import yfinance as yf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error

import datetime

# Function to fetch data and train model

def stock\_price\_prediction(stock\_symbol, years):

    # Get today's date

    end\_date = datetime.datetime.now().strftime('%Y-%m-%d')

    # Calculate the start date based on the number of years

    start\_date = (datetime.datetime.now() - datetime.timedelta(days=365 \* years)).strftime('%Y-%m-%d')

    # Fetch stock data

    stock\_data = yf.download(stock\_symbol, start=start\_date, end=end\_date)

    # Preprocess data (use Close price for prediction)

    stock\_data = stock\_data[['Close']]

    # Feature Engineering: Using previous 5 days to predict the 6th day

    stock\_data['Prediction'] = stock\_data['Close'].shift(-1)

    # Drop the last row since it has no target value

    stock\_data.dropna(inplace=True)

    # Split the data into features (X) and labels (y)

    X = stock\_data[['Close']]

    y = stock\_data['Prediction']

    # Train-test split (80% train, 20% test)

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

    # Initialize the model

    model = RandomForestRegressor(n\_estimators=100, random\_state=42)

    # Train the model

    model.fit(X\_train, y\_train)

    # Make predictions

    predictions = model.predict(X\_test)

    # Calculate Mean Absolute Error (MAE) as a metric

    mae = mean\_absolute\_error(y\_test, predictions)

    print(f'Mean Absolute Error: {mae}')

    # Predict the next month's stock price (one step ahead)

    last\_close = stock\_data['Close'].iloc[-1]

    future\_prediction = model.predict([[last\_close]])

    print(f'Predicted next month stock price: {future\_prediction[0]:.2f}')

    # Plotting the results

    plt.figure(figsize=(10, 6))

    plt.plot(stock\_data.index, stock\_data['Close'], label='Historical Stock Prices')

    plt.plot(stock\_data.index[-len(y\_test):], predictions, label='Predicted Prices', color='orange')

    plt.title(f'{stock\_symbol} Stock Price Prediction')

    plt.xlabel('Date')

    plt.ylabel('Stock Price')

    plt.legend()

    plt.grid(True)

    plt.show()

    # Show the first 15 entries of the data

    print("First 15 rows of the stock data:")

    print(stock\_data.head(15))

    # Save the historical data and predictions to an Excel file

    output\_file = f'{stock\_symbol}\_stock\_data.xlsx'

    # Create a DataFrame for the results

    result\_data = stock\_data.copy()

    result\_data['Predicted\_Price'] = np.concatenate([predictions, [np.nan]])  # Adding predictions to the historical data

    result\_data['Future\_Prediction'] = np.nan  # Adding a column for future prediction

    # Add the one-month prediction to the last row

    result\_data.iloc[-1, result\_data.columns.get\_loc('Future\_Prediction')] = future\_prediction[0]

    # Save to Excel with separate sheets for historical data and predictions

    with pd.ExcelWriter(output\_file, engine='openpyxl') as writer:

        result\_data.to\_excel(writer, sheet\_name='Stock Data with Prediction', index=True)

        stock\_data.to\_excel(writer, sheet\_name='Original Stock Data', index=True)

    print(f'Historical stock data with predictions saved to {output\_file}')

    return future\_prediction[0]

# Example usage

stock\_symbol = input("Enter the stock symbol (e.g., 'AAPL', 'GOOGL'): ")

years = int(input("Enter the number of years of data: "))

predicted\_price = stock\_price\_prediction(stock\_symbol, years)

print(f'Predicted stock price for the next month: {predicted\_price:.2f}')