```python
# Disable warnings
import warnings
warnings.filterwarnings("ignore")

# Import libraries
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib import dates

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Activation, Dropout

from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_absolute_percentage_error

# Loading data and displaying the first 5 rows

df = pd.read_csv('NSE-TATAGLOBAL.csv')
df.head()
```

```
         Date    Open    High     Low    Last   Close  Total Trade
Quantity  \
0  2018-09-28  234.05  235.95  230.20  233.50  233.75
3069914
1  2018-09-27  234.55  236.80  231.10  233.80  233.25
5082859
2  2018-09-26  240.00  240.00  232.50  235.00  234.25
2240909
3  2018-09-25  233.30  236.75  232.00  236.25  236.10
2349368
4  2018-09-24  233.55  239.20  230.75  234.00  233.30
3423509

   Turnover (Lacs)
0          7162.35
1         11859.95
2          5248.60
3          5503.90
4          7999.55
```

```python
# Checking the dataset for duplicate rows

print('Number of duplicate rows - ', len(df[df.duplicated()].values))
```

```
Number of duplicate rows -  0
```

```
# Checking the number of rows, data types and the absence of missing
values

df.info()
```

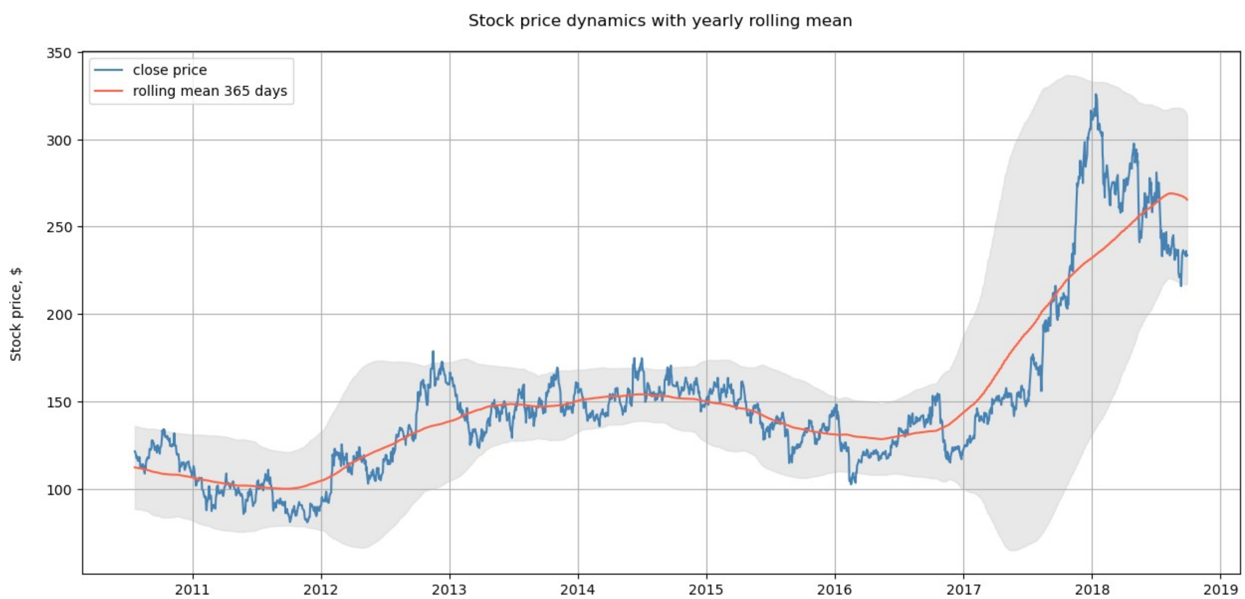
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2035 entries, 0 to 2034
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  2035 non-null   object
 1   Open                  2035 non-null   float64
 2   High                  2035 non-null   float64
 3   Low                   2035 non-null   float64
 4   Last                  2035 non-null   float64
 5   Close                 2035 non-null   float64
 6   Total Trade Quantity  2035 non-null   int64
 7   Turnover (Lacs)       2035 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 127.3+ KB
```

```
# Convert Date to datatime format and sort by date

df.Date = pd.to_datetime(df.Date, dayfirst=True)
df = df.sort_values('Date', ascending=True)
df.head()
```

```
           Date   Open    High     Low    Last   Close  Total Trade
Quantity  \
2034 2010-07-21  122.1  123.00  121.05  121.10  121.55
658666
2033 2010-07-22  120.3  122.00  120.25  120.75  120.90
293312
2032 2010-07-23  121.8  121.95  120.25  120.35  120.65
281312
2031 2010-07-26  120.1  121.00  117.10  117.10  117.60
658440
2030 2010-07-27  117.6  119.50  112.00  118.80  118.65
586100

      Turnover (Lacs)
2034           803.56
2033           355.17
2032           340.31
2031           780.01
2030           694.98
```

```
# Will build the model for the price of stocks at the close of trading

# Display graphs of the dynamics of the stock price, as well as a
rolling mean with a month and year window and Bollinger
```

```
# bands with 95% confidence interval

ts = df[['Date', 'Close']].rename(columns={'Close': 'Close
price'}).set_index('Date').squeeze()

rolling_mean_month = ts.rolling(window=30, center=True,
min_periods=15).mean()
rolling_std_month = ts.rolling(window=30, center=True,
min_periods=15).std()
lower_bound_month = rolling_mean_month - (1.96 * rolling_std_month)
upper_bound_month = rolling_mean_month + (1.96 * rolling_std_month)

rolling_mean_year = ts.rolling(window=365, center=True,
min_periods=182).mean()
rolling_std_year = ts.rolling(window=365, center=True,
min_periods=182).std()
lower_bound_year = rolling_mean_year - (1.96 * rolling_std_year)
upper_bound_year = rolling_mean_year + (1.96 * rolling_std_year)

fig, ax = plt.subplots(2, 1, figsize=(15, 15))

ax[0].plot(ts, label='close price', color='steelblue')
ax[0].plot(rolling_mean_month, 'g', label='rolling mean 30 days',
color='tomato')
ax[0].fill_between(x=ts.index, y1=lower_bound_month,
y2=upper_bound_month, color='lightgrey', alpha=0.5)
ax[0].legend(loc='upper left')
ax[0].set_title('Stock price dynamics with monthly rolling mean\n')
ax[0].set_ylabel('Stock price, $\n')
ax[0].xaxis.set_major_locator(dates.AutoDateLocator())
ax[0].grid()

ax[1].plot(ts, label='close price', color='steelblue')
ax[1].plot(rolling_mean_year, 'g', label='rolling mean 365 days',
color='tomato')
ax[1].fill_between(x=ts.index, y1=lower_bound_year,
y2=upper_bound_year, color='lightgrey', alpha=0.5)
ax[1].legend(loc='upper left')
ax[1].set_title('Stock price dynamics with yearly rolling mean\n')
ax[1].set_ylabel('Stock price, $\n')
ax[1].xaxis.set_major_locator(dates.AutoDateLocator())
ax[1].grid();
```




```
# Put historical data for the model in a separate dataframe

analysis_df = df[['Date', 'Close']].rename(columns={'Close': 'Close
price'}).reset_index().drop(columns='index')

# Create a numpy array for the stock price column at the close of
trading

close_price = analysis_df['Close price'].to_numpy().reshape(-1, 1)

# Normalizing data with MinMaxScaler

scaler = MinMaxScaler()
```

```
scaler.fit(close_price)
close_price_scaled = scaler.transform(close_price)

# Set the date of separation of historical data into training and test
ones and the length of the sequences for the formation
# of arrays fed to the input of the model

len_sqn = 100
split_date = '2018-01-01'

# Determining the index corresponding to the selected date
split_index = analysis_df[analysis_df.iloc[:,0]==split_date].index[0]

# Creating function to form an array contains sequences of length
len_sqn with an offset of 1 day (more precisely, 1 date from
# the time series, since not all days in a row are present in the
series)

def to_sequences(data, raw_len):
    sqn_list = []
    for index in range(len(data) - raw_len):
        sqn_list.append(data[index: index + raw_len])
    return np.array(sqn_list)

# Creation a function for splitting on training and test samples. The
input is a 1d array of stock price values, the length
# of the sequences, and the date of division into a train/test.
Output: dataset divided into training and test samples as
# numpy arrays, where X_train, X_test are sets of sequences of length
len_sqn-1, and y_train, y_test are an array of the last
# values of each of the received sequences.

def preprocessing(data_1d_array, raw_len, index):
    data = to_sequences(data_1d_array, raw_len)
    X_train = data[:index, :-1, :]
    y_train = data[:index, -1, :]
    X_test = data[index:, :-1, :]
    y_test = data[index:, -1, :]
    return X_train, y_train, X_test, y_test

train_x, train_y, test_x, test_y = preprocessing(close_price_scaled,
len_sqn, split_index)

# Building and compiling a model with 4 LSTM and Dense layers. Use
dropout layers after each LSTM layer to reduce overfitting.
# Take MSE as a loss function.

windows_size = len_sqn - 1

model = Sequential()
model.add(LSTM(windows_size, return_sequences=True,
```

```python
input_shape=(windows_size, train_x.shape[-1])))
model.add(Dropout(0.4))
model.add(LSTM(windows_size*2, return_sequences = True))
model.add(Dropout(0.4))
model.add(LSTM(windows_size*2, return_sequences = True))
model.add(Dropout(0.4))
model.add(LSTM(windows_size))
model.add(Dropout(0.6))
model.add(Dense(1, activation='linear'))

model.compile(loss='mean_squared_error', optimizer='adam')

# Fit model on 70 epochs with a batch size of 32.

history = model.fit(train_x, train_y, epochs=70, batch_size=32,
validation_split=0.1)
```

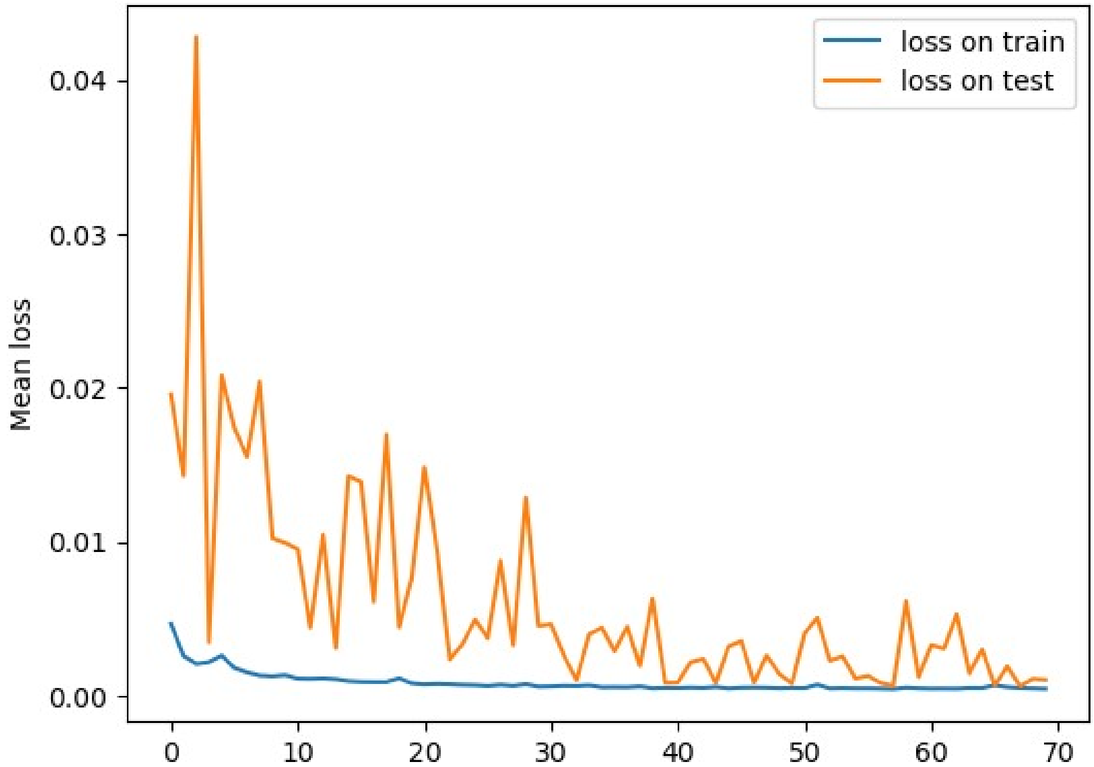
```
Epoch 1/70
53/53 [==============================] - 7s 54ms/step - loss: 0.0047 -
val_loss: 0.0196
Epoch 2/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0026 -
val_loss: 0.0143
Epoch 3/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0020 -
val_loss: 0.0428
Epoch 4/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0022 -
val_loss: 0.0034
Epoch 5/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0026 -
val_loss: 0.0208
Epoch 6/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0018 -
val_loss: 0.0174
Epoch 7/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0015 -
val_loss: 0.0155
Epoch 8/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0013 -
val_loss: 0.0204
Epoch 9/70
53/53 [==============================] - 2s 36ms/step - loss: 0.0012 -
val_loss: 0.0102
Epoch 10/70
53/53 [==============================] - 2s 36ms/step - loss: 0.0013 -
val_loss: 0.0099
Epoch 11/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0011 -
val_loss: 0.0095
```

```
Epoch 12/70
53/53 [==============================] - 2s 36ms/step - loss: 0.0011 -
val_loss: 0.0044
Epoch 13/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0011 -
val_loss: 0.0105
Epoch 14/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0011 -
val_loss: 0.0031
Epoch 15/70
53/53 [==============================] - 2s 35ms/step - loss: 9.2727e-
04 - val_loss: 0.0142
Epoch 16/70
53/53 [==============================] - 2s 35ms/step - loss: 8.8247e-
04 - val_loss: 0.0139
Epoch 17/70
53/53 [==============================] - 2s 36ms/step - loss: 8.7195e-
04 - val_loss: 0.0061
Epoch 18/70
53/53 [==============================] - 2s 36ms/step - loss: 8.7092e-
04 - val_loss: 0.0170
Epoch 19/70
53/53 [==============================] - 2s 35ms/step - loss: 0.0011 -
val_loss: 0.0044
Epoch 20/70
53/53 [==============================] - 2s 35ms/step - loss: 7.8538e-
04 - val_loss: 0.0076
Epoch 21/70
53/53 [==============================] - 2s 35ms/step - loss: 7.1489e-
04 - val_loss: 0.0148
Epoch 22/70
53/53 [==============================] - 2s 35ms/step - loss: 7.4941e-
04 - val_loss: 0.0094
Epoch 23/70
53/53 [==============================] - 2s 35ms/step - loss: 7.2412e-
04 - val_loss: 0.0023
Epoch 24/70
53/53 [==============================] - 2s 36ms/step - loss: 6.8236e-
04 - val_loss: 0.0033
Epoch 25/70
53/53 [==============================] - 2s 36ms/step - loss: 6.6532e-
04 - val_loss: 0.0049
Epoch 26/70
53/53 [==============================] - 2s 35ms/step - loss: 6.1055e-
04 - val_loss: 0.0037
Epoch 27/70
53/53 [==============================] - 2s 36ms/step - loss: 6.9229e-
04 - val_loss: 0.0088
Epoch 28/70
```

```
53/53 [==============================] - 2s 35ms/step - loss: 6.1588e-
04 - val_loss: 0.0033
Epoch 29/70
53/53 [==============================] - 2s 35ms/step - loss: 7.3162e-
04 - val_loss: 0.0129
Epoch 30/70
53/53 [==============================] - 2s 35ms/step - loss: 5.6975e-
04 - val_loss: 0.0045
Epoch 31/70
53/53 [==============================] - 2s 35ms/step - loss: 5.8553e-
04 - val_loss: 0.0046
Epoch 32/70
53/53 [==============================] - 2s 35ms/step - loss: 6.2482e-
04 - val_loss: 0.0026
Epoch 33/70
53/53 [==============================] - 2s 35ms/step - loss: 6.1285e-
04 - val_loss: 9.9073e-04
Epoch 34/70
53/53 [==============================] - 2s 35ms/step - loss: 6.5481e-
04 - val_loss: 0.0040
Epoch 35/70
53/53 [==============================] - 2s 35ms/step - loss: 5.2815e-
04 - val_loss: 0.0044
Epoch 36/70
53/53 [==============================] - 2s 35ms/step - loss: 5.3554e-
04 - val_loss: 0.0029
Epoch 37/70
53/53 [==============================] - 2s 35ms/step - loss: 5.2706e-
04 - val_loss: 0.0045
Epoch 38/70
53/53 [==============================] - 2s 35ms/step - loss: 5.8494e-
04 - val_loss: 0.0019
Epoch 39/70
53/53 [==============================] - 2s 35ms/step - loss: 4.6017e-
04 - val_loss: 0.0063
Epoch 40/70
53/53 [==============================] - 2s 35ms/step - loss: 4.8893e-
04 - val_loss: 8.2490e-04
Epoch 41/70
53/53 [==============================] - 2s 35ms/step - loss: 4.7941e-
04 - val_loss: 8.3487e-04
Epoch 42/70
53/53 [==============================] - 2s 35ms/step - loss: 5.0417e-
04 - val_loss: 0.0022
Epoch 43/70
53/53 [==============================] - 2s 35ms/step - loss: 4.8318e-
04 - val_loss: 0.0024
Epoch 44/70
53/53 [==============================] - 2s 35ms/step - loss: 5.4295e-
```

```
04 - val_loss: 8.0532e-04
Epoch 45/70
53/53 [==============================] - 2s 35ms/step - loss: 4.4945e-04 - val_loss: 0.0032
Epoch 46/70
53/53 [==============================] - 2s 35ms/step - loss: 4.9933e-04 - val_loss: 0.0035
Epoch 47/70
53/53 [==============================] - 2s 35ms/step - loss: 5.0744e-04 - val_loss: 8.3076e-04
Epoch 48/70
53/53 [==============================] - 2s 35ms/step - loss: 4.9696e-04 - val_loss: 0.0026
Epoch 49/70
53/53 [==============================] - 2s 35ms/step - loss: 4.6549e-04 - val_loss: 0.0014
Epoch 50/70
53/53 [==============================] - 2s 35ms/step - loss: 4.8807e-04 - val_loss: 7.6775e-04
Epoch 51/70
53/53 [==============================] - 2s 35ms/step - loss: 4.7260e-04 - val_loss: 0.0040
Epoch 52/70
53/53 [==============================] - 2s 35ms/step - loss: 6.9280e-04 - val_loss: 0.0050
Epoch 53/70
53/53 [==============================] - 2s 36ms/step - loss: 4.5120e-04 - val_loss: 0.0023
Epoch 54/70
53/53 [==============================] - 2s 35ms/step - loss: 4.7302e-04 - val_loss: 0.0025
Epoch 55/70
53/53 [==============================] - 2s 35ms/step - loss: 4.5406e-04 - val_loss: 0.0011
Epoch 56/70
53/53 [==============================] - 2s 35ms/step - loss: 4.5546e-04 - val_loss: 0.0013
Epoch 57/70
53/53 [==============================] - 2s 35ms/step - loss: 4.3992e-04 - val_loss: 8.0752e-04
Epoch 58/70
53/53 [==============================] - 2s 35ms/step - loss: 4.1858e-04 - val_loss: 6.1888e-04
Epoch 59/70
53/53 [==============================] - 2s 35ms/step - loss: 5.0122e-04 - val_loss: 0.0061
Epoch 60/70
53/53 [==============================] - 2s 35ms/step - loss: 4.5892e-04 - val_loss: 0.0012
```

```
Epoch 61/70
53/53 [==============================] - 2s 36ms/step - loss: 4.3969e-04 - val_loss: 0.0033
Epoch 62/70
53/53 [==============================] - 2s 35ms/step - loss: 4.4311e-04 - val_loss: 0.0030
Epoch 63/70
53/53 [==============================] - 2s 35ms/step - loss: 4.3684e-04 - val_loss: 0.0053
Epoch 64/70
53/53 [==============================] - 2s 35ms/step - loss: 4.7996e-04 - val_loss: 0.0014
Epoch 65/70
53/53 [==============================] - 2s 35ms/step - loss: 4.7774e-04 - val_loss: 0.0030
Epoch 66/70
53/53 [==============================] - 2s 35ms/step - loss: 6.6265e-04 - val_loss: 6.6018e-04
Epoch 67/70
53/53 [==============================] - 2s 35ms/step - loss: 5.3631e-04 - val_loss: 0.0019
Epoch 68/70
53/53 [==============================] - 2s 35ms/step - loss: 4.7768e-04 - val_loss: 5.9199e-04
Epoch 69/70
53/53 [==============================] - 2s 35ms/step - loss: 4.5470e-04 - val_loss: 0.0011
Epoch 70/70
53/53 [==============================] - 2s 35ms/step - loss: 4.2890e-04 - val_loss: 0.0010

# Learning history visualization

plt.plot(history.history['loss'], label='loss on train')
plt.plot(history.history['val_loss'], label='loss on test')
plt.ylabel('Mean loss')
plt.legend(loc='upper right')
plt.show()
```




```
# Make a prediction for the test sample

pred_y = model.predict(test_x)

3/3 [==============================] - 1s 15ms/step

# Performing an inverse transformation of the predicted and test
values

inversed_test_y = scaler.inverse_transform(test_y)
inversed_pred_y = scaler.inverse_transform(pred_y)

# Get the root mean squared error (RMSE)

rmse = mean_squared_error(inversed_test_y, inversed_pred_y,
squared=False)
rmse

6.195235388479212

# Visualize the historycal and prediction data

train = analysis_df[:(split_index+100)].set_index('Date')
valid = analysis_df[(split_index+100):].set_index('Date')
valid['predict'] = inversed_pred_y

plt.figure(figsize=(16,8))
```

```
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close price'])
plt.plot(valid[['Close price', 'predict']])
plt.legend(['Train', 'Val', 'predict'], loc='lower right')
plt.show()
```