Individual Project Report

Jack Clark

011 : Team 04

ENGR 133: Transforming Ideas To Innovation, EPICS

Professor John Cole

12/11/2024

1. Project Introduction

This Python application analyzes the stock market and provides investors with useful information for both immediate and long-term decision-making. The project is prompted by the growing need for data-based, user-friendly solutions that make financial markets easier to understand. The tool is helpful for both long-term investors seeking sustainable growth and day traders seeking short-term forecasts since it integrates technical indicators, fundamental measurements, and economic elements to allow users to study trends over a variety of timeframes. This project connects to my dream project idea of developing predictive models for currencies and bonds, which leverage time series data to uncover relationships between economic indicators and asset prices. Both utilize live data to make predictions for future market prices.

2. Project Overview of Inputs and Outputs

Inputs:

- Start date for the analysis period
- End date for the analysis period
- Number of stocks the user wishes to analyze
- Stock ticker symbols
- The current month's inflation rate

Outputs:

- Graph of stock prices, RSI, and EMA based on the start and end date analysis period
- Combined analysis plots
- Inflation impact analysis
- Calculations for each stock including
    - RSI and EMA values
    - Current price
    - Performance metrics
        - Total Return, Volatility, Shape Ratio
- It also includes short-term and long-term recommendations for the user

3. User-defined Functions

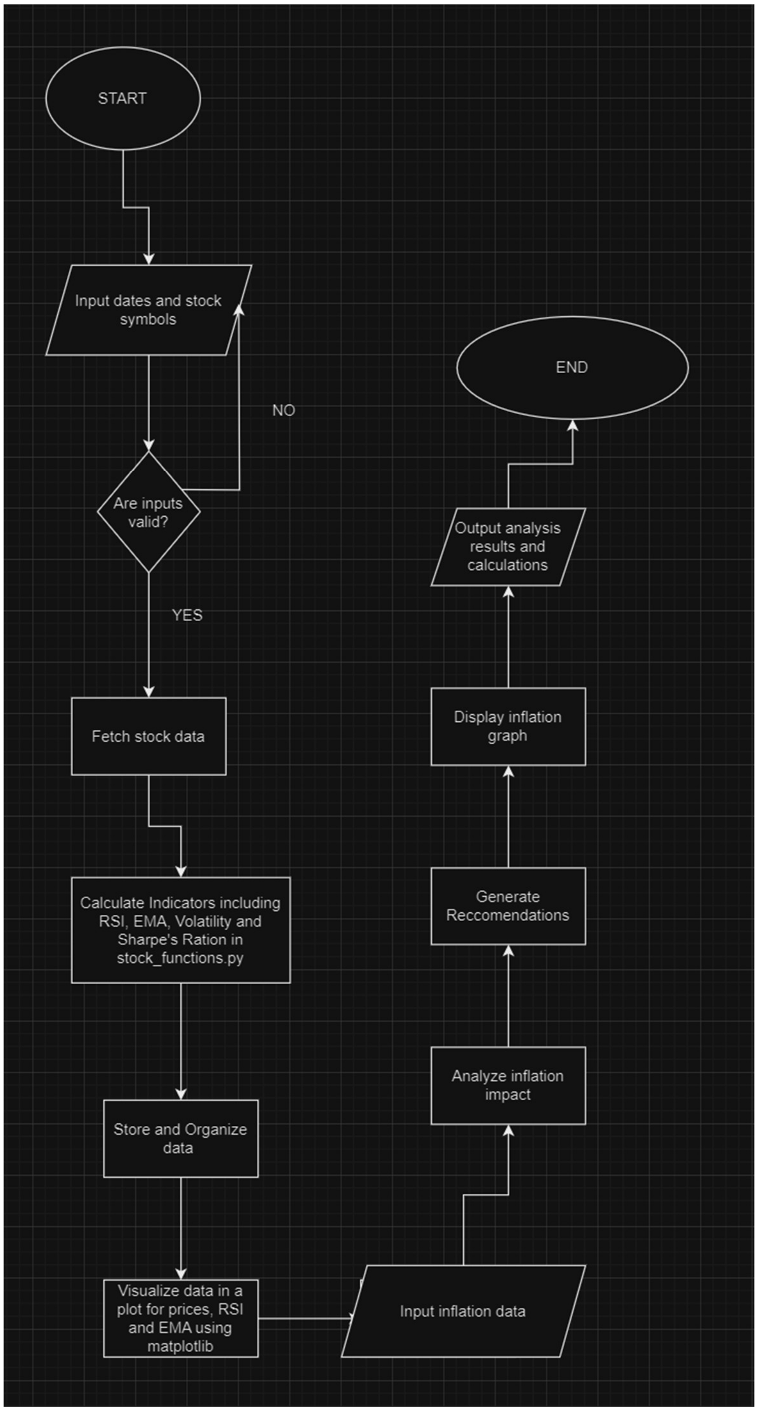
validate-dates(): Ensures users input the dates in the proper format
get_stock_data(): fetches historical stock data using finance
calculate_ema(): computes the exponential moving average
plot_stock_prices(): visualizes the stock price data over the given range
plot_rsi(): visualizes whether the stock is overbought or oversold over the given range
plot_ema(): visualizes the exponential moving average of the data
analyze_ionflation_impact(): assesses the effect of inflation on stock returns

get_recommendation(): provides investment recommendations based on metrics calculated in other user-defined functions
calculate_rsi(): implemented in another file, this takes data from get_stock_data() and calculates if it was overbought or oversold over the given range
calculate_metrics(): also implemented in stock_function.py file, this calculates the total returns, volatility, and Sharpe ratio over the given time.

User manual

1. Run the program
2. Enter the start and end dates for analysis when prompted. Use the format YYYY-MM-DD.
3. Input the number of stocks you want to analyze
4. Enter the ticker symbols for each stock
5. You should see graphs of the stock data, RSI, and EMA
6. Provide the current month’s inflation rate after closing the graphs.
7. It will then plot the inflation data with two different lines of nominal cumulative returns and real cumulative returns.
8. Read the recommendations it gives you for the short and long-term and the calculated values after closing the inflation graph.

Flowchart


START
Input dates and stock symbols
Are inputs valid?
NO
YES
Fetch stock data
Calculate Indicators including RSI, EMA, Volatility and Sharpe's Ration in stock_functions.py
Store and Organize data
Visualize data in a plot for prices, RSI and EMA using matplotlib
Input inflation data
Analyze inflation impact
Generate Reccomendations
Display inflation graph
Output analysis results and calculations
END

Code

final_Project_file.py

```
"""
Course Number: ENGR 13300
Semester: e.g. Fall 2024

Description:
    This code implements a comprehensive stock market analysis tool using python libraries to fetch,
process, visualize and predict financial trends.

Assignment Information:
    Assignment:     Final Project
    Team ID:        011 - 04 (e.g. LC1 - 01; for section LC1, team 01)
    Author:         Jack Clark, login@purdue.edu
    Date:           12/5/2024

Contributors:
    Name, login@purdue [repeat for each]

    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
```

```
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.

Academic Integrity Statement:
    I have not used source code obtained from any unauthorized
    source, either modified or unmodified; nor have I provided
    another student access to my code.  The project I am
    submitting is my own original work.
"""


""" Write any import statements here (and delete this line)."""
import yfinance as yf  # Library for fetching financial data
import pandas as pd  # Library for data manipulation and analysis
import numpy as np
import matplotlib.pyplot as plt
from stock_functions import calculate_rsi, calculate_metrics


def validate_dates():
    while True:
        start_date = input("Enter the start date for analysis (YYYY-MM-DD): ")
        end_date = input("Enter the end date for analysis (YYYY-MM-DD): ")
        if len(start_date) == 10 and len(end_date) == 10:
            return start_date, end_date
        else:
            print("Error: Dates must be in the format YYYY-MM-DD (exactly 10 characters).")


def get_stock_data(ticker, start_date, end_date):
    stock = yf.Ticker(ticker)  # Using yfinance to create a Ticker object
    data = stock.history(start=start_date, end=end_date)  # Fetching historical data as a pandas DataFrame
    return data


def calculate_ema(prices, period=20):
    ema = [0] * len(prices)
    ema[period-1] = sum(prices[:period]) / period
    multiplier = 2 / (period + 1)
    ##############First for loop########################
    for i in range(period, len(prices)):
        ema[i] = (prices[i] - ema[i-1]) * multiplier + ema[i-1]
    return ema


def plot_stock_prices(stock_data):
    plt.figure(figsize=(12, 6))
    for ticker, data in stock_data.items():
        if ticker != '^GSPC':
            plt.plot(data.index, data['Close'], label=ticker)
    plt.title('Stock Prices')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.legend()
    plt.grid(True)
    plt.show()

def plot_rsi(stock_data):
    plt.figure(figsize=(12, 6))
    for ticker, data in stock_data.items():
        if ticker != '^GSPC':
            plt.plot(data.index, data['RSI'], label=f'{ticker} RSI')  # Using pandas DataFrame index and 'RSI' column
    plt.axhline(y=70, color='r', linestyle='--')
    plt.axhline(y=30, color='g', linestyle='--')
    plt.title('Relative Strength Index (RSI)')
    plt.xlabel('Date')
    plt.ylabel('RSI')
    plt.legend()
    plt.grid(True)
    plt.show()


def plot_ema(stock_data):
    plt.figure(figsize=(12, 6))
    for ticker, data in stock_data.items():
        if ticker != '^GSPC':
            plt.plot(data.index, data['EMA'], label=f'{ticker} EMA')  # Using pandas DataFrame index and
'EMA' column
    plt.title('Exponential Moving Average (EMA)')
    plt.xlabel('Date')
    plt.ylabel('EMA')
    plt.legend()
    plt.grid(True)
    plt.show()


def plot_combined_analysis(stock_data):
    fig, axs = plt.subplots(3, 1, figsize=(12, 18), sharex=True)
    for ticker, data in stock_data.items():
        if ticker != '^GSPC':
            axs[0].plot(data.index, data['Close'], label=ticker)
    axs[0].set_title('Stock Prices')
    axs[0].legend()
    axs[0].grid(True)
    for ticker, data in stock_data.items():
        if ticker != '^GSPC':
            axs[1].plot(data.index, data['RSI'], label=f'{ticker} RSI')  # Using pandas DataFrame index and 'RSI' column
    axs[1].axhline(y=70, color='r', linestyle='--')
    axs[1].axhline(y=30, color='g', linestyle='--')
    axs[1].set_title('Relative Strength Index (RSI)')
    axs[1].legend()
    axs[1].grid(True)

    for ticker, data in stock_data.items():
        if ticker != '^GSPC':
            axs[2].plot(data.index, data['EMA'], label=f'{ticker} EMA')  #
    axs[2].set_title('Exponential Moving Average (EMA)')
    axs[2].legend()
    axs[2].grid(True)

    plt.tight_layout()
    plt.show()

def analyze_inflation_impact(stock_df, inflation_rate, ticker):
    stock_returns = stock_df['Close'].pct_change()  # Using pandas pct_change() method
    real_returns = stock_returns - inflation_rate/100/12
    cumulative_real_returns = (1 + real_returns).cumprod() - 1  # Using pandas cumprod() method

    plt.figure(figsize=(12, 6))
    plt.plot(stock_df.index, cumulative_real_returns, label='Real Cumulative Returns')
    plt.plot(stock_df.index, stock_df['Close']/stock_df['Close'].iloc[0] - 1, label='Nominal Cumulative
Returns')
    plt.title(f'Impact of Inflation on {ticker} Returns')
    plt.xlabel('Date')
    plt.ylabel('Cumulative Returns')
    plt.legend()
    plt.grid(True)
    plt.show()

def get_recommendation(metrics, rsi, price, ema):
    short_term = "Hold"
    long_term = "Hold"

    if rsi > 70:
        short_term = "Sell"
    elif rsi < 30:
        short_term = "Buy"
    elif price > ema:
        short_term = "Buy"
    elif price < ema:
        short_term = "Sell"

    if metrics['Total_Return'] > 0.2 and metrics['Sharpe_Ratio'] > 0.5:
        long_term = "Buy"
    elif metrics['Total_Return'] < -0.1 or metrics['Sharpe_Ratio'] < 0:
        long_term = "Sell"

    return short_term, long_term

def get_valid_number(prompt):
    #####################First while loop###################
    while True:
        try:
            value = float(input(prompt))
            return value
        except ValueError:
            print("Error: Please enter a valid number.")

def main():
    start_date, end_date = validate_dates()
    num_stocks = int(get_valid_number("Enter the number of stocks you want to analyze: "))
    tickers = []
    for i in range(num_stocks):
        ticker = input(f"Enter stock ticker symbol {i+1}: ").upper()
        tickers.append(ticker)

    if '^GSPC' not in tickers:
        tickers.append('^GSPC')

    stock_data = {}
    metrics = {}

    for ticker in tickers:
        data = get_stock_data(ticker, start_date, end_date)
        data['RSI'] = calculate_rsi(data['Close'])
        data['EMA'] = calculate_ema(data['Close'])
        stock_data[ticker] = data

        total_return, volatility, sharpe_ratio = calculate_metrics(data)
        metrics[ticker] = {
            'Total_Return': total_return,
            'Volatility': volatility,
            'Sharpe_Ratio': sharpe_ratio
        }
    plot_stock_prices(stock_data)
    plot_rsi(stock_data)
    plot_ema(stock_data)
    plot_combined_analysis(stock_data)

    current_inflation = get_valid_number("Enter the current month's inflation rate (%): ")

    for ticker in tickers:
        if ticker != '^GSPC':
            analyze_inflation_impact(stock_data[ticker], current_inflation, ticker)

    print("\nStock Analysis Results:")

    for ticker in tickers:
        if ticker == '^GSPC':
            continue

        data = stock_data[ticker]
        last_close = data['Close'].iloc[-1]  # Using pandas iloc for indexing
        last_rsi = data['RSI'].iloc[-1]
        last_ema = data['EMA'].iloc[-1]
        short_term, long_term = get_recommendation(metrics[ticker], last_rsi, last_close, last_ema)


        print(f"\n{ticker}:")
        print(f"Current Price: ${last_close:.2f}")
        print(f"RSI: {last_rsi:.2f}")
        print(f"EMA: ${last_ema:.2f}")
        print(f"Total Return: {metrics[ticker]['Total_Return']:.2%}")
        print(f"Volatility: {metrics[ticker]['Volatility']:.2%}")
        print(f"Sharpe Ratio: {metrics[ticker]['Sharpe_Ratio']:.2f}")
        print(f"Short-term Recommendation: {short_term}")
        print(f"Long-term Recommendation: {long_term}")


    if current_inflation > 2:
        print(f"High inflation ({current_inflation}%). Consider inflation-protected assets such as real estate, bonds, or commodities.")
    else:
        print(f"Low inflation ({current_inflation}%). Stocks may outperform.")


if __name__ == "__main__":
    main()
```

stock_functions.py

```

"""
```

```
Course Number: ENGR 13300

Semester: e.g. Fall 2024


Description:

    This code calculates RSI sharpe’s ratio, volatility and other metrics.


Assignment Information:

    Assignment:     Python Final Project

    Team ID:        011 - 04 (e.g. LC1 - 01; for section LC1, team 01)

    Author:         Jack Clark, clar1037@purdue.edu

    Date:           12/05/2024


Contributors:

    Name, login@purdue [repeat for each]


    My contributor(s) helped me:

    [ ] understand the assignment expectations without

        telling me how they will approach it.

    [ ] understand different ways to think about a solution

        without helping me plan my solution.

    [ ] think through the meaning of a specific error or

        bug present in my code without looking at my code.

    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.

Academic Integrity Statement:
    I have not used source code obtained from any unauthorized
    source, either modified or unmodified; nor have I provided
    another student access to my code.  The project I am
    submitting is my own original work.
"""

import numpy as np

def calculate_rsi(prices, period=14):
    #By default, RSI looks back 14 days
    delta = [0] * len(prices)
    gain = [0] * len(prices)
    loss = [0] * len(prices)
    ######################FIRST NESTED LOOP######################
    for i in range(1, len(prices)):
        delta[i] = prices[i] - prices[i-1]
        if delta[i] > 0:
            gain[i] = delta[i]
        else:
            loss[i] = -delta[i]
    avg_gain = [0] * len(prices)

    avg_loss = [0] * len(prices)

    avg_gain[period] = sum(gain[1:period+1]) / period

    avg_loss[period] = sum(loss[1:period+1]) / period

    i = period + 1

    while i < len(prices):

        avg_gain[i] = (avg_gain[i-1] * (period-1) + gain[i]) / period

        avg_loss[i] = (avg_loss[i-1] * (period-1) + loss[i]) / period

        i += 1

    rs = [0] * len(prices)

    rsi = [0] * len(prices)

    if period < len(prices):

        for i in range(period, len(prices)):

            if avg_loss[i] != 0:

                rs[i] = avg_gain[i] / avg_loss[i]

            else:

                rs[i] = 100

            rsi[i] = 100 - (100 / (1 + rs[i]))

    return rsi


def calculate_metrics(data):

    total_return = (data['Close'][-1] / data['Close'][0]) - 1

    volatility = data['Close'].pct_change().std() * np.sqrt(252)
    sharpe_ratio = total_return / volatility

    return total_return, volatility, sharpe_ratio
```