Lstm Stock Prediction with and without feature

import numpy as np # For numerical operations

import pandas as pd # For handling dataframes

import tensorflow as tf # TensorFlow for deep learning

import yfinance as yf # To fetch stock data

import matplotlib.pyplot as plt # For plotting

from sklearn.preprocessing import MinMaxScaler # To scale input data

from sklearn.metrics import mean\_squared\_error # To evaluate model performance

from tensorflow.keras.models import Sequential # For model creation

from tensorflow.keras.layers import LSTM, Dense, Dropout # LSTM and other layers

# Function to download historical stock data using Yahoo Finance API

def load\_stock\_data(ticker, start\_date, end\_date):

data = yf.download(ticker, start=start\_date, end=end\_date) # Download data

return data

# Preprocessing function using only the Close price

# Scales Close price and returns the scaled column and scaler

def preprocess\_data\_basic(data):

data.dropna(inplace=True) # Drop missing values

scaler = MinMaxScaler() # Initialize MinMaxScaler

data['Scaled\_Close'] = scaler.fit\_transform(data[['Close']]) # Scale Close column

return data[['Scaled\_Close']], scaler # Return scaled close and the scaler

# Preprocessing with additional features like SMA and price change

def preprocess\_data\_with\_features(data):

data.dropna(inplace=True)

scaler = MinMaxScaler() # Initialize scaler

data['Scaled\_Close'] = scaler.fit\_transform(data[['Close']]) # Scale Close column

data['SMA\_40'] = data['Close'].rolling(window=40).mean() # 40-day SMA

data['SMA\_100'] = data['Close'].rolling(window=100).mean() # 100-day SMA

data['Price\_Change'] = data['Close'].pct\_change() # Daily % change

data.fillna(0, inplace=True) # Replace NaNs with 0

features = ['Scaled\_Close', 'SMA\_40', 'SMA\_100', 'Price\_Change'] # Define features

return data[features], scaler

# Function to create input sequences of length `sequence\_length`

def create\_sequences(data, sequence\_length):

sequences, labels = [], []

for i in range(len(data) - sequence\_length):

sequences.append(data[i:i+sequence\_length]) # Sequence of inputs

labels.append(data[i+sequence\_length, 0]) # Next value to predict (Close)

return np.array(sequences), np.array(labels) # Return arrays

# Builds the LSTM model

def build\_lstm\_model(input\_shape):

model = Sequential([

LSTM(50, return\_sequences=True, input\_shape=input\_shape), # First LSTM layer

Dropout(0.2), # Prevent overfitting

LSTM(50, return\_sequences=False), # Second LSTM layer

Dropout(0.2),

Dense(25, activation='relu'), # Dense layer with ReLU

Dense(1) # Output layer

])

model.compile(optimizer='adam', loss='mse') # Compile model

return model

# Train and evaluate LSTM model

def train\_and\_evaluate\_lstm(ticker, start\_date, end\_date, sequence\_length=50, use\_features=False):

data = load\_stock\_data(ticker, start\_date, end\_date) # Load stock data

if use\_features:

data, scaler = preprocess\_data\_with\_features(data) # With features

else:

data, scaler = preprocess\_data\_basic(data) # Only Close

X, y = create\_sequences(data.values, sequence\_length) # Create sequences

split = int(0.8 \* len(X)) # 80% train, 20% test

X\_train, y\_train = X[:split], y[:split]

X\_test, y\_test = X[split:], y[split:]

model = build\_lstm\_model((X\_train.shape[1], X\_train.shape[2])) # Build model

model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test), verbose=1) # Train model

predictions = model.predict(X\_test) # Predict on test data

mse = mean\_squared\_error(y\_test, predictions) # MSE

rmse = np.sqrt(mse) # RMSE

print(f'MSE: {mse}, RMSE: {rmse}') # Print results

return model, scaler, mse, rmse

# Main execution code

start\_date = '2020-01-01'

end\_date = '2025-01-01'

ticker = 'TATAMOTORS.BO' # Tata Motors stock ticker

# Train without feature engineering

print("Training LSTM without feature engineering...")

lstm\_basic, scaler\_basic, mse\_basic, rmse\_basic = train\_and\_evaluate\_lstm(

ticker, start\_date, end\_date, use\_features=False)

# Train with feature engineering

print("Training LSTM with feature engineering...")

lstm\_with\_features, scaler\_features, mse\_features, rmse\_features = train\_and\_evaluate\_lstm(

ticker, start\_date, end\_date, use\_features=True)

# Print final RMSE comparisons

print(f"LSTM without Features -> MSE: {mse\_basic}, RMSE: {rmse\_basic}")

print(f"LSTM with Features -> MSE: {mse\_features}, RMSE: {rmse\_features}")