import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, r2\_score

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

def prepare\_lstm\_data(data, time\_step=60):

X, y = [], []

for i in range(len(data) - time\_step):

X.append(data[i:(i + time\_step)])

y.append(data[i + time\_step])

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

def train\_and\_evaluate\_model(company\_name, df):

company\_data = df[df['Company'] == company\_name].sort\_values('Date')

if len(company\_data) <= 60:

print(f"Not enough data for {company\_name}.")

return

close\_prices = company\_data['Close'].values.reshape(-1, 1)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(close\_prices)

time\_step = 60

X, y = prepare\_lstm\_data(scaled\_data, time\_step)

X = X.reshape(X.shape[0], X.shape[1], 1)

split\_idx = int(len(X) \* 0.8)

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

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

model = Sequential([

Bidirectional(LSTM(64, return\_sequences=True, input\_shape=(time\_step, 1))),

Dropout(0.2),

Bidirectional(LSTM(64, return\_sequences=True)),

Dropout(0.2),

Bidirectional(LSTM(32)),

Dropout(0.2),

Dense(25, activation='relu'),

Dense(1)

])

optimizer = Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss='mean\_squared\_error')

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=5, min\_lr=1e-6)

model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test),

verbose=1, callbacks=[early\_stopping, reduce\_lr])

test\_predict = model.predict(X\_test)

test\_predict = scaler.inverse\_transform(test\_predict)

y\_test\_actual = scaler.inverse\_transform(y\_test.reshape(-1, 1))

mse = mean\_squared\_error(y\_test\_actual, test\_predict)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test\_actual, test\_predict)

accuracy = (1 - (rmse / np.mean(y\_test\_actual))) \* 100

# Adjust figure size for better visibility

plt.figure(figsize=(10, 5))

plt.plot(company\_data['Date'][-len(y\_test\_actual):], y\_test\_actual, label='Actual Prices', color='blue')

plt.plot(company\_data['Date'][-len(y\_test\_actual):], test\_predict, label='Predicted Prices', color='red')

plt.title(f'{company\_name.upper()} Stock Price Prediction', fontsize=14)

plt.xlabel('Date', fontsize=12)

plt.ylabel('Stock Price', fontsize=12)

plt.legend(fontsize=12)

# Save image for better visibility

plt.savefig(f"{company\_name}\_prediction.png", dpi=300, bbox\_inches='tight')

plt.show()

print(f'{company\_name.upper()} - MSE: {mse:.4f}, RMSE: {rmse:.4f}, R2 Score: {r2:.4f}, Accuracy: {accuracy:.2f}%')

# Load and preprocess data

df = pd.read\_csv(r"C:\Users\hp\Desktop\EV stocks\evStock.csv")

df['Date'] = pd.to\_datetime(df['Date'], errors='coerce')

numeric\_columns = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']

df[numeric\_columns] = df[numeric\_columns].apply(pd.to\_numeric, errors='coerce')

df.dropna(inplace=True)

# Train model for each company

unique\_companies = df['Company'].unique()

for company in unique\_companies:

train\_and\_evaluate\_model(company, df)