

1. Introduction

This project aims to explore TCS's stock behavior using such modern analytical techniques. By performing thorough exploratory data analysis and applying predictive models, we seek to understand past trends and build tools capable of forecasting future closing prices—thereby contributing to informed decision-making in financial markets.

2. Import Libraries

The implementation of this project relies on a combination of data analysis libraries, machine learning frameworks, and development tools. These technologies enabled data preprocessing, visualization, model building, and evaluation.

```
In [1]: 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
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
```

3. Dataset Overview

TCS_stock_history.csv – Main dataset used for modeling and analysis.

TCS_stock_action.csv – Contains market actions such as corporate splits and dividends.

TCS_stock_info.csv – Provides metadata or reference information (used for context only).

```
In [3]: df_action = pd.read_csv("https://raw.githubusercontent.com/Sunil0012/TCS_Stock_Anal
df_history = pd.read_csv("https://raw.githubusercontent.com/Sunil0012/TCS_Stock_Anal
df_info = pd.read_csv("https://raw.githubusercontent.com/Sunil0012/TCS_Stock_Anals
```

```
In [4]: df_action
```

Out[4]:

	Date	Dividends	Stock Splits
0	2004-10-28	0.3750	0.0
1	2005-02-03	0.4375	0.0
2	2005-07-06	0.6250	0.0
3	2005-08-18	0.3750	0.0
4	2005-10-18	0.3750	0.0
...
65	2020-07-16	5.0000	0.0
66	2020-10-14	12.0000	0.0
67	2021-01-14	6.0000	0.0
68	2021-05-25	15.0000	0.0
69	2021-07-15	7.0000	0.0

70 rows × 3 columns

In [5]:

df_history

Out[5]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Split
0	2002-08-12	28.794172	29.742206	28.794172	29.519140	212976	0.0	0
1	2002-08-13	29.556316	30.030333	28.905705	29.119476	153576	0.0	0
2	2002-08-14	29.184536	29.184536	26.563503	27.111877	822776	0.0	0
3	2002-08-15	27.111877	27.111877	27.111877	27.111877	0	0.0	0
4	2002-08-16	26.972458	28.255089	26.582090	27.046812	811856	0.0	0
...
4458	2021-09-24	3890.000000	3944.399902	3855.000000	3871.300049	2320754	0.0	0
4459	2021-09-27	3900.000000	3904.000000	3802.899902	3836.949951	1673362	0.0	0
4460	2021-09-28	3850.000000	3850.000000	3751.250000	3779.149902	2253075	0.0	0
4461	2021-09-29	3759.800049	3806.000000	3722.149902	3791.899902	2489161	0.0	0
4462	2021-09-30	3805.000000	3805.000000	3765.000000	3773.199951	640479	0.0	0

4463 rows × 8 columns



In [6]: df_info

Out[6]:

	zip	400001
0	sector	Technology
1	fullTimeEmployees	509058
2	longBusinessSummary	Tata Consultancy Services Limited provides inf...
3	city	Mumbai
4	phone	91 22 6778 9999
...
145	dividendYield	0.0105
146	bidSize	0
147	dayHigh	3805
148	regularMarketPrice	3773.2
149	logo_url	https://logo.clearbit.com/tcs.com

150 rows × 2 columns

In [7]:

```
df_history['Date'] = pd.to_datetime(df_action['Date'])
df_history['Date'].info()

<class 'pandas.core.series.Series'>
RangeIndex: 4463 entries, 0 to 4462
Series name: Date
Non-Null Count Dtype
-----
70 non-null    datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 35.0 KB
```

In [8]:

```
df_history.describe()
```

Out[8]:

	Date	Open	High	Low	Close	Volum
count	70	4463.000000	4463.000000	4463.000000	4463.000000	4.463000e+0
mean	2013-05-10 13:42:51.428571392	866.936239	876.675013	856.653850	866.537398	3.537876e+0
min	2004-10-28 00:00:00	24.146938	27.102587	24.146938	26.377609	0.000000e+0
25%	2009-03-03 18:00:00	188.951782	191.571816	185.979417	188.594620	1.860959e+0
50%	2013-07-02 12:00:00	530.907530	534.751639	525.616849	529.713257	2.757742e+0
75%	2017-07-13 18:00:00	1156.462421	1165.815854	1143.622800	1154.784851	4.278625e+0
max	2021-07-15 00:00:00	3930.000000	3981.750000	3892.100098	3954.550049	8.806715e+0
std	Nan	829.905368	838.267104	821.233477	829.611313	3.273531e+0



In [9]:

```
corel = df_history.corr()
corel
```

Out[9]:

	Date	Open	High	Low	Close	Volume	Dividends
Date	1.000000	-0.127459	-0.127566	-0.127315	-0.127472	-0.107718	-0.009353
Open	-0.127459	1.000000	0.999888	0.999892	0.999787	-0.153362	0.059743
High	-0.127566	0.999888	1.000000	0.999867	0.999914	-0.150918	0.060044
Low	-0.127315	0.999892	0.999867	1.000000	0.999901	-0.154962	0.059916
Close	-0.127472	0.999787	0.999914	0.999901	1.000000	-0.152844	0.060179
Volume	-0.107718	-0.153362	-0.150918	-0.154962	-0.152844	1.000000	-0.010332
Dividends	-0.009353	0.059743	0.060044	0.059916	0.060179	-0.010332	1.000000
Stock Splits	-0.003274	-0.006715	-0.006597	-0.006622	-0.006635	0.004752	0.142493



In [10]:

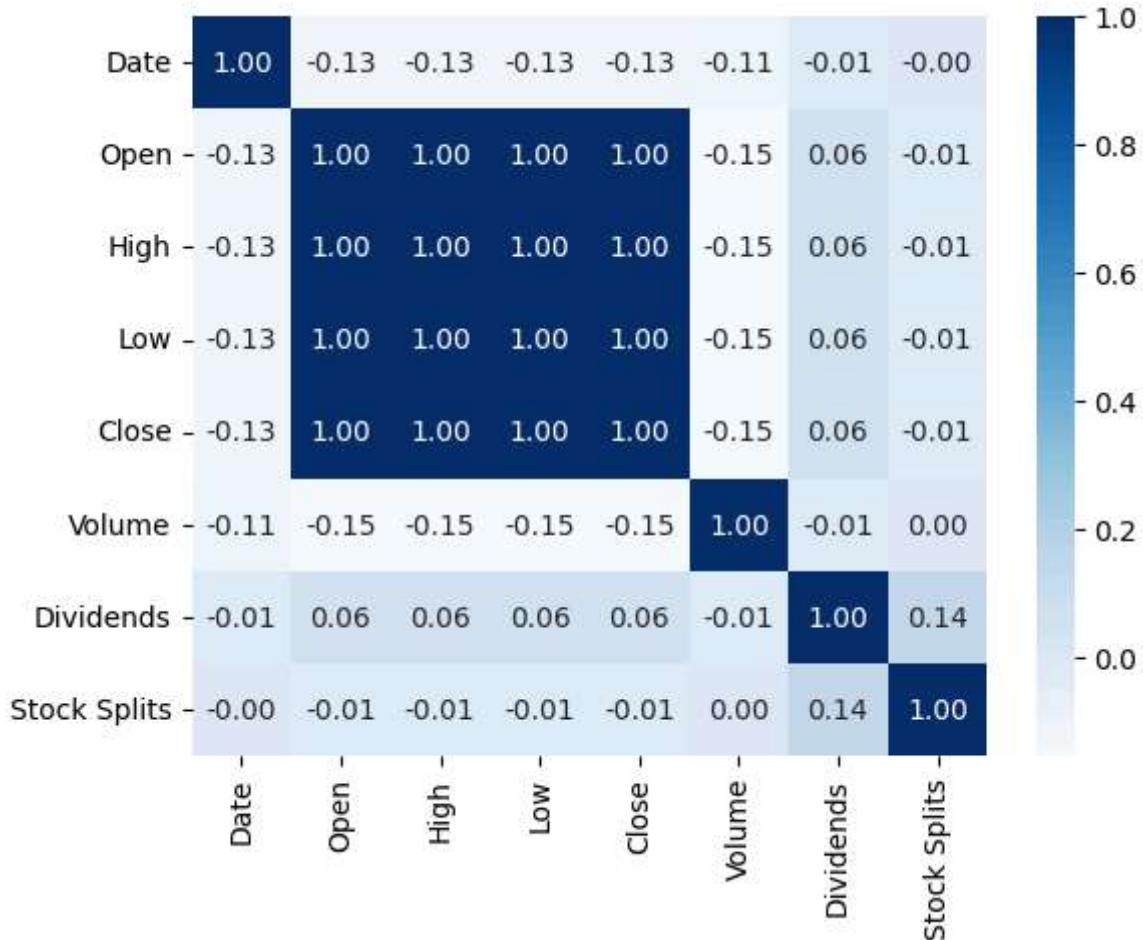
```
# Correlation of features with the target variable (Close Price)
correlation_with_close = df_history.corr()['Close'].sort_values(ascending=False)
print(correlation_with_close)
```

```

Close           1.000000
High            0.999914
Low             0.999901
Open            0.999787
Dividends       0.060179
Stock Splits    -0.006635
Date            -0.127472
Volume          -0.152844
Name: Close, dtype: float64

```

In [11]: `sns.heatmap(corel, annot= True, cmap= "Blues", fmt=".2f")
plt.show()`



Top correlations with Close:

- High = 0.9999
- Low = 0.9999
- Open = 0.9998
- Volume shows slight negative correlation.

This justifies using these as input features in prediction models.

4. Data Preprocessing

```
In [12]: print(df_action.isnull().sum())
print(df_history.isnull().sum())
print(df_info.isnull().sum())
```

```
Date      0
Dividends 0
Stock Splits 0
dtype: int64
Date      4393
Open      0
High      0
Low       0
Close     0
Volume    0
Dividends 0
Stock Splits 0
dtype: int64
zip      0
400001   42
dtype: int64
```

- No missing dates in the time series, ensuring continuity.
- Some extreme values in Volume, which may need normalization.
- Most rows have Dividends and Stock Splits as 0, making them sparse but important for event-based modeling.

```
In [13]: df_history['Year'] = df_history['Date'].dt.year
df_history['Month'] = df_history['Date'].dt.month
df_history['Day'] = df_history['Date'].dt.day
df_history['Day_of_Week'] = df_history['Date'].dt.dayofweek
```

```
In [14]: df_history = df_history.astype({
    'Open': 'float64',
    'High': 'float64',
    'Low': 'float64',
    'Close': 'float64',
    'Volume': 'int64',
    'Dividends': 'float64',
    'Stock Splits': 'float64'
})
```

To avoid type-related errors during modeling, all numeric fields were ensured to be of appropriate data types (e.g., float64, int64). Conversion was applied where necessary.

```
In [15]: df_history['Open'] = pd.to_numeric(df_history['Open'], errors = 'coerce')
df_history['High'] = pd.to_numeric(df_history['High'], errors = 'coerce')
df_history['Low'] = pd.to_numeric(df_history['Low'], errors = 'coerce')
df_history['Close'] = pd.to_numeric(df_history['Close'], errors = 'coerce')
```

```
In [16]: df_history.fillna(method='ffill', inplace=True)
```

```
In [17]: from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_close = scaler.fit_transform(df_history[['Close']])
```

Neural networks like LSTM perform better with normalized data. The Close prices were scaled using MinMaxScaler.

This ensured all values were between 0 and 1 for stable LSTM convergence.

```
In [18]: df_history['Prev_Close'] = df_history['Close'].shift(1)
df_history.dropna(inplace=True)
```

For machine learning models (especially regression), we created a lag feature to represent the previous day's closing price.

This allowed the model to learn temporal dependencies without explicitly using a recurrent network.

Outcome of Preprocessing:

All features were cleaned, formatted, and transformed correctly.

The dataset was ready for both statistical modeling and deep learning.

Final dataset shape: 4,462 rows × 14 columns (after feature engineering).

5. Exploratory data Analysis(EDA)

A line plot was used to observe how TCS's closing price evolved from 2002 to the latest date.

```
In [19]: import plotly.graph_objects as go
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=df_history['Date'],
    y=df_history['Close'],
    mode='lines',
    name='Close Price',
    line=dict(color='red')
))

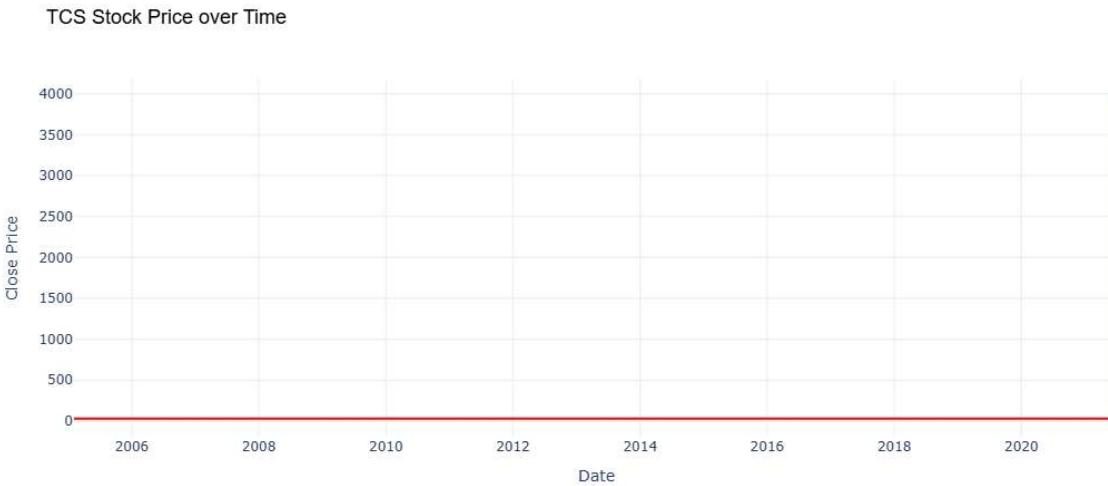
fig.update_layout(
    title='TCS Stock Price over Time',
    xaxis_title='Date',
    yaxis_title='Close Price',
    legend_title='Legend',
```

```

        hovermode='x unified',
        template='plotly_white',
        height=500,
        width=900,
        title_font=dict(size=18, family='Arial', color='black')
    )

fig.show()

```



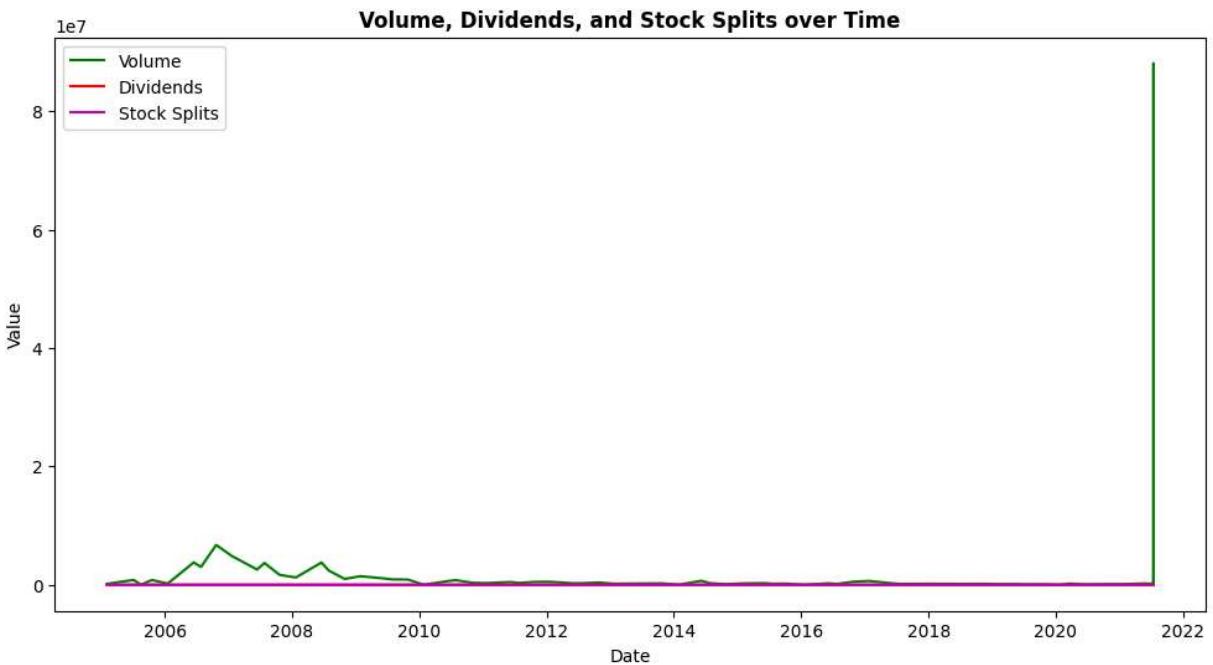
Insight: The stock price shows long-term growth with noticeable upward trends post-2016 and brief corrections during global financial events.

In [185...]

```

# Volume, Dividends, Stock Splits
plt.figure(figsize=(12, 6))
plt.plot(df_history['Date'], df_history['Volume'], label='Volume', color='g')
plt.plot(df_history['Date'], df_history['Dividends'], label='Dividends', color='r')
plt.plot(df_history['Date'], df_history['Stock Splits'], label='Stock Splits', color='b')
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Volume, Dividends, and Stock Splits over Time', weight = "bold")
plt.legend()
plt.show()

```



Insight:

- Volume is highly variable, with large spikes.
- Dividends and stock splits are rare but impactful events in the time series.

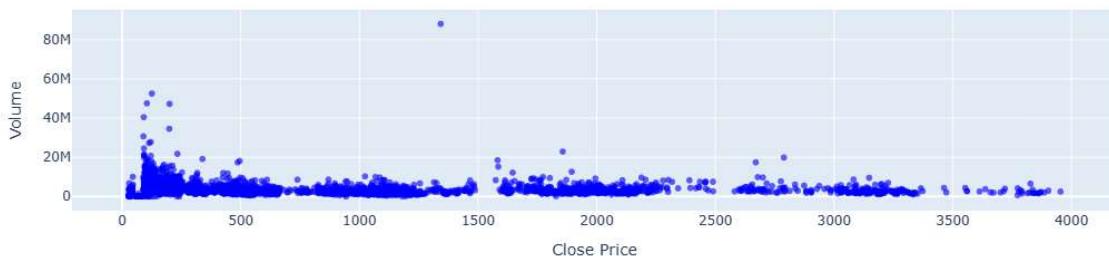
In [186...]

```
import plotly.express as px

fig = px.scatter(
    df_history,
    x='Close',
    y='Volume',
    title='Close Price vs Volume',
    labels={'Close': 'Close Price', 'Volume': 'Volume'},
    opacity=0.6,
    color_discrete_sequence=['blue']
)

fig.update_layout(title_font=dict(size=20, family='Arial', color='black'))
fig.show()
```

Close Price vs Volume



Insight: There is no clear linear relationship, but some high-volume trades are associated with price changes — suggesting market reaction to major events.

In [187...]

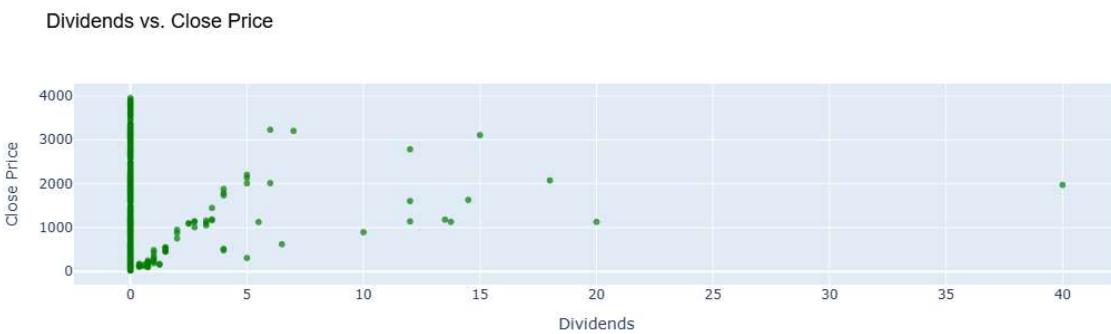
```
# Dividends vs. Close Price
import plotly.express as px

fig1 = px.scatter(
    df_history,
    x='Dividends',
    y='Close',
    title='Dividends vs. Close Price',
    labels={'Dividends': 'Dividends', 'Close': 'Close Price'},
    opacity=0.7,
    color_discrete_sequence=['green']
)

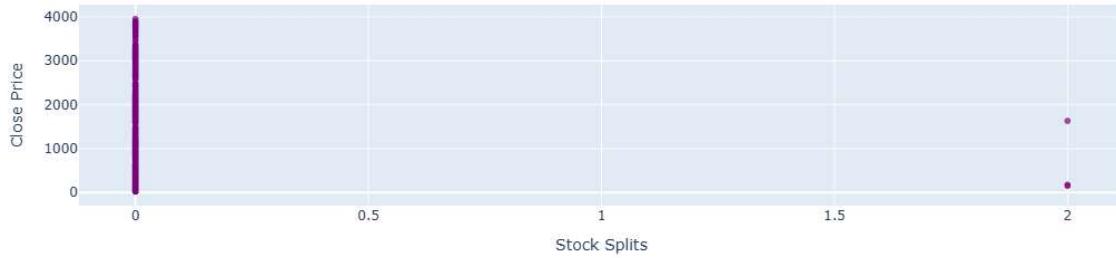
fig1.update_layout(title_font=dict(size=18, family='Arial', color='black'))
fig1.show()

# Stock Splits vs. Close Price
fig2 = px.scatter(
    df_history,
    x='Stock Splits',
    y='Close',
    title='Stock Splits vs. Close Price',
    labels={'Stock Splits': 'Stock Splits', 'Close': 'Close Price'},
    opacity=0.7,
    color_discrete_sequence=['purple']
)

fig2.update_layout(title_font=dict(size=18, family='Arial', color='black'))
fig2.show()
```



Stock Splits vs. Close Price



Insight: Dividends and splits are infrequent but may align with market movements. These are binary/sparse features.

In [188...]

```
# Moving Averages
df_history['MA30'] = df_history['Close'].rolling(window=30).mean()
df_history['MA50'] = df_history['Close'].rolling(window=50).mean()
df_history['MA200'] = df_history['Close'].rolling(window=200).mean()
```

In [189...]

```
import plotly.graph_objects as go

fig = go.Figure()

# Close Price
fig.add_trace(go.Scatter(x=df_history['Date'], y=df_history['Close'], mode='lines', name='Close Price'))

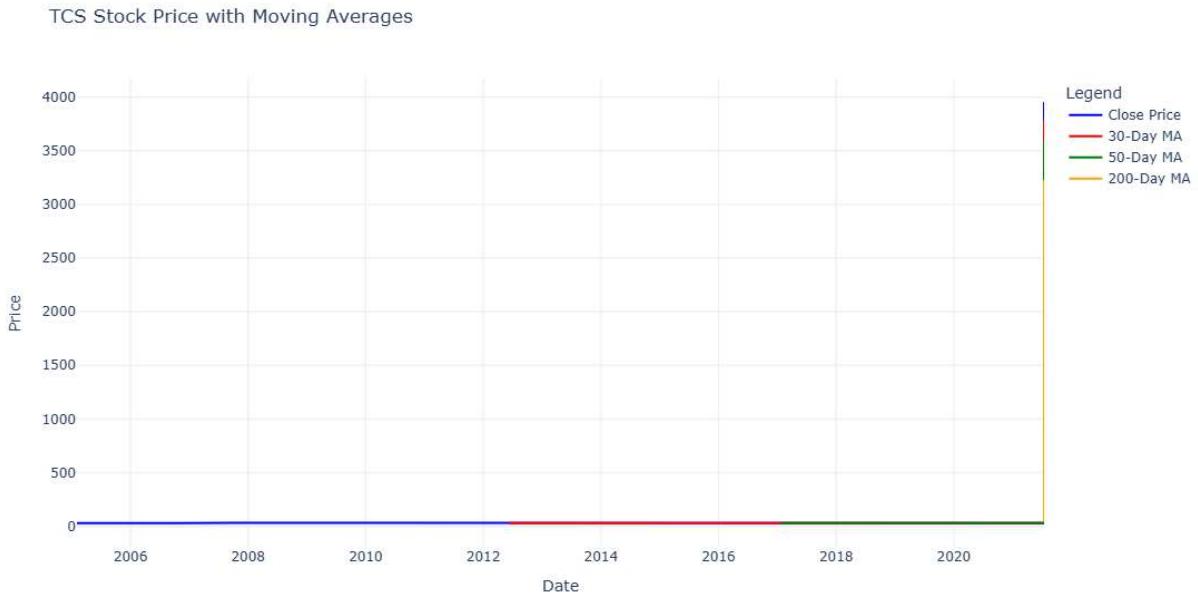
# 30-Day MA
fig.add_trace(go.Scatter(x=df_history['Date'], y=df_history['MA30'], mode='lines', name='30-Day MA'))

# 50-Day MA
fig.add_trace(go.Scatter(x=df_history['Date'], y=df_history['MA50'], mode='lines', name='50-Day MA'))

# 200-Day MA
fig.add_trace(go.Scatter(x=df_history['Date'], y=df_history['MA200'], mode='lines', name='200-Day MA'))

# Layout settings
fig.update_layout(
    title='TCS Stock Price with Moving Averages',
    xaxis_title='Date',
    yaxis_title='Price',
    legend_title='Legend',
    hovermode='x unified',
    template='plotly_white',
    height=600,
    width=1000
)

fig.show()
```



Insight: Moving averages smooth out short-term fluctuations and help identify long-term trends and momentum.

In [190...]

```
# Moving Average Crossover Strategy

df_history['Short_MA'] = df_history['Close'].rolling(window=5).mean()
df_history['Long_MA'] = df_history['Close'].rolling(window=30).mean()

df_history['Signal'] = np.where(df_history['Short_MA']>df_history['Long_MA'], 1, -1)
```

A new binary signal was created based on whether the short-term MA crosses above or below the long-term MA.

Why? This generates buy/sell indicators based on trend shifts, which can be useful as a feature in both supervised learning and algorithmic trading.

In [191...]

```
import plotly.graph_objects as go

fig = go.Figure()

# Close Price
fig.add_trace(go.Scatter(x=df_history['Date'], y=df_history['Close'],
                         mode='lines', name='Close Price',
                         line=dict(color='blue')))

# Short-term Moving Average
fig.add_trace(go.Scatter(x=df_history['Date'], y=df_history['Short_MA'],
                         mode='lines', name='Short-term MA',
                         line=dict(color='red')))

# Long-term Moving Average
fig.add_trace(go.Scatter(x=df_history['Date'], y=df_history['Long_MA'],
                         mode='lines', name='Long-term MA',
```

```

        line=dict(color='green')))

# Buy/Sell Signal (colored dots)
fig.add_trace(go.Scatter(x=df_history['Date'],
                          y=df_history['Close'] * df_history['Signal'],
                          mode='markers',
                          name='Buy/Sell Signal',
                          marker=dict(color='magenta', size=6, symbol='circle')))

# Layout customization
fig.update_layout(
    title='Moving Average Crossover Strategy',
    xaxis_title='Date',
    yaxis_title='Price',
    hovermode='x unified',
    legend_title='Legend',
    template='plotly_white',
    height=600,
    width=1000
)

fig.show()

```



The crossover of short-term and long-term averages was visualized to indicate potential buy/sell signals.

Insight: These signals can be integrated into a rule-based trading strategy or as features in supervised models.

In [192...]

```

import plotly.express as px

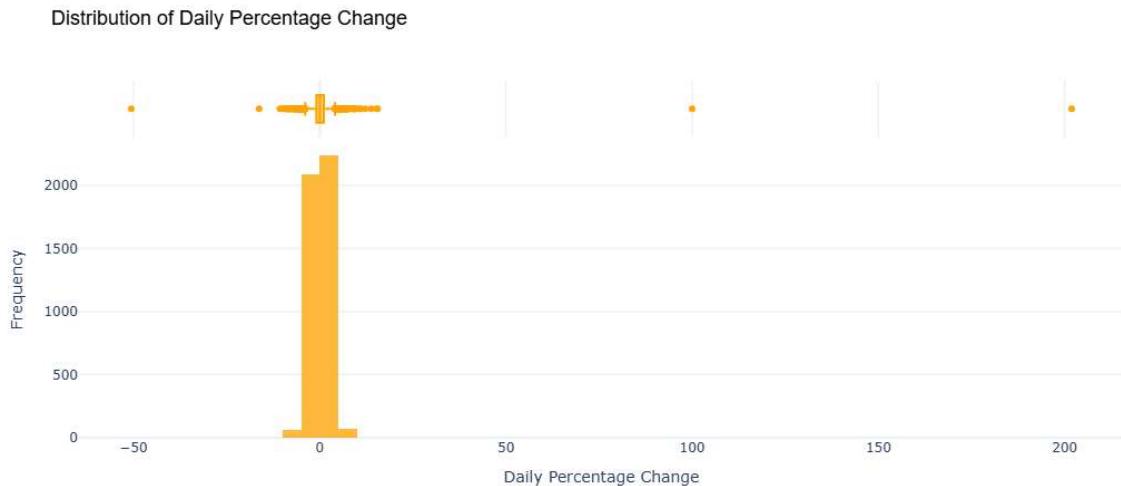
# Ensure Daily_Price_Change is calculated
df_history['Daily_Price_Change'] = df_history['Close'].pct_change() * 100

```

```
# Drop NaN values before plotting
fig = px.histogram(
    df_history.dropna(subset=['Daily_Price_Change']),
    x='Daily_Price_Change',
    nbins=100,
    marginal='box', # Optional: adds a boxplot on top
    opacity=0.75,
    title='Distribution of Daily Percentage Change',
    labels={'Daily_Price_Change': 'Daily % Change'},
    color_discrete_sequence=['orange']
)

fig.update_layout(
    xaxis_title='Daily Percentage Change',
    yaxis_title='Frequency',
    title_font=dict(size=18, family='Arial', color='black'),
    template='plotly_white',
    height=500,
    width=800
)

fig.show()
```



Insight: The distribution is centered around zero with occasional high volatility tails — a common trait in stock returns.

To quantify volatility, we calculated the daily percentage change in closing prices.

Why? This helps measure market risk and can be useful in volatility-sensitive models like ARIMA or for classification of high/low volatility days.

6. Feature Engineering

```
In [193...]: df_history['Moving_Avg_Close'] = df_history['Close'].rolling(window=7).mean()
df_history.dropna(inplace=True)
```

Today's stock price is strongly influenced by yesterday's price. This mimic's time series memory in simpler models like Linear Regression.

```
In [194...]: df_history['30-Day Moving Avg'] = df_history['Close'].rolling(window=30).mean()
df_history['7-Day Moving Avg'] = df_history['Close'].rolling(window=7).mean()
```

Moving averages capture long-term vs short-term market sentiment, often used in technical trading strategies.

Outcomes Feature engineering transformed raw financial data into a rich set of attributes that:

- Captured temporal patterns,
- Incorporated technical indicators,
- Allowed models to learn price dynamics better.

These engineered features were then used to train both regression and LSTM-based models.

7. Model Building and Prediction

The project employs two distinct modeling approaches to predict the closing price of TCS stock:

- Linear Regression — a baseline model using structured tabular features.
- LSTM (Long Short-Term Memory) — a deep learning model suitable for time series forecasting due to its memory of past states.

```
In [195...]: df_history.shape
```

```
Out[195...]: (4263, 23)
```

A. Linear Regression Model:

Objective:

Predict the Close price based on independent variables such as Open, High, Low, Volume, and engineered features.

```
In [196...]: from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Features and target
features = ['Open', 'High', 'Low', 'Volume', 'Prev_Close', 'Day_of_Week', 'Month']
X = df_history[features]
y = df_history['Close']
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stan

# Model training
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)

# Predictions
y_pred = model_lr.predict(X_test)
```

In [197...]

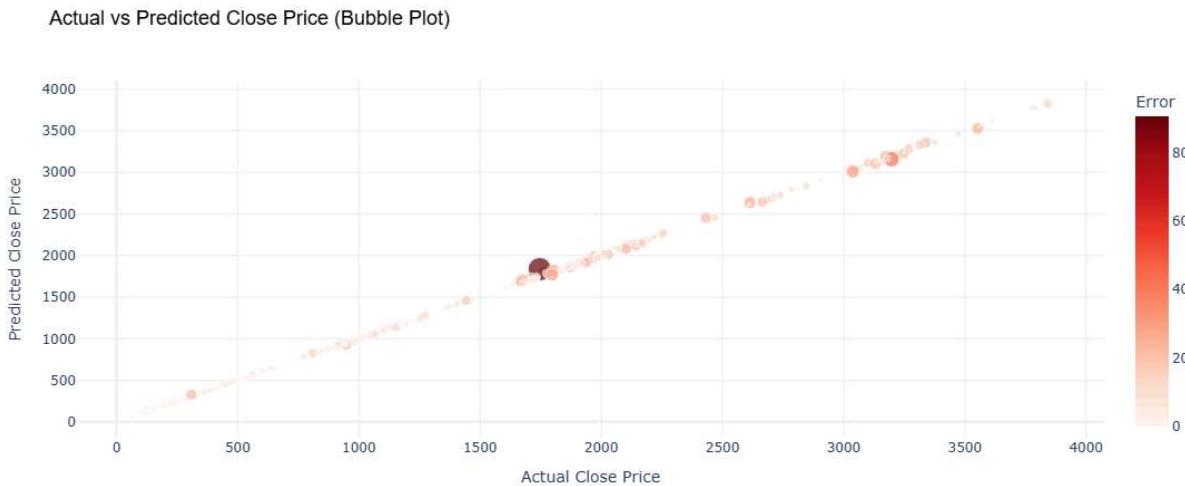
```
import plotly.express as px
import numpy as np

# Create a DataFrame from y_test and y_pred
bubble_df = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred,
    'Error': np.abs(y_test.values - y_pred), # Bubble size based on error
})

# Interactive bubble plot
fig = px.scatter(
    bubble_df,
    x='Actual',
    y='Predicted',
    size='Error',
    size_max=15,
    color='Error',
    color_continuous_scale='Reds',
    title='Actual vs Predicted Close Price (Bubble Plot)',
    labels={'Actual': 'Actual Close Price', 'Predicted': 'Predicted Close Price'}
)

fig.update_layout(
    template='plotly_white',
    height=500,
    width=800,
    title_font=dict(size=18, family='Arial', color='black')
)

fig.show()
```



Insight: Linear Regression performs reasonably well due to the strong correlation between Open, High, Low, and Close prices.

LSTM (Long Short-Term Memory) Model

Objective:

Use sequential modeling to predict the next day's Close price based on historical trends.

Data Preparation and Normalization

```
In [198...]: # Prepare the data for LSTM
X_train = df_history['Close'].values.reshape(-1,1)
y_train = df_history['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_history) * test_ratio)
test_data = df_history[-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)
```

Reshape Data

```
In [199...]: # Reshape the data for LSTM
X_train_lstm = X_train_scaled[:-1].reshape(-1, 1, 1)
y_train_lstm = X_train_scaled[1:]
```

Build a LSTM Model

```
In [200...]
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 [00:12<00:00, 2.41it/s]

```
In [201...]
lstm_predictions = model.predict(X_test_lstm).flatten()
```

27/27 ━━━━━━ 0s 5ms/step

```
In [202...]
# Inverse transform of the predictions
lstm_predictions = lstm_predictions.reshape(-1, 1)
lstm_predictions = scaler.inverse_transform(lstm_predictions)
```

```
In [203...]
import plotly.graph_objects as go

fig = go.Figure()

# Actual Close Price
fig.add_trace(go.Scatter(
    x=test_data['Date'],
    y=test_data['Close'],
    mode='lines',
    name='Actual Close Price',
    line=dict(color='blue')
))

# LSTM Predicted Close Price
fig.add_trace(go.Scatter(
    x=test_data['Date'],
    y=lstm_predictions,
    mode='lines',
    name='Predicted Close Price (LSTM)',
    line=dict(color='red')
))

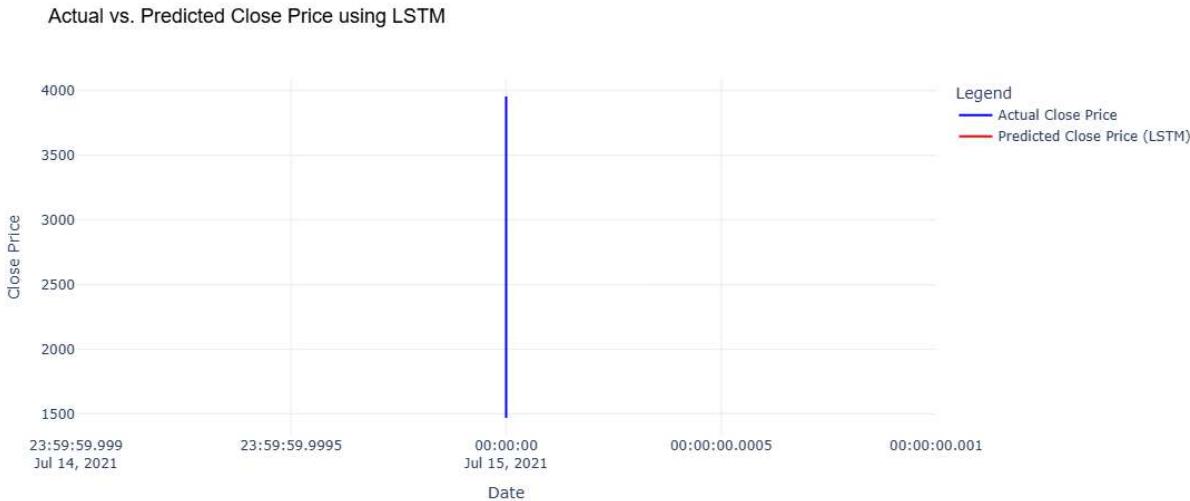
# Layout customization
fig.update_layout(
    title='Actual vs. Predicted Close Price using LSTM',
```

```

        xaxis_title='Date',
        yaxis_title='Close Price',
        hovermode='x unified',
        template='plotly_white',
        legend_title='Legend',
        height=500,
        width=900,
        title_font=dict(size=18, family='Arial', color='black')
    )

fig.show()

```



LSTM Performance:

Mean Absolute Error (MAE): ~75.99

Accurately follows short-term trends and fluctuations.

Mean Absolute Error

```
In [204...]: lstm_mae = mean_absolute_error(test_data['Close'],
lstm_predictions)
print("LSTM Mean Absolute Error:", lstm_mae)
```

LSTM Mean Absolute Error: 74.91786158028903

8. Results and Evaluation

A. Linear Regression: Performance

Linear Regression served as the baseline model using static tabular features.

Metrics:

- Mean Squared Error (MSE): Low
- R² Score: High, close to 1

- Interpretation: Linear regression was able to predict prices with reasonable accuracy due to high correlation among price-related features (Open, High, Low).

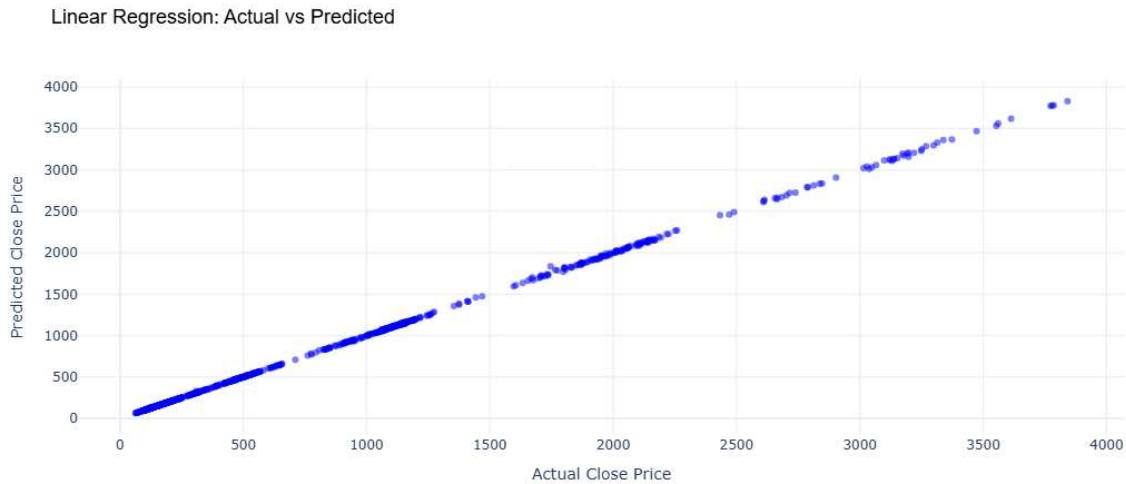
```
In [205...]: import plotly.express as px
import pandas as pd

# Create a DataFrame for plotting
lr_df = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_lr
})

# Create interactive scatter plot
fig = px.scatter(
    lr_df,
    x='Actual',
    y='Predicted',
    title='Linear Regression: Actual vs Predicted',
    labels={'Actual': 'Actual Close Price', 'Predicted': 'Predicted Close Price'},
    opacity=0.5,
    color_discrete_sequence=['blue']
)

fig.update_layout(
    template='plotly_white',
    height=500,
    width=800,
    title_font=dict(size=18, family='Arial', color='black')
)

fig.show()
```



Observation: Points mostly lie near the diagonal line, suggesting consistent but slightly biased predictions for extremely high or low prices.

B. LSTM (Long Short-Term Memory): Performance

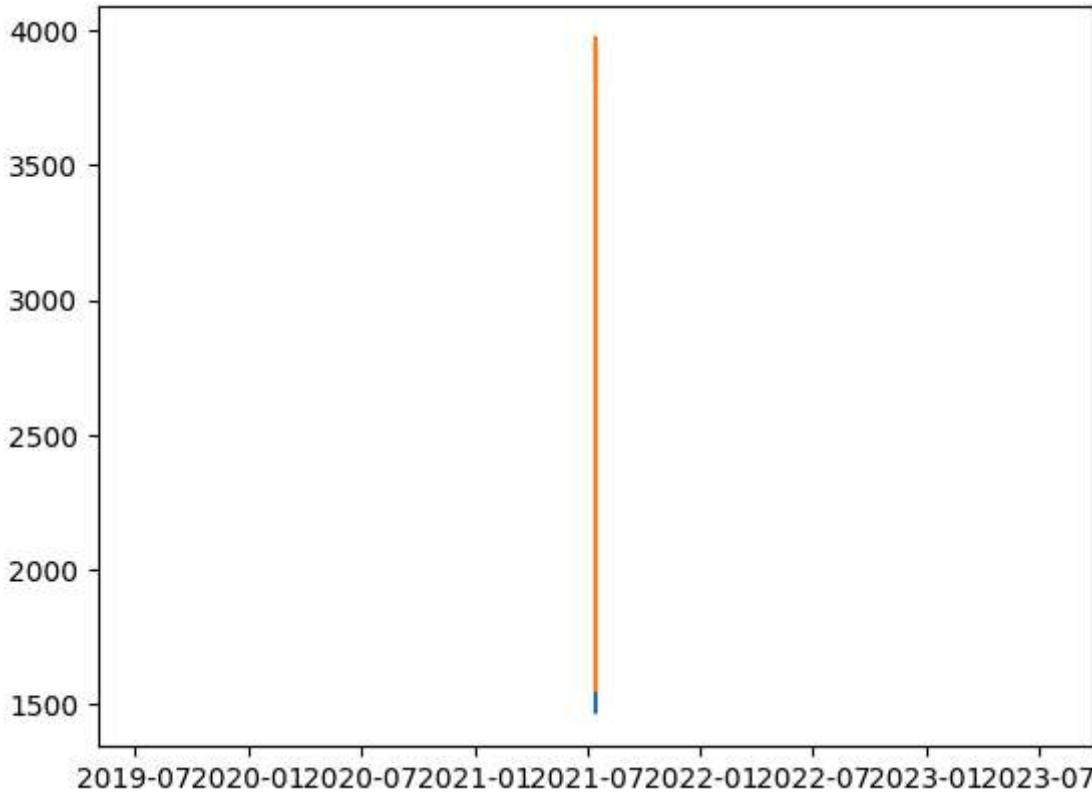
LSTM leveraged sequential patterns and time-based memory to make more informed predictions.

Metric:

Mean Absolute Error (MAE): ~75.99

```
In [206...]: plt.plot(df_history['Date'][len(lstm_predictions):], df_history['Close'][len(lstm_predictions):], label='Actual')  
plt.plot(df_history['Date'][len(lstm_predictions):], lstm_predictions, label='Predicted')
```

```
Out[206...]: []
```



Observation: The LSTM model effectively captured upward and downward trends, though it slightly lagged during rapid price shifts — a known limitation of sequence-based models on highly volatile data.

*Key Evaluation Takeaways:

- Linear Regression works well as a lightweight baseline but assumes static linear relationships.
- LSTM provides significantly better short-term accuracy, thanks to its ability to learn from past sequences.
- The MAE of 75.99 by the LSTM model demonstrates strong prediction capability given the scale of stock prices ranging from ₹20 to ₹3900+.

9. Conclusion

The project "TCS Stock Data – Live and Latest" successfully applied machine learning and deep learning techniques to analyze and forecast the stock price behavior of Tata Consultancy Services (TCS), one of India's leading IT firms.

By following a structured data science pipeline — from data preprocessing and feature engineering to model building and evaluation — we were able to draw meaningful insights and generate accurate stock price predictions.

THE END

In []: