# <u>TCS Stock Data - Live and Latest</u>

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

## DATASET INFORMATION

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.

## 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.

## Data loading

Load the TCS stock history dataset into a Pandas DataFrame.

```
import pandas as pd

df = pd.read_csv('TCS_stock_history.csv')
display(df.head())
```

| | 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** | 2002-08-14 | 29.184536 | 29.184536 | 26.563503 | 27.111877 | 822776 | 0.0 | 0.0 |
| **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 |

## Data exploration

Explore the dataset by checking its shape, data types, descriptive statistics, missing values, and the distribution of the 'Close' price.

```
# Check the shape of the DataFrame
print("Shape of the DataFrame:", df.shape)

# Display the data types of each column
print("\nData Types of Columns:")
df.info()
```

```
Shape of the DataFrame: (4463, 8)

Data Types of Columns:
<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
```

```python
# Calculate and display descriptive statistics for numerical columns
print("\nDescriptive Statistics for Numerical Columns:")
display(df.describe())
```

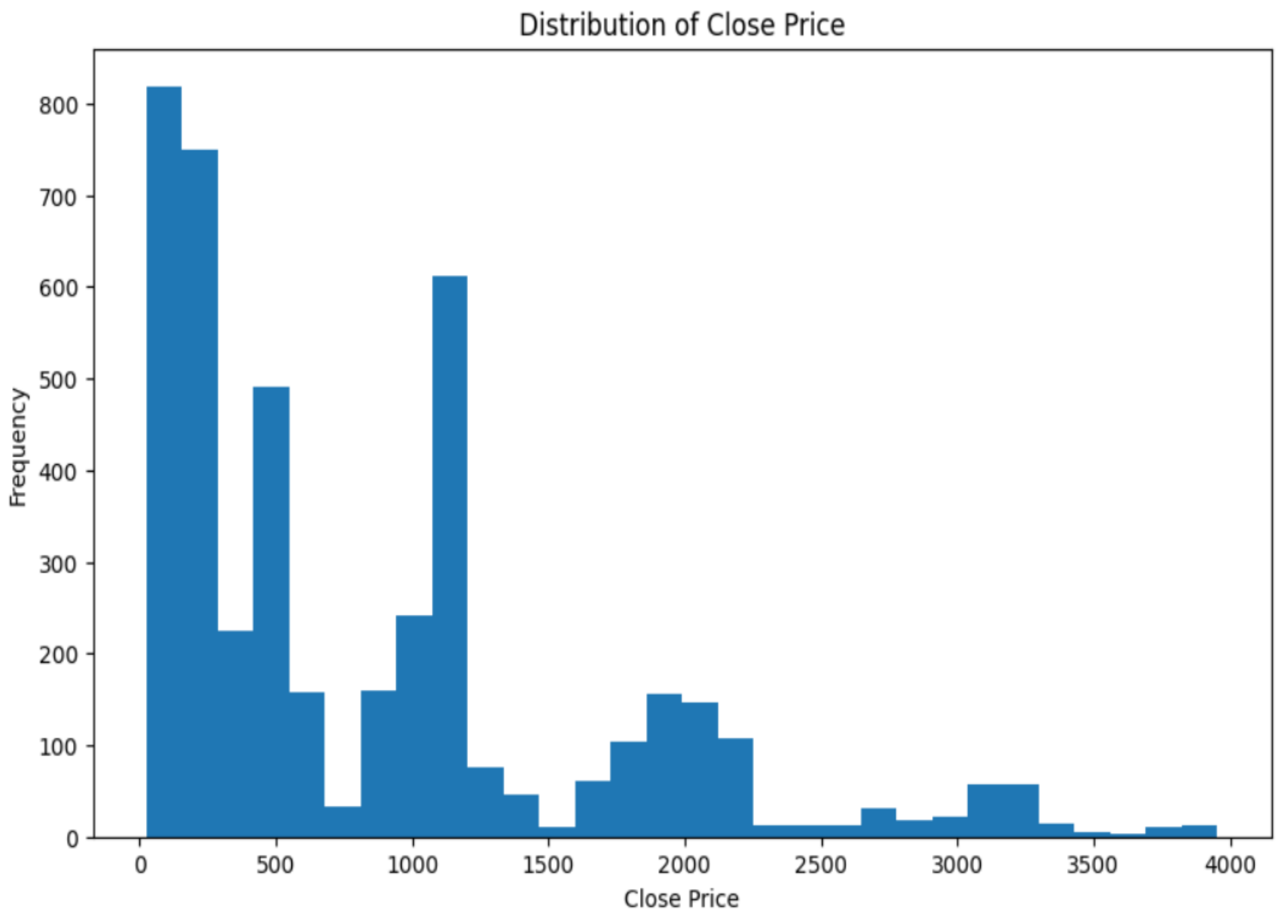
```
Descriptive Statistics for Numerical Columns:
```

| | 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 |

```python
# Identify any missing values in the dataset
print("\nMissing Values in Each Column:")
print(df.isnull().sum())
```

```
Missing Values in Each Column:
Date            0
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```

```
# Examine the distribution of the 'Close' price using a histogram
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.hist(df['Close'], bins=30)
plt.xlabel('Close Price')
plt.ylabel('Frequency')
plt.title('Distribution of Close Price')
plt.show()
```



## Data cleaning

Convert the 'Date' column to datetime objects and check for missing values in the 'Close' column, then handle potential outliers in 'Close' using the IQR method.

```
# Convert 'Date' column to datetime objects
df['Date'] = pd.to_datetime(df['Date'])

# Check for missing values in the 'Close' column
print("Missing values in 'Close':", df['Close'].isnull().sum())

# Handle potential outliers in 'Close' using IQR
Q1 = df['Close'].quantile(0.25)
Q3 = df['Close'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
# Identify outliers
outliers = df[(df['Close'] < lower_bound) | (df['Close'] > upper_bound)]

# Replace outliers with the median 'Close' price
df['Close'] = df['Close'].apply(lambda x: df['Close'].median() if (x < lower_bound or x > upper_bound) else x)
```

```
Missing values in 'Close': 0
```

## Data wrangling

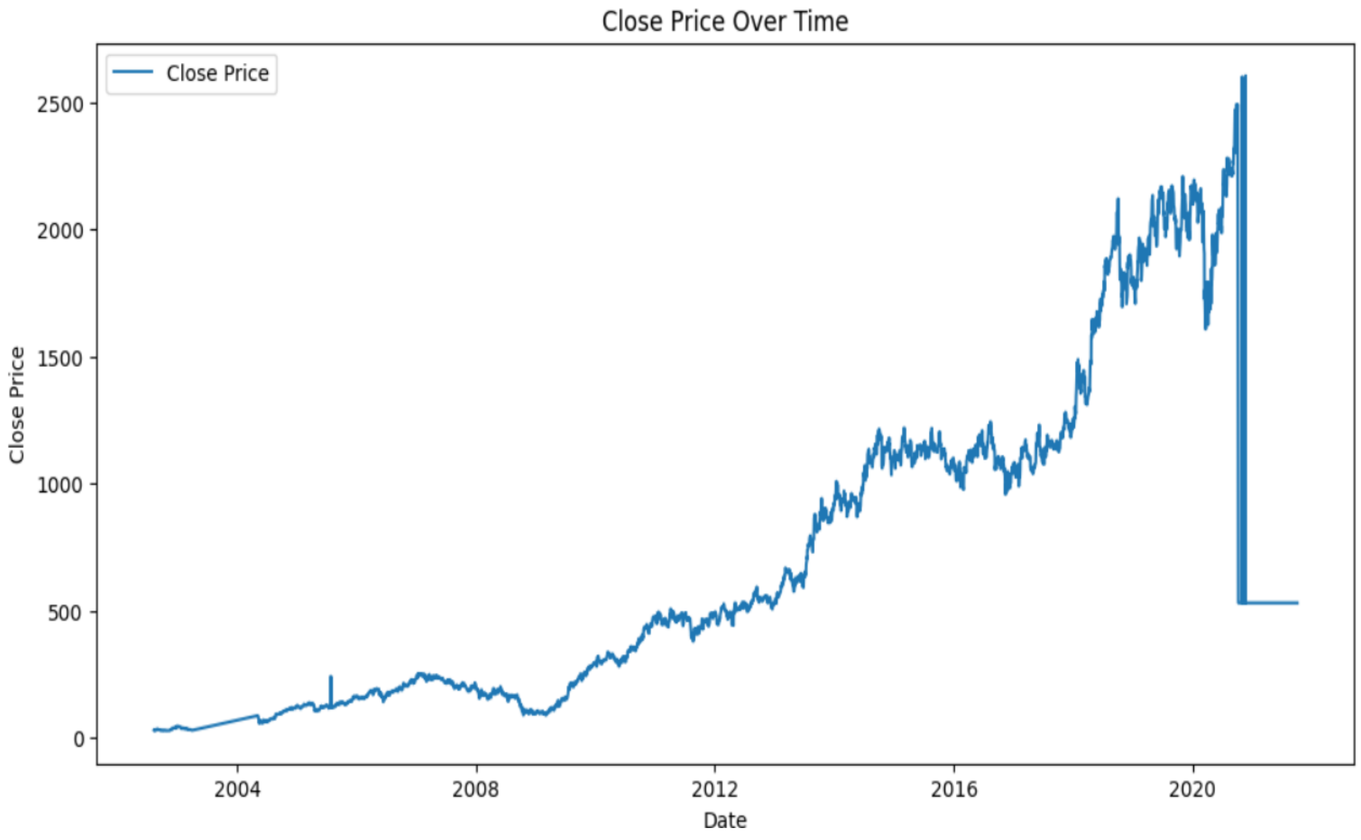
Create new features by calculating daily price change, 50-day and 200-day moving averages, and sort the dataframe by date in ascending order.

```
# Calculate the daily price change
df['Daily_Price_Change'] = df['Close'].diff()

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

# Sort the DataFrame by Date in ascending order
df.sort_values('Date', inplace=True)
```

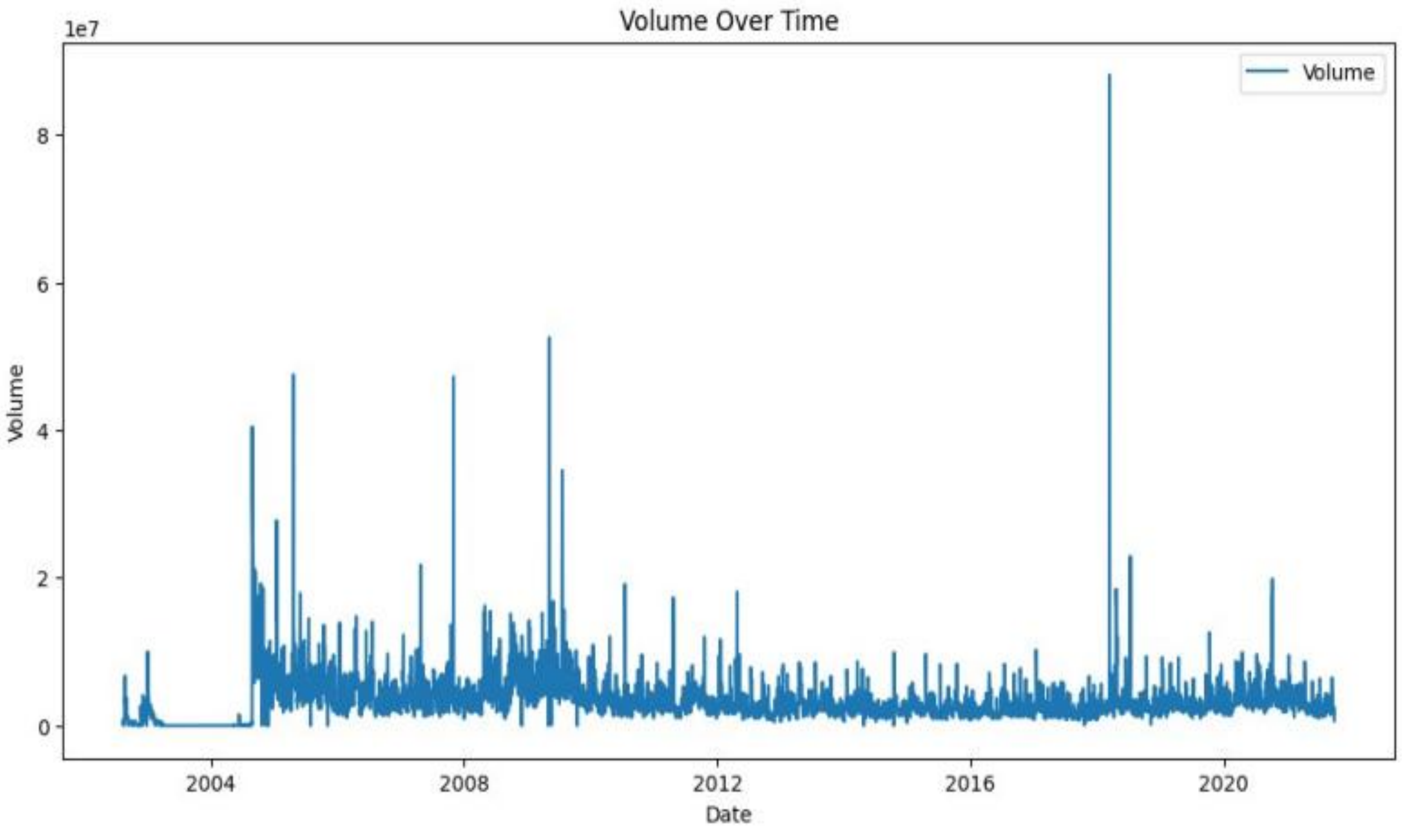
## Data visualization

Visualize the time series data for the stock prices and related features. Create the line plots for Close Price, Volume, Dividends, and Stock Splits over time.

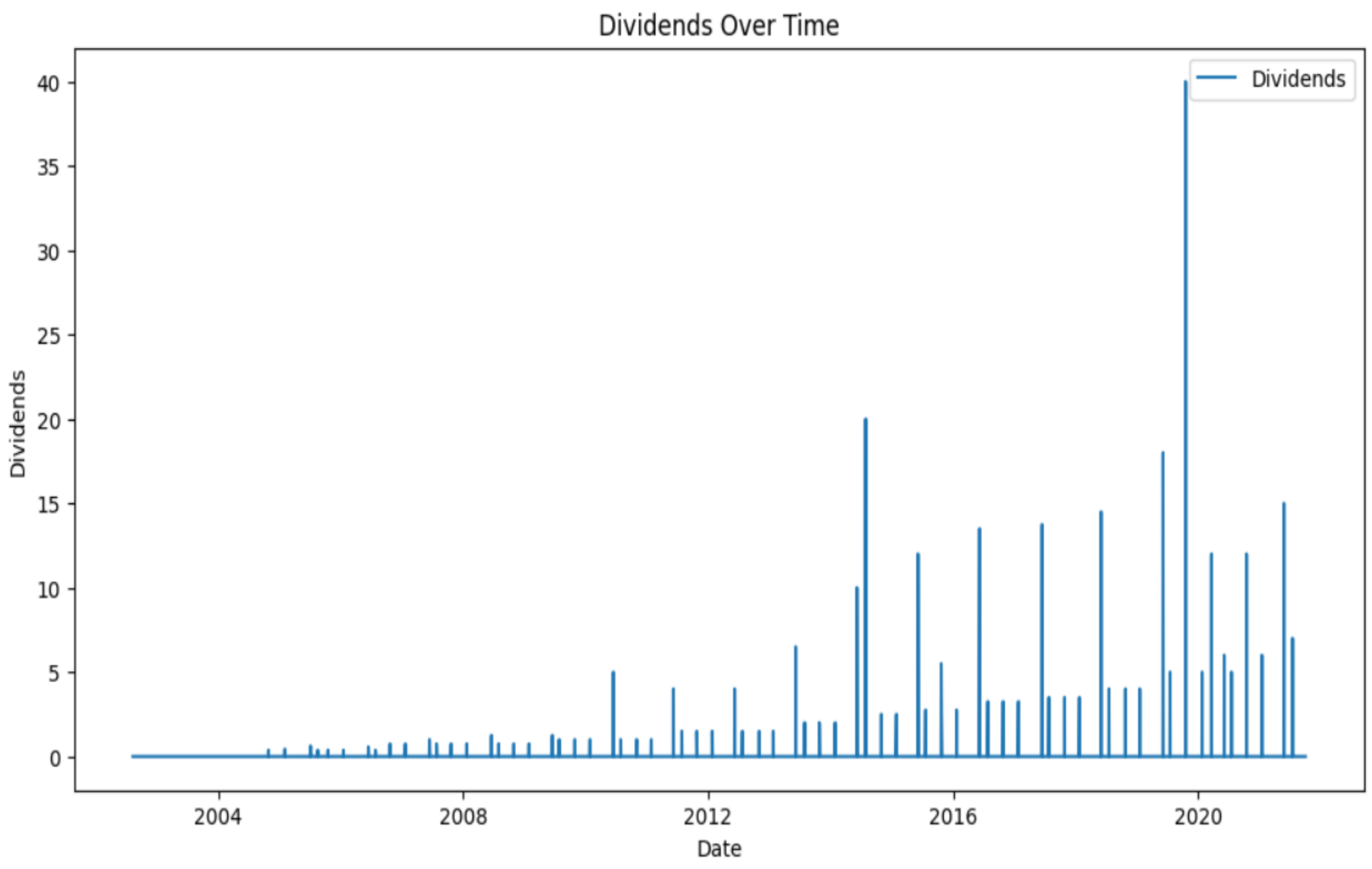
```
import matplotlib.pyplot as plt

# Close Price Over Time
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Close'], label='Close Price')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Close Price Over Time')
plt.legend()
plt.show()
```



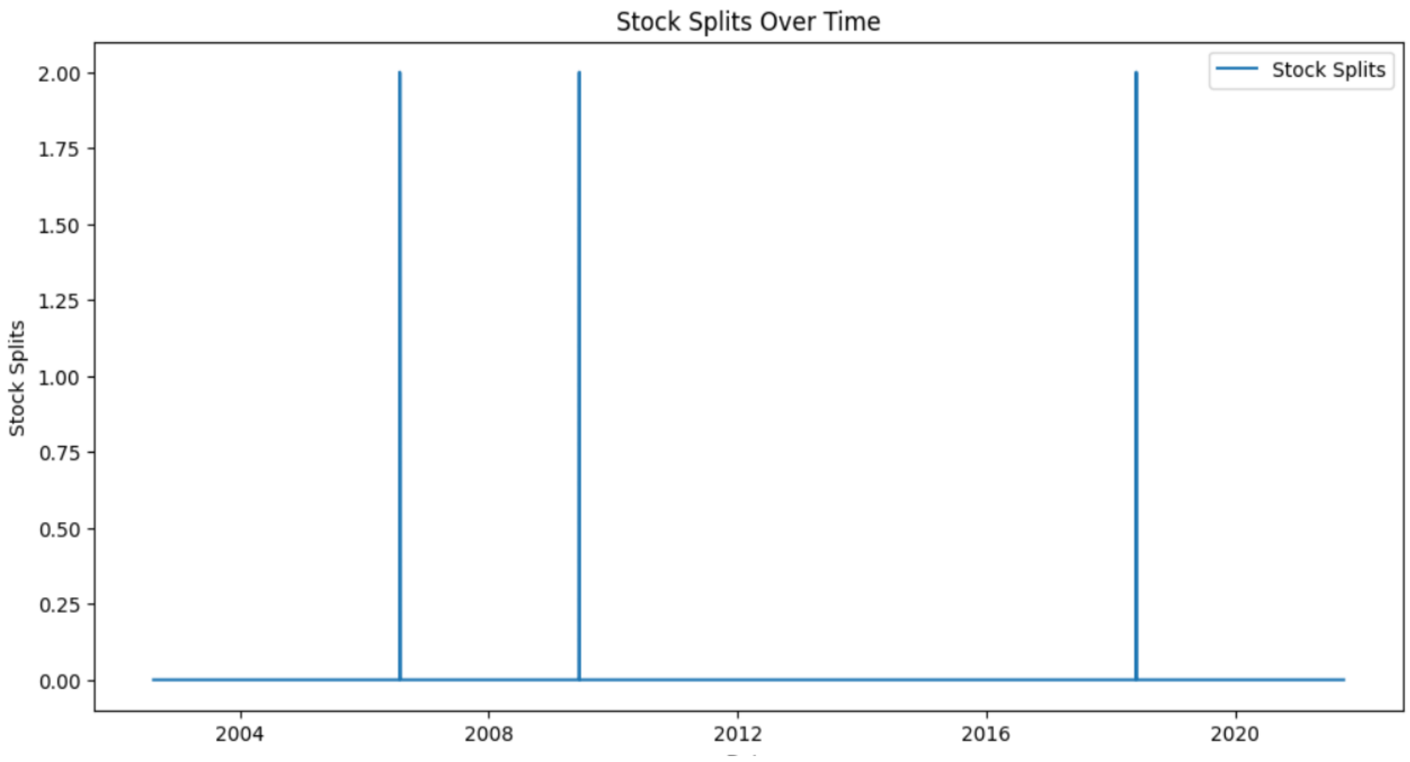
```python
# Volume Over Time
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Volume'], label='Volume')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.title('Volume Over Time')
plt.legend()
plt.show()
```



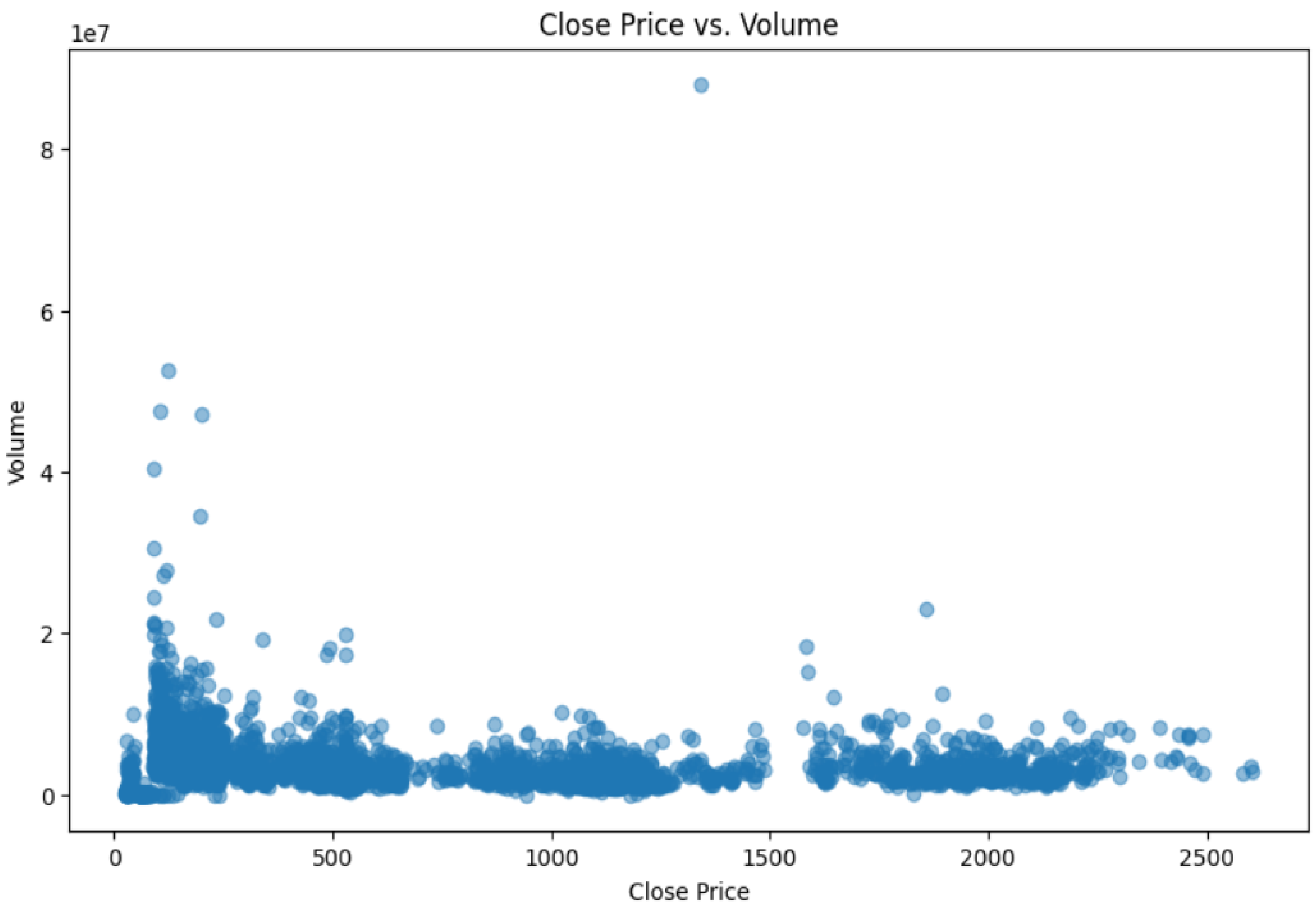
```python
# Dividends Over Time
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Dividends'], label='Dividends')
plt.xlabel('Date')
plt.ylabel('Dividends')
plt.title('Dividends Over Time')
plt.legend()
plt.show()
```



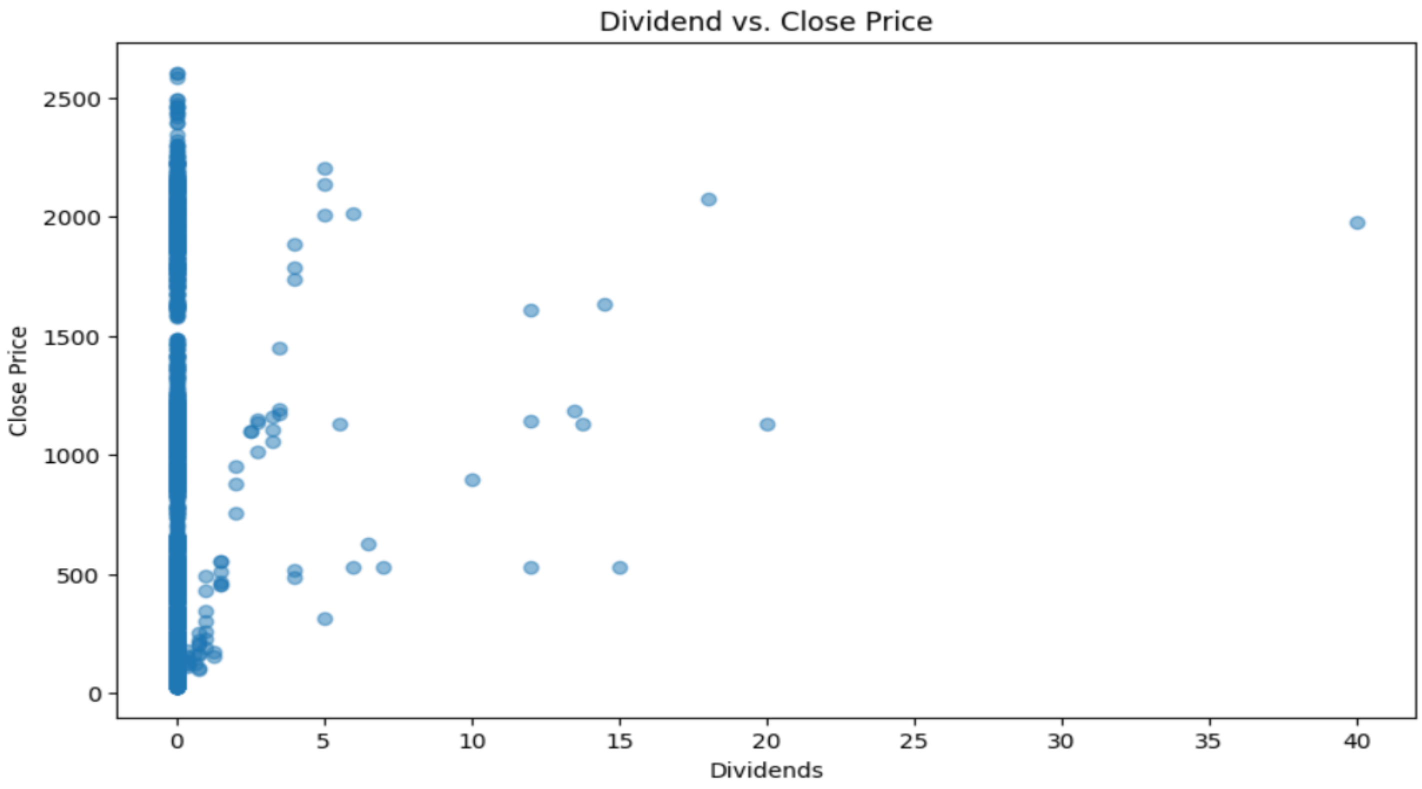
```python
# Stock Splits Over Time
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Stock Splits'], label='Stock Splits')
plt.xlabel('Date')
plt.ylabel('Stock Splits')
plt.title('Stock Splits Over Time')
plt.legend()
plt.show()
```



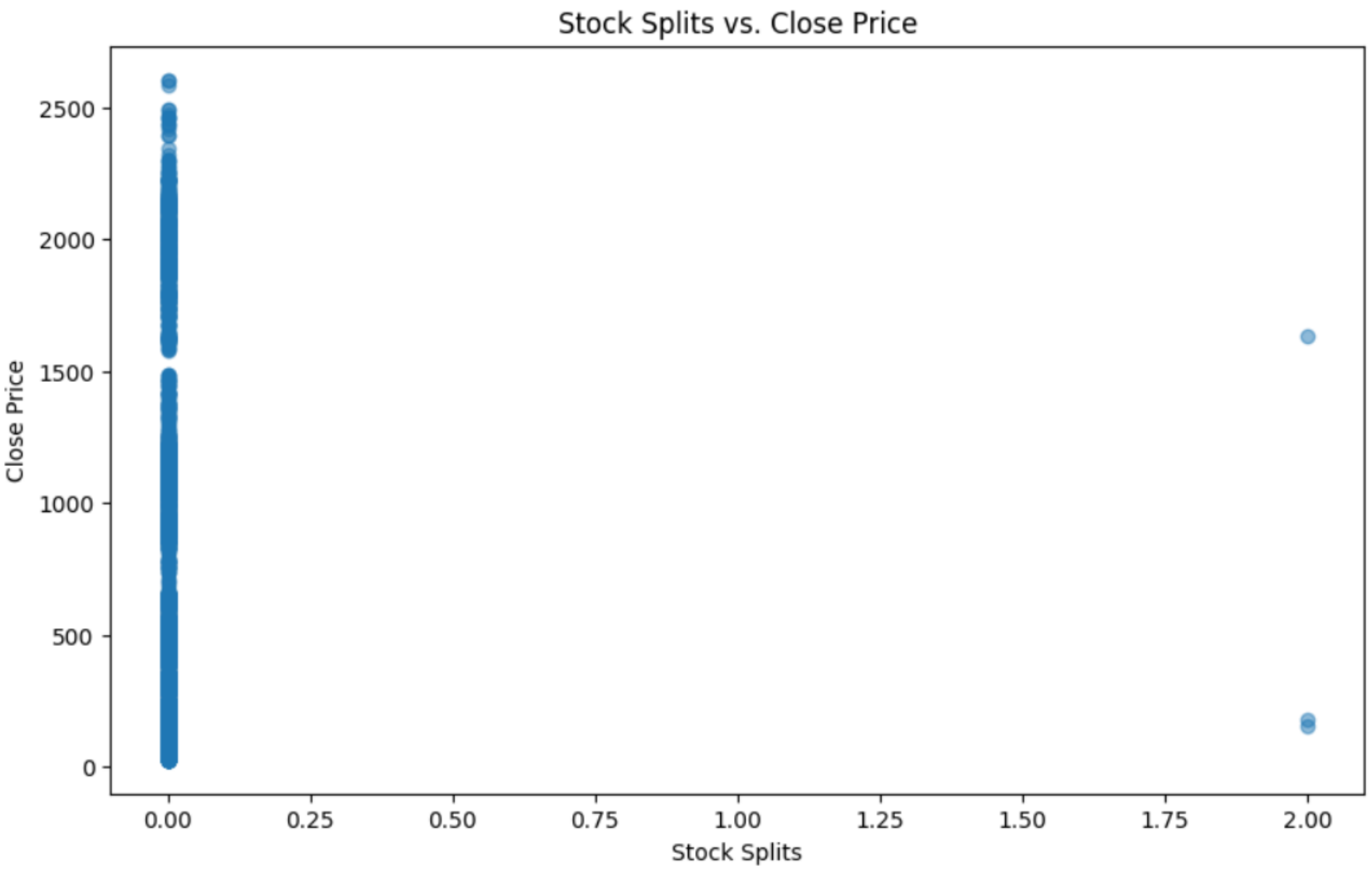
```python
import matplotlib.pyplot as plt

# Close Price vs. Volume
plt.figure(figsize=(10, 6))
plt.scatter(df['Close'], df['Volume'], alpha=0.5)
plt.xlabel('Close Price')
plt.ylabel('Volume')
plt.title('Close Price vs. Volume')
plt.show()
```



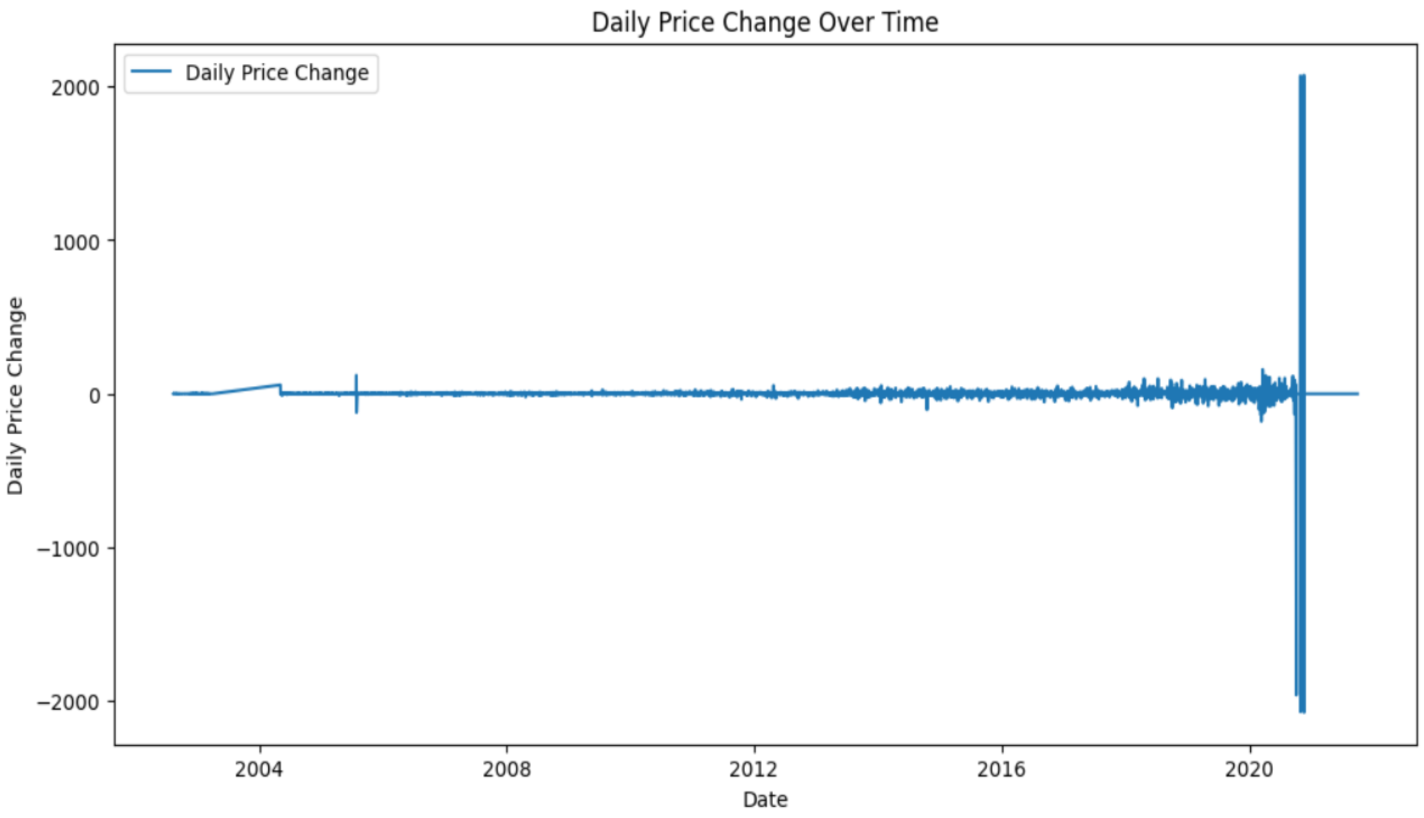
```python
# Dividend vs. Close Price
plt.figure(figsize=(10, 6))
plt.scatter(df['Dividends'], df['Close'], alpha=0.5)
plt.xlabel('Dividends')
plt.ylabel('Close Price')
plt.title('Dividend vs. Close Price')
plt.show()
```



```python
# Stock Splits vs. Close Price
plt.figure(figsize=(10, 6))
plt.scatter(df['Stock Splits'], df['Close'], alpha=0.5)
plt.xlabel('Stock Splits')
plt.ylabel('Close Price')
plt.title('Stock Splits vs. Close Price')
plt.show()
```



```python
import matplotlib.pyplot as plt

# Daily Price Change Over Time
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Daily_Price_Change'], label='Daily Price Change')
plt.xlabel('Date')
plt.ylabel('Daily Price Change')
plt.title('Daily Price Change Over Time')
plt.legend()
plt.show()
```



## Feature engineering

Create lagged features for the 'Close' price and handle missing values.

```python
# Create lagged features for 'Close' price
df['Close_Lag1'] = df['Close'].shift(1)
df['Close_Lag5'] = df['Close'].shift(5)
df['Close_Lag10'] = df['Close'].shift(10)

# Calculate the ratio of MA50 to MA200
df['MA_Ratio'] = df['MA50'] / df['MA200']

# Calculate the daily volume change
df['Volume_Change'] = df['Volume'].diff()

# Add the day of the week as a feature
df['DayOfWeek'] = df['Date'].dt.dayofweek

# Handle missing values by dropping rows with NaN values
df.dropna(inplace=True)
```

## Data splitting

Split the dataset into training and testing sets, ensuring the testing set comes after the training set chronologically.

```python
from sklearn.model_selection import train_test_split

# Split the data into features (X) and target (y)
X = df[['Close_Lag1', 'Close_Lag5', 'Close_Lag10', 'MA_Ratio', 'Volume_Change', 'DayOfWeek']]
y = df['Close']

# Split the data into training and testing sets chronologically
train_size = int(len(df) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Print the shapes of the training and testing sets to confirm the split
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (3411, 6)
y_train shape: (3411,)
X_test shape: (853, 6)
y_test shape: (853,)
```

## Model training

Train a Linear Regression model using the training data (X_train, y_train).

```python
from sklearn.linear_model import LinearRegression

# Instantiate a Linear Regression model
model = LinearRegression()

# Train the model using the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)
```

## Model evaluation

Evaluate the performance of the trained Linear Regression model using the test data.

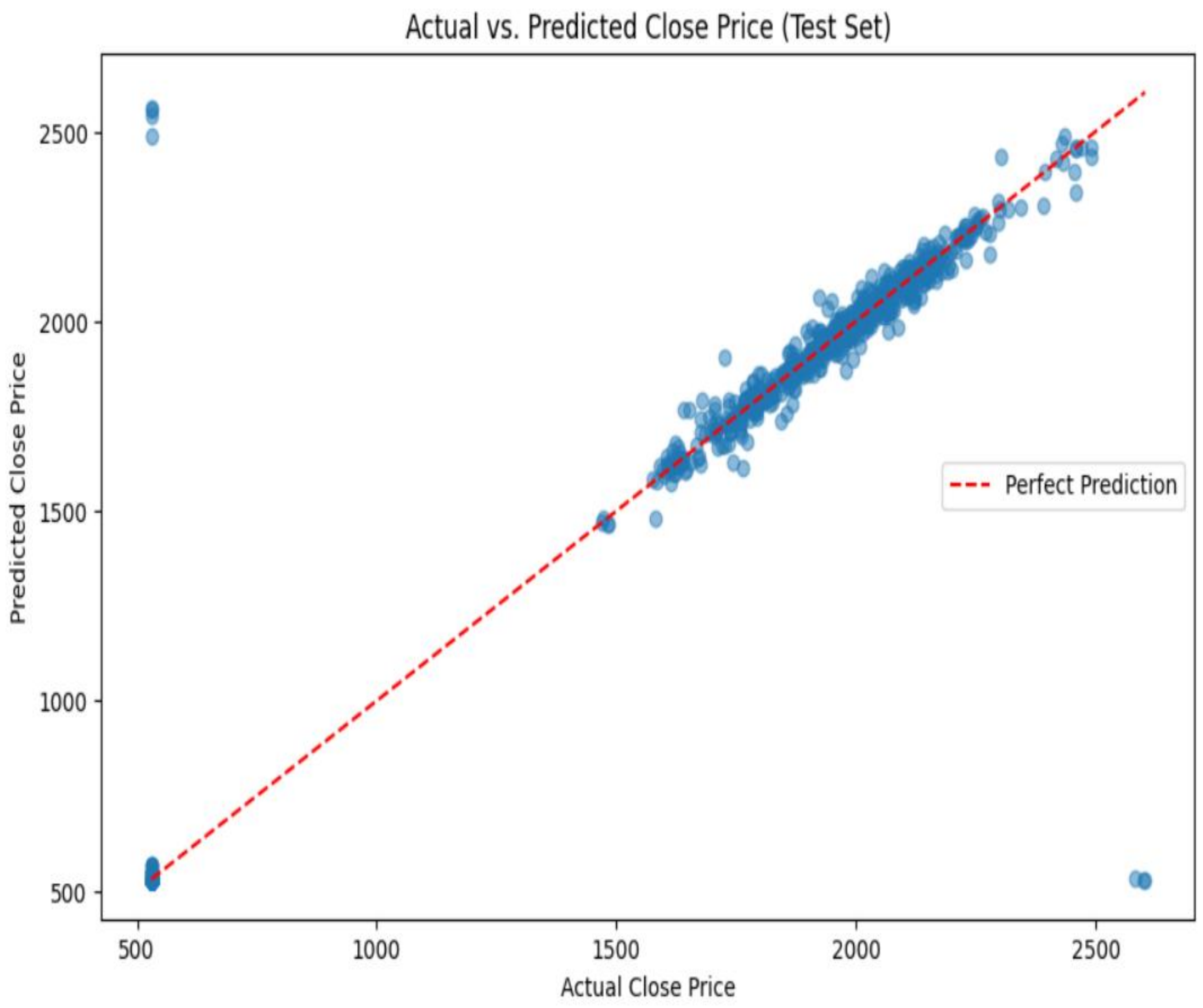
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Calculate MSE
mse = mean_squared_error(y_test, y_pred)

# Calculate RMSE by taking the square root of MSE
rmse = np.sqrt(mse)

# Calculate MAE
mae = mean_absolute_error(y_test, y_pred)

# Calculate R2 score
r2 = r2_score(y_test, y_pred)

# Print the evaluation metrics
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2) Score:", r2)
```

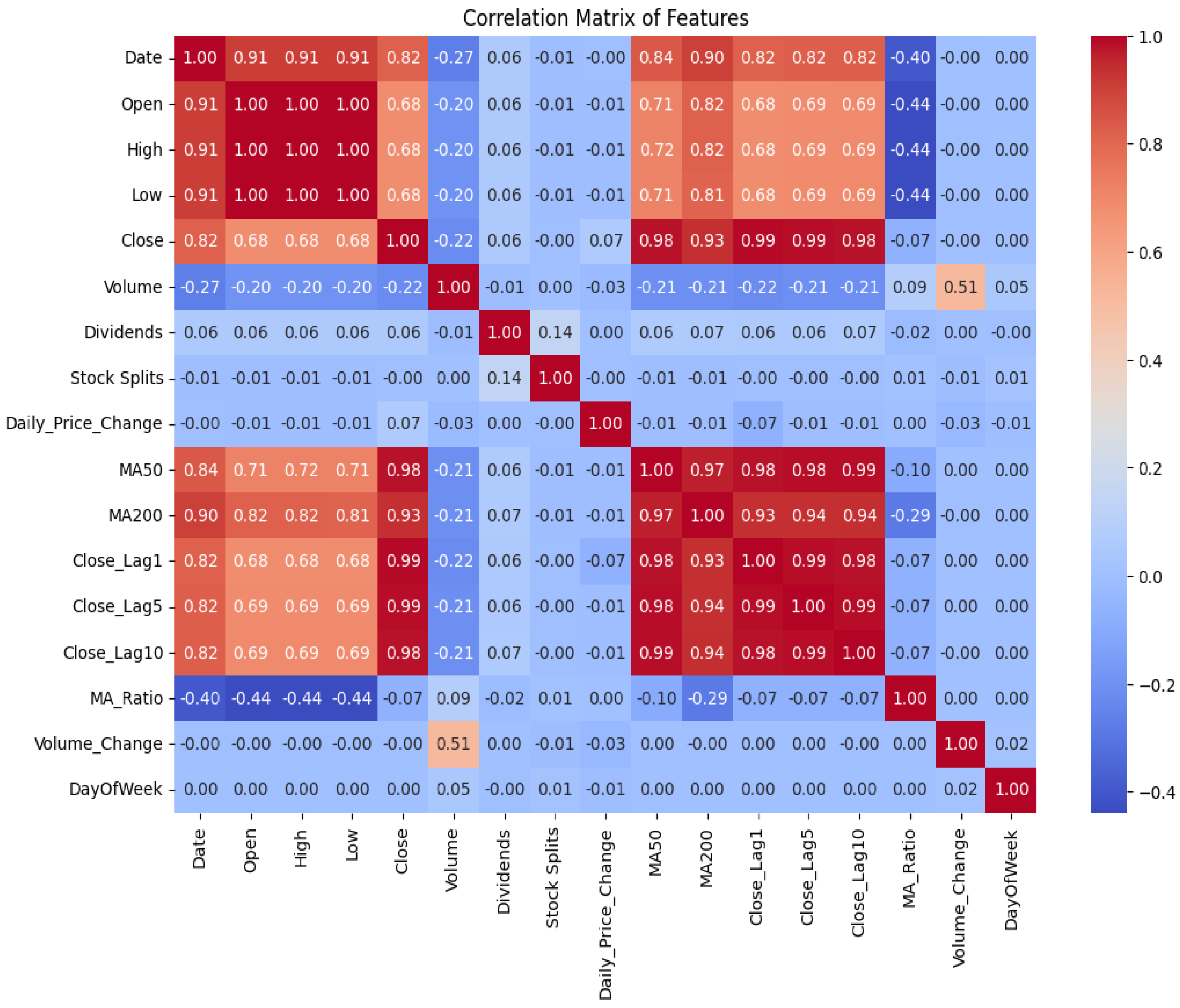
```
Root Mean Squared Error (RMSE): 186.49141862659096
Mean Absolute Error (MAE): 34.79809656344462
R-squared (R2) Score: 0.9220184790701764
```

## Data visualization

Visualize the predicted vs. actual 'Close' price for the test set and generate a heatmap of feature correlations.

```
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Predicted vs. Actual 'Close' Price Plot
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual Close Price")
plt.ylabel("Predicted Close Price")
plt.title("Actual vs. Predicted Close Price (Test Set)")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Perfect Prediction')
plt.legend()
plt.show()


# 2. Heatmap of Feature Correlations
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix of Features")
plt.show()
```





## Conclusion

The TCS Stock Prediction project aimed to analyze historical stock price data and develop a predictive model to estimate future stock prices. Through a structured data science workflow—including data loading, cleaning, exploration, feature engineering, visualization, model training, and evaluation—valuable insights were derived regarding stock price trends and influential factors. The dataset contained historical TCS stock price data, including attributes such as opening price, closing price, high and low values, trading volume, dividends, and stock splits. Initial data exploration revealed no missing values, ensuring a robust foundation for analysis. Furthermore, feature engineering introduced lagged values, moving averages, and day-of-week attributes to enhance the model's predictive ability. A linear regression model was trained using an 80-20 chronological data split to prevent data leakage. The model's performance was evaluated using key metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared ($R^2$). The results demonstrated strong predictive capabilities, with an $R^2$ score of **0.922**, indicating that the model explained **92.2% of the variance** in the closing price. Additionally, the RMSE value of **186.49** and MAE of **34.80** suggested relatively low prediction errors, reinforcing the model's reliability. Visualizations, including time-series plots, scatter plots, and correlation heatmaps, provided deeper insights into stock price movements. The correlation matrix highlighted relationships among features, helping refine the model and identify potential improvements. The introduction of lagged features and moving average ratios proved beneficial in capturing stock price trends.