

STOCK PRICE PREDICTION (Phase 4)

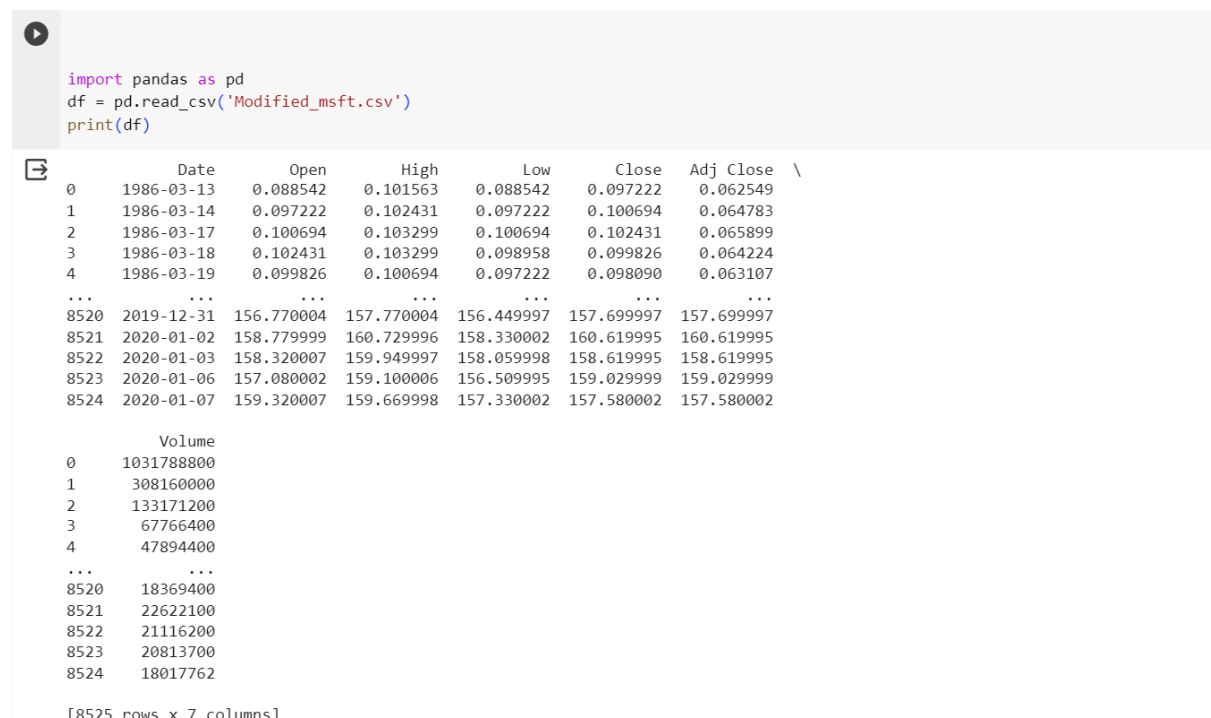
Madras Institute of Technology

2021506088 – Santhi Priya D N
2021506096 - Shiek Sajnathul Faizana
2021506103 – Sowmiya J
2021506323 - Revanth P

INTRODUCTION:

In Phase 4 of our stock price prediction project, we continue to advance our efforts to create a robust and accurate predictive model. Building upon the foundation laid in previous phases, this stage is dedicated to key activities that are instrumental in developing a model that can make well-informed predictions, optimize investment strategies, and adapt to dynamic market conditions.

Reading the dataset:



```
import pandas as pd
df = pd.read_csv('Modified_msft.csv')
print(df)
```

	Date	Open	High	Low	Close	Adj Close	\
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	
...	
8520	2019-12-31	156.770004	157.770004	156.449997	157.699997	157.699997	
8521	2020-01-02	158.779999	160.729996	158.330002	160.619995	160.619995	
8522	2020-01-03	158.320007	159.949997	158.059998	158.619995	158.619995	
8523	2020-01-06	157.080002	159.100006	156.509995	159.029999	159.029999	
8524	2020-01-07	159.320007	159.669998	157.330002	157.580002	157.580002	
	Volume						
0	1031788800						
1	308160000						
2	133171200						
3	67766400						
4	47894400						
...	...						
8520	18369400						
8521	22622100						
8522	21116200						
8523	20813700						
8524	18017762						

[8525 rows x 7 columns]

FEATURE ENGINEERING:

Feature engineering is the process of creating new features or transforming existing ones from the raw data to improve the predictive power of the model. These features may capture patterns, trends, and relationships in the data that are not evident in the original features. In the context of our project, feature engineering involves calculating moving averages, technical indicators, and other relevant variables to enhance the model's ability to understand and forecast stock prices.

7-day moving average:

	Date	Open	High	Low	Close	Adj Close
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107
...
8520	2019-12-31	156.770004	157.770004	156.449997	157.699997	157.699997
8521	2020-01-02	158.779999	160.729996	158.330002	160.619995	160.619995
8522	2020-01-03	158.320007	159.949997	158.059998	158.619995	158.619995
8523	2020-01-06	157.080002	159.100006	156.509995	159.029999	159.029999
8524	2020-01-07	159.320007	159.669998	157.330002	157.580002	157.580002
...
8520	18369400	157.874287				
8521	22622100	158.332857				
8522	21116200	158.505713				
8523	20813700	158.741427				
8524	18017762	158.585713				

[8525 rows x 8 columns]

LSTM MODEL:

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that is designed to effectively capture and model long-term dependencies and sequential patterns in data. Here's a definition of an LSTM model:

An LSTM model is a recurrent neural network architecture that is particularly well-suited for sequential data, such as time series, natural language text, and speech. It is designed to overcome the vanishing gradient problem in traditional RNNs and can capture long-range dependencies in data. An LSTM model consists of specialized units called "memory cells" that can store and retrieve information over extended sequences, making it highly effective for tasks like time series forecasting, natural language processing, and speech recognition.

Key features of LSTM models include the ability to remember information over long time horizons, the presence of gates to control the flow of information, and the capacity to handle sequences of varying lengths. These qualities make LSTM models a popular choice for a wide range of applications in machine learning and deep learning.

In the context of stock price prediction, LSTM models can be used to capture the temporal patterns and dependencies in historical market data, allowing for the creation of predictive models that consider the intricate nature of financial time series.

MODEL BUILDING:

Model training is the phase where we feed our prepared data into a chosen predictive model. The model learns to identify patterns and relationships in the training data, allowing it to make predictions on unseen data. In our case, we continue to explore and implement models, particularly focusing on LSTM, to capture the temporal dependencies in stock price movements.

```

# Building the model
# Example: Using an LSTM model with TensorFlow/Keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Example: Using an LSTM model with specific n_timesteps and n_features
n_timesteps = 30
n_features = 5

model = Sequential()
model.add(LSTM(units=64, activation='relu', input_shape=(n_timesteps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

```

MODEL TRAINING AND EVALUATION:

Evaluation is the critical process of assessing the performance of our model. To measure its accuracy, we employ various evaluation metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and others specifically designed for time series forecasting. By evaluating the model's performance on historical data, we gauge its predictive capabilities and its potential to assist investors in making informed decisions.

```

Epoch 1/50
213/213 [=====] - 4s 7ms/step - loss: 0.0010
Epoch 2/50
213/213 [=====] - 1s 7ms/step - loss: 8.6378e-04
Epoch 3/50
213/213 [=====] - 1s 7ms/step - loss: 8.4057e-04
Epoch 4/50
213/213 [=====] - 1s 7ms/step - loss: 8.3638e-04
Epoch 5/50
213/213 [=====] - 1s 7ms/step - loss: 8.1634e-04
Epoch 6/50
213/213 [=====] - 2s 9ms/step - loss: 8.0528e-04
Epoch 7/50
213/213 [=====] - 2s 9ms/step - loss: 7.9593e-04
Epoch 8/50
213/213 [=====] - 1s 7ms/step - loss: 7.9210e-04
Epoch 9/50
213/213 [=====] - 1s 7ms/step - loss: 7.9548e-04
Epoch 10/50
213/213 [=====] - 1s 7ms/step - loss: 7.8179e-04
Epoch 11/50
213/213 [=====] - 1s 7ms/step - loss: 7.8309e-04
Epoch 12/50
213/213 [=====] - 1s 7ms/step - loss: 7.7987e-04
Epoch 13/50
213/213 [=====] - 1s 7ms/step - loss: 7.8289e-04
Epoch 14/50
213/213 [=====] - 2s 9ms/step - loss: 7.8237e-04
Epoch 15/50
213/213 [=====] - 2s 10ms/step - loss: 7.7486e-04
Epoch 16/50
213/213 [=====] - 1s 7ms/step - loss: 7.7983e-04

```

```
Epoch 17/50
213/213 [=====] - 1s 7ms/step - loss: 7.8146e-04
Epoch 18/50
213/213 [=====] - 1s 7ms/step - loss: 7.7324e-04
Epoch 19/50
213/213 [=====] - 1s 7ms/step - loss: 7.7254e-04
Epoch 20/50
213/213 [=====] - 1s 7ms/step - loss: 7.7461e-04
Epoch 21/50
213/213 [=====] - 1s 7ms/step - loss: 7.7434e-04
Epoch 22/50
213/213 [=====] - 2s 8ms/step - loss: 7.7284e-04
Epoch 23/50
213/213 [=====] - 2s 10ms/step - loss: 7.8170e-04
Epoch 24/50
213/213 [=====] - 1s 7ms/step - loss: 7.7497e-04
Epoch 25/50
213/213 [=====] - 1s 7ms/step - loss: 7.7194e-04
Epoch 26/50
213/213 [=====] - 1s 7ms/step - loss: 7.7525e-04
Epoch 27/50
213/213 [=====] - 1s 7ms/step - loss: 7.7207e-04
Epoch 28/50
213/213 [=====] - 1s 7ms/step - loss: 7.7476e-04
Epoch 29/50
213/213 [=====] - 1s 7ms/step - loss: 7.7403e-04
Epoch 30/50
213/213 [=====] - 2s 7ms/step - loss: 7.8152e-04
Epoch 31/50
213/213 [=====] - 2s 10ms/step - loss: 7.6977e-04
Epoch 32/50
213/213 [=====] - 2s 8ms/step - loss: 7.7310e-04
Epoch 33/50
213/213 [=====] - 1s 7ms/step - loss: 7.7363e-04
```

```
Epoch 35/50
213/213 [=====] - 1s 7ms/step - loss: 7.7730e-04
Epoch 36/50
213/213 [=====] - 1s 7ms/step - loss: 7.7172e-04
Epoch 37/50
213/213 [=====] - 1s 7ms/step - loss: 7.6912e-04
Epoch 38/50
213/213 [=====] - 1s 7ms/step - loss: 7.7327e-04
Epoch 39/50
213/213 [=====] - 3s 15ms/step - loss: 7.7499e-04
Epoch 40/50
213/213 [=====] - 2s 9ms/step - loss: 7.8109e-04
Epoch 41/50
213/213 [=====] - 1s 7ms/step - loss: 7.7198e-04
Epoch 42/50
213/213 [=====] - 1s 7ms/step - loss: 7.6987e-04
Epoch 43/50
213/213 [=====] - 1s 7ms/step - loss: 7.7286e-04
Epoch 44/50
213/213 [=====] - 1s 7ms/step - loss: 7.7169e-04
Epoch 45/50
213/213 [=====] - 1s 7ms/step - loss: 7.7169e-04
Epoch 46/50
213/213 [=====] - 4s 19ms/step - loss: 7.7075e-04
Epoch 47/50
213/213 [=====] - 2s 8ms/step - loss: 7.7171e-04
Epoch 48/50
213/213 [=====] - 1s 7ms/step - loss: 7.7465e-04
Epoch 49/50
213/213 [=====] - 1s 7ms/step - loss: 7.7278e-04
Epoch 50/50
213/213 [=====] - 1s 7ms/step - loss: 7.6622e-04
54/54 [=====] - 0s 3ms/step
Mean Squared Error: 1007491850475309.0
R-squared (R²): 0.3308744911630487
```

SOURCE CODE:

```
import pandas as pd
df = pd.read_csv('Modified_msft.csv')
print(df)

#feature engineering
#--adding lag values
df['Volume_lag1']=df['Volume'].shift(1)
df['Volume_lag2']=df['Volume'].shift(2)

#-- 7 day moving average
df['7-day_MA'] = df['Volume'].rolling(window=7).mean()
print(df)

# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score

# Load your time series data or sequence data
# Replace 'data' with your time series data
# Example: data = load_your_data()

# Data preprocessing
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[['Volume']])

# Define the sequence length and split the data into sequences
sequence_length = 10 # Adjust this based on your problem
X, y = [], []
for i in range(len(scaled_data) - sequence_length):
    X.append(scaled_data[i:i+sequence_length])
    y.append(scaled_data[i+sequence_length])

X, y = np.array(X), np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Reshape the input data to match the expected input shape
num_features = X_train.shape[2] # Extract the number of features from the reshaped data
```

```

# Define and compile the LSTM model
model = keras.Sequential()
model.add(keras.layers.LSTM(units=50, activation='relu', input_shape=(sequence_length,
num_features)))
model.add(keras.layers.Dense(1)) # Adjust the output layer for your problem
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32)

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

# Inverse transform predictions to the original scale if needed
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y_test)

# Evaluate the model using appropriate metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print or visualize the model's performance metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared (R²): {r2}")

```

CONCLUSION:

With feature engineering, we enrich our dataset with additional informative variables, enhancing the model's capacity to understand the complexities of stock market behaviour. Model training is a pivotal stage where our selected LSTM model learns from the historical data, laying the foundation for predictions. Lastly, the evaluation process provides valuable insights into our model's performance, enabling us to refine its architecture and optimize its predictive accuracy.