**Lstm Comparative Modeling**

import numpy as np # For numerical computations

import pandas as pd # For handling tabular data

import tensorflow as tf # For deep learning operations

import yfinance as yf # For downloading stock data

import matplotlib.pyplot as plt # For visualizing results

from sklearn.preprocessing import MinMaxScaler # For normalizing features

from sklearn.metrics import mean\_squared\_error # For evaluating model performance

from tensorflow.keras.models import Sequential # To build sequential model

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

# Load historical stock data from Yahoo Finance

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

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

return data

# Basic preprocessing: scales only the Close price

def preprocess\_data\_basic(data):

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

scaler = MinMaxScaler() # Initialize scaler

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

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

# Preprocessing with feature engineering: SMA and Price Change

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

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

scaler = MinMaxScaler() # Initialize MinMaxScaler

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 return percentage

data.fillna(0, inplace=True) # Fill NaNs created by rolling and pct\_change

features = ['Scaled\_Close', 'SMA\_40', 'SMA\_100', 'Price\_Change'] # Select feature columns

return data[features], scaler

# Convert time series data to LSTM-compatible sequences

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

sequences, labels = [], [] # Initialize lists

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

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

labels.append(data[i+sequence\_length, 0]) # Target is next value of Scaled\_Close

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

# Define LSTM model architecture

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

model = Sequential([

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

Dropout(0.2), # Dropout to prevent overfitting

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

Dropout(0.2),

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

Dense(1) # Output prediction

])

model.compile(optimizer='adam', loss='mse') # Compile with mean squared error

return model

# Train and evaluate the 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) # Download stock data

# Apply the appropriate preprocessing

if use\_features:

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

else:

data, scaler = preprocess\_data\_basic(data) # Basic scaling

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

split = int(0.8 \* len(X)) # 80% training, 20% testing split

X\_train, y\_train = X[:split], y[:split] # Training data

X\_test, y\_test = X[split:], y[split:] # Testing data

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

model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test), verbose=0) # Train the model

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

mse = mean\_squared\_error(y\_test, predictions) # Compute Mean Squared Error

rmse = np.sqrt(mse) # Compute Root Mean Squared Error

return model, scaler, mse, rmse, y\_test, predictions # Return results

# Define stock symbol and dates

start\_date = '2020-01-01'

end\_date = '2025-01-01'

ticker = 'TATAMOTORS.BO'

# Train basic LSTM model (no engineered features)

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

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

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

# Train LSTM model with engineered features

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

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

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

# Print and compare RMSE values

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

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

# Plot predicted vs actual values (no features)

plt.figure(figsize=(12, 6))

plt.plot(y\_test\_basic, label="Actual Prices", linestyle="dashed")

plt.plot(predictions\_basic, label="LSTM Predictions (No Features)")

plt.title("LSTM Predictions Without Feature Engineering")

plt.xlabel("Time")

plt.ylabel("Stock Price")

plt.legend()

plt.show()

# Plot predicted vs actual values (with features)

plt.figure(figsize=(12, 6))

plt.plot(y\_test\_features, label="Actual Prices", linestyle="dashed")

plt.plot(predictions\_features, label="LSTM Predictions (With Features)")

plt.title("LSTM Predictions With Feature Engineering")

plt.xlabel("Time")

plt.ylabel("Stock Price")

plt.legend()

plt.show()