**Name: Rushikesh Ratnakar Waghmare**

**Class: TE-C Roll No: 52**

**Sub: AI Miniproject**

**Title: Stock Price Prediction**

**Code:**

import yfinance as yf

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

import matplotlib.pyplot as plt

"""

    //  Function Name : download\_data

    //  Description :  Downloads historical stock data for the given ticker.

    //  Input :        ticker (str), start (str), end (str)

    //  Output :       DataFrame containing stock data

    //  Author :       Rushikesh Ratnakar Waghmare

    //  Date :         05/10/2024

"""

def download\_data(ticker: str, start: str, end: str) -> pd.DataFrame:

    data = yf.download(ticker, start=start, end=end)

    return data

"""

    //  Function Name : preprocess\_data

    //  Description :  Preprocess the stock data by filling missing values and normalizing.

    //  Input :        data (DataFrame)

    //  Output :       DataFrame after preprocessing

    //  Author :       Rushikesh Ratnakar Waghmare

    //  Date :         05/10/2024

"""

def preprocess\_data(data: pd.DataFrame) -> pd.DataFrame:

    data.fillna(method='ffill', inplace=True)

    # Feature normalization

    scaler = MinMaxScaler()

    scaled\_data = scaler.fit\_transform(data[['Open', 'High', 'Low', 'Close', 'Volume']])

    data[['Open', 'High', 'Low', 'Close', 'Volume']] = scaled\_data

    return data

"""

    //  Function Name : feature\_engineering

    //  Description :  Adds technical indicators as features to the stock data.

    //  Input :        data (DataFrame)

    //  Output :       DataFrame with additional features

    //  Author :       Rushikesh Ratnakar Waghmare

    //  Date :         05/10/2024

"""

def feature\_engineering(data: pd.DataFrame) -> pd.DataFrame:

    # Moving Average as a feature

    data['Moving\_Average'] = data['Close'].rolling(window=20).mean()

    # Relative Strength Index (RSI)

    delta = data['Close'].diff(1)

    gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()

    loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()

    rs = gain / loss

    data['RSI'] = 100 - (100 / (1 + rs))

    data.dropna(inplace=True)

    return data

"""

    //  Function Name : train\_model

    //  Description :  Trains a Random Forest model using the provided features and target.

    //  Input :        X (DataFrame), y (Series)

    //  Output :       Trained model

    //  Author :       Rushikesh Ratnakar Waghmare

    //  Date :         05/10/2024

"""

def train\_model(X: pd.DataFrame, y: pd.Series):

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

    model.fit(X, y)

    return model

"""

    //  Function Name : merge\_sort

    //  Description :  Implements the merge sort algorithm to sort an array.

    //  Input :        arr (list)

    //  Output :       None (sorted in place)

    //  Author :       Rushikesh Ratnakar Waghmare

    //  Date :         05/10/2024

"""

def merge\_sort(arr):

    if len(arr) > 1:

        mid = len(arr) // 2

        L = arr[:mid]

        R = arr[mid:]

        merge\_sort(L)

        merge\_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):

            if L[i] < R[j]:

                arr[k] = L[i]

                i += 1

            else:

                arr[k] = R[j]

                j += 1

            k += 1

        while i < len(L):

            arr[k] = L[i]

            i += 1

            k += 1

        while j < len(R):

            arr[k] = R[j]

            j += 1

            k += 1

"""

    //  Function Name : evaluate\_model

    //  Description :  Evaluates the trained model using Mean Squared Error and Mean Absolute Error.

    //  Input :        model, X\_test (DataFrame), y\_test (Series)

    //  Output :       None (prints evaluation metrics and shows an enhanced graph)

    //  Author :       Rushikesh Ratnakar Waghmare

    //  Date :         05/10/2024

"""

def evaluate\_model(model, X\_test, y\_test):

    y\_pred = model.predict(X\_test)

    mse = mean\_squared\_error(y\_test, y\_pred)

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

    print(f'Mean Squared Error: {mse}')

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

    # Set up a cleaner visualization

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8), sharex=True)

    # Plot actual prices

    ax1.plot(y\_test.values, color='blue', label='Actual Prices', lw=2)

    ax1.set\_title('Actual Stock Prices', fontsize=14)

    ax1.set\_ylabel('Price')

    ax1.grid(True)

    # Plot predicted prices

    ax2.plot(y\_pred, color='green', label='Predicted Prices', lw=2)

    ax2.set\_title('Predicted Stock Prices', fontsize=14)

    ax2.set\_xlabel('Time')

    ax2.set\_ylabel('Price')

    ax2.grid(True)

    # Display both subplots

    plt.tight\_layout()

    plt.show()

"""

    //  Function Name : main

    //  Description :  Entry point of the application. Calls other functions to execute the stock prediction workflow.

    //  Input :        None

    //  Output :       None

    //  Author :       Rushikesh Ratnakar Waghmare

    //  Date :         05/10/2024

"""

def main():

    # Step 1: Download data

    ticker = 'AAPL'

    data = download\_data(ticker, start='2020-01-01', end='2024-01-01')

    # Step 2: Preprocess data

    data = preprocess\_data(data)

    # Step 3: Feature engineering

    data = feature\_engineering(data)

    # Step 4: Prepare data for model training

    X = data[['Open', 'High', 'Low', 'Volume', 'RSI', 'Moving\_Average']]

    y = data['Close']

    # Split dataset

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    # Step 5: Train model

    model = train\_model(X\_train, y\_train)

    # Step 6: Evaluate model

    evaluate\_model(model, X\_test, y\_test)

    # Step 7: Sort data for demonstration using merge sort

    sample\_data = [3, 6, 1, 5, 2, 4]

    merge\_sort(sample\_data)

    print("Sorted data:", sample\_data)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**



