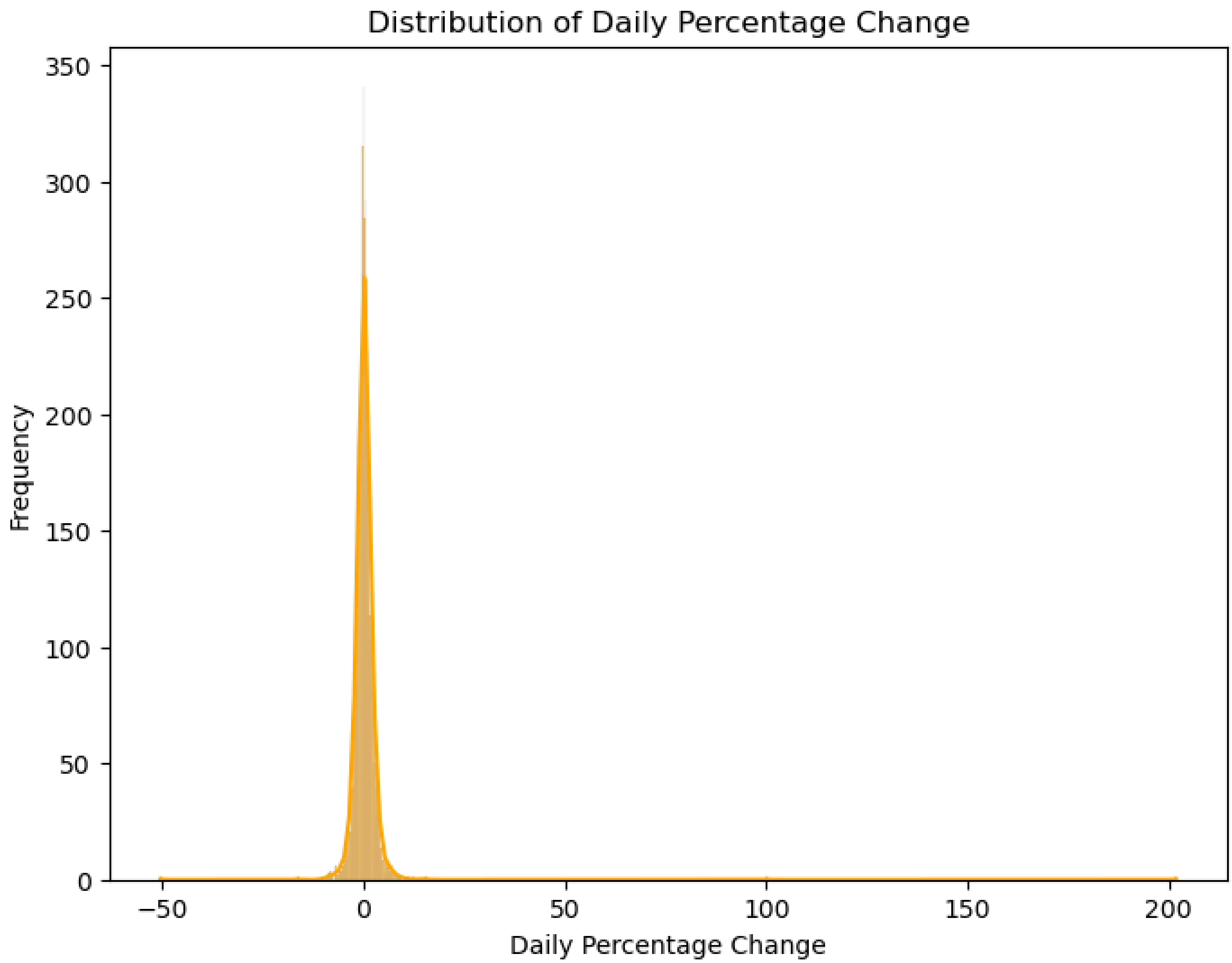


Daily Price Change

In [16]:

```
df['Daily_Price_Change'] = df['Close'].pct_change() * 100


# Distribution of daily percentage change
plt.figure(figsize=(8, 6))
sns.histplot(df['Daily_Price_Change'].dropna(), kde=True,
color='orange')
plt.xlabel('Daily Percentage Change')
plt.ylabel('Frequency')
plt.title('Distribution of Daily Percentage Change')
plt.show()
```



**Feature Engineering

In [17]:

```
df['Moving_Avg_Close'] = df['Close'].rolling(window=7).mean()
```

**Modelling

In [18]:

```
df.shape
```

Out[18]:

```
(4463, 14)
```

Data Preparation & Normalization

In [19]:

```
# Prepare the data for LSTM
X_train = df['Close'].values.reshape(-1, 1)
y_train = df['Close'].shift(-1).dropna().values

# Normalize the data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Define the test data
test_ratio = 0.2
test_size = int(len(df) * test_ratio)
test_data = df[-test_size:]

# Prepare the data for prediction
X_test = test_data['Close'].values.reshape(-1, 1)
X_test_scaled = scaler.transform(X_test)
X_test_lstm = X_test_scaled.reshape(-1, 1, 1)
```

Reshaping Data

In [20]:

```
# Reshape the data for LSTM
X_train_lstm = X_train_scaled[:-1].reshape(-1, 1, 1)
y_train_lstm = X_train_scaled[1:]
```

Building a LSTM Model

In [21]:

```
model = Sequential()
model.add(LSTM(50, input_shape=(1, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Set the number of epochs and batch size
epochs = 30
batch_size = 15

# Train the model with tqdm progress bar
for epoch in tqdm(range(epochs)):
    for i in range(0, len(X_train_lstm), batch_size):
        X_batch = X_train_lstm[i:i+batch_size]
```

```
        y_batch = y_train_lstm[i:i+batch_size]
        model.train_on_batch(X_batch, y_batch)

# Prepare the data for prediction
X_test = test_data['Close'].values.reshape(-1, 1)
X_test_scaled = scaler.transform(X_test)
X_test_lstm = X_test_scaled.reshape(-1, 1, 1)
```

```
100%|██████████| 30/30 [01:13<00:00,  2.46s/it]
```

Predictions using LSTM

In [22]:

```
lstm_predictions = model.predict(X_test_lstm).flatten()
```

```
28/28 [==============================] - 1s 3ms/step
```

Inverse transform of the predictions

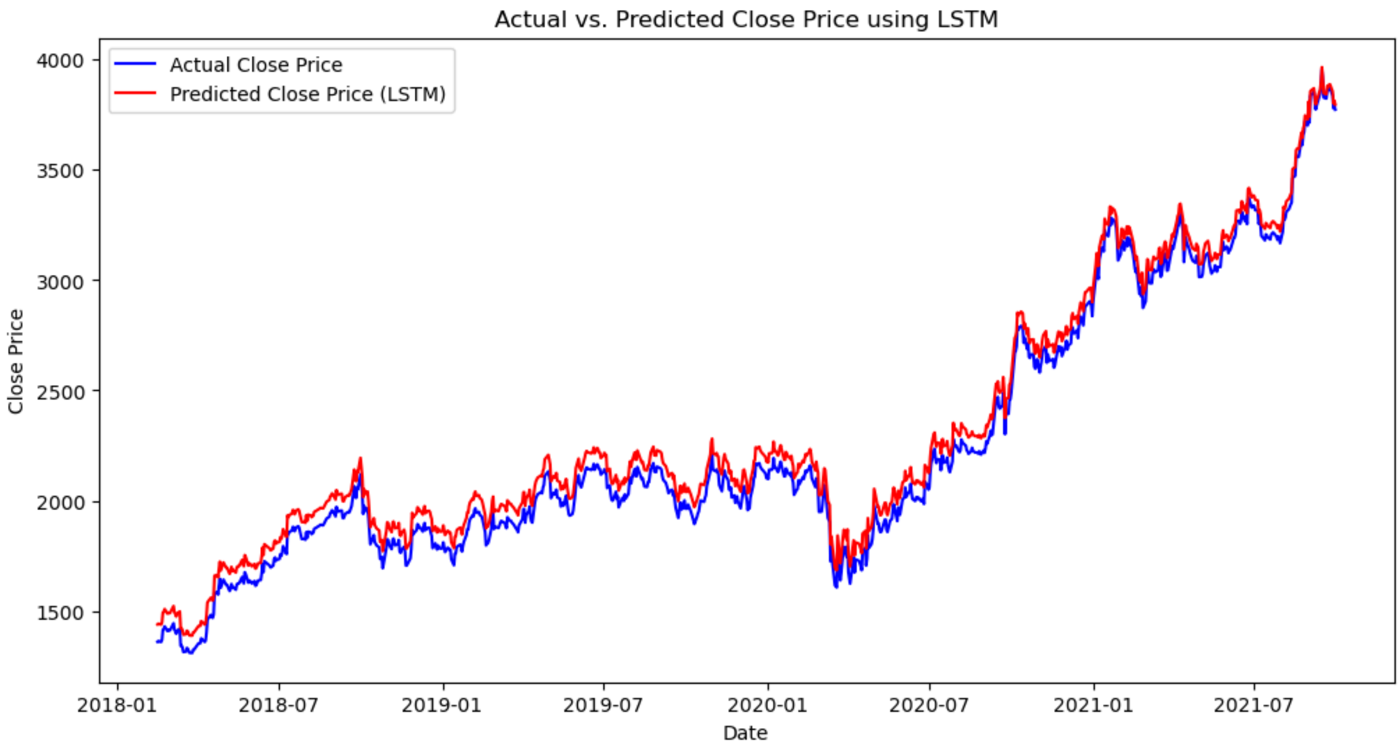
In [23]:

```
lstm_predictions = lstm_predictions.reshape(-1, 1)
lstm_predictions = scaler.inverse_transform(lstm_predictions)
```

Visualization of LSTM predictions

In [24]:

```
plt.figure(figsize=(12, 6))
plt.plot(test_data['Date'], test_data['Close'], label='Actual
Close Price', color='b')
plt.plot(test_data['Date'], lstm_predictions, label='Predicted
Close Price (LSTM)', color='r')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Actual vs. Predicted Close Price using LSTM')
plt.legend()
plt.show()
```



Mean Absolute Error

In [25]:

```
lstm_mae = mean_absolute_error(test_data['Close'],
lstm_predictions)
print("LSTM Mean Absolute Error:", lstm_mae)
```

```
LSTM Mean Absolute Error: 69.52758706952424
```

In [26]:

```
lstm_predictions = lstm_predictions.reshape(-1, 1)
lstm_predictions = scaler.inverse_transform(lstm_predictions)
```

```python
date_index = test_data.index[-len(lstm_predictions):]
predictions_df = pd.DataFrame({'Date': date_index,
'Predicted_Close': lstm_predictions.flatten()})



predictions_df.to_csv('predictions.csv', index=False)
```

[Reference link]