ASSIGNMENT 20 - For Computer Wizards

Luca D'Ambrosio, Filippo De Min
3121995, 3143769
Bocconi University

Contents

1 Introduction

Gaining access to readable and well presented financial data has always been a harrowing task, mainly due to the technical aspects of the task and the physical cost that has to be paid in order to access such platforms (i.e. Bloomberg, Reuters etc.). The aim of this project is to construct an accessible alternative that allows the user to retrieve and analyze substantial amounts of financial data. The user will be able to retrieve price data for a stock, and to access the company's balance sheet as well as its financial statements. Moreover, it will be possible to form portfolios composed of a maximum of ten different stocks, whose weights will have to be specified by the user. After the portfolio is formed, the time trend and relevant key metrics will be displayed in the spreadsheet. The process can be summarized in six steps:

- **Selection**: The user either selects a single stock or a basket of up to 10 stocks to form a portfolio.
- **Time span**: The user selects the time frame over which they would like to conduct their analysis.
- **VBA macro**: Through a VBA script we trigger a Python script from terminal which fetches the financial data from Yahoo Finance.
- **Download**: Through a Python script, we connect to Yahoo finance and download the relevant data.
- **Data Manipulation**: We compute the returns over time and store them, together with the balance sheet and financial statements, into separate CSV files in the working directory. If the user wants to build a portfolio we download price data for the selected stocks and use the weights to compute portfolio returns over time.
- **Upload**: the data is then uploaded to the Excel workbook in different worksheets.
- **Display**: The dashboard automatically updates displaying the results.

  As a bonus step for the portfolio results, we add the return data of the S&P500 as a benchmark.

  We believe that our dashboard is intuitive and easily usable, although some limitations will be discussed in the following sections.

2 Code

Our project requires to work simultaneously in Visual Basic and Python enviroments, creating a connection between the two through Windows PowerShell.

2.1 VBA

We developed two VBA macros: *Main*, which retrieves historical data for a single stock, and *portfolio_creation*, which fetches historical data for a user-defined portfolio of up to 10 securities. Both macros are triggered by pressing the **Get Data** button. The file is ready for execution once the user specifies the path to their Python executable and adjusts the working directory in the macros that invoke the Python script.

Each macro is composed of several underlying macros that can be grouped in 2 categories:

1. Macros that import csv files in a worksheet of the current file. They use the *QueryTables.Add* method to load the CSV data into the sheet, specifying various parameters for how the file should be parsed. These include delimiters (comma and tab), the starting row for the data, and the text qualifier.

2. Macros that start a python script located in the same folder. These locate the inputs given in specific cells of the current worksheet, apply some transformation and execute the python file with some given inputs.

```
    period = Range("C12").Value
    tickers = Join(Application.Transpose(Range("D2:
        D11").Value), "/")
    weights = Join(Application.Transpose(Range("E2:
        E11").Value), "/")
```

The cell above shows how a series of cells are being transformed into a single string separated by "/".

```
    objShell.Run PythonExePath & " " & fileName & "
        --period=" & Chr(34) & period & Chr(34) & "
         --tickers=" & Chr(34) & tickers & Chr(34)
        & " --weights=" & Chr(34) & weights & Chr
        (34)
```

We used the code above to parse the excel inputs into Python.

Finally, we included a cooldown of 5 and 10 seconds to allow the script to execute and load the csv files in the current folder.

2.2 Python

We coded 2 distinct Python files:

1. **get_data.py** is triggered by the macros RunPythonScript. This file takes as input a ticker and a period and through yahoo finance api creates 4 different csv files in the same folder:

   (a) Stock prices and returns over the desired period.

   (b) Financial statements of the stock.

   (c) Balance sheet of the stock.

   (d) SP500 prices and returns over the same period (to create a comparison with the given stock)

   To compute returns, we built the function **get_stock_returns**:

```
def get_stock_returns(ticker: str, period: str) -> pd
    .DataFrame:
    """
    Fetch stock data for the given ticker and period,
        calculate returns, and reorder columns.

    Args:
    ticker (str): Stock ticker symbol.
    period (str): Time period for the stock data.

    Returns:
    pd.DataFrame: Processed DataFrame with returns.
    """
    # Fetch stock data
    stock = yf.Ticker(ticker)
    data = stock.history(period=period)

    # Reset the index to flatten the MultiIndex
    data = data.reset_index()

    # Convert 'Date' to datetime and set it as the
        index
    data['Date'] = pd.to_datetime(data['Date']).dt.
        strftime('%Y-%m-%d')
    data = data.set_index('Date')

    # Calculate daily returns based on the 'Close'
        column
    data['Returns'] = data['Close'].pct_change()

    # Drop rows with NaN values (e.g., first row
        after calculating returns)
    data = data.dropna(subset=['Returns'])

    # Reorder columns so that 'Return' is the first
        column
    columns = ['Returns'] + [col for col in data.
        columns if col != 'Returns']
    data = data[columns]

    return data
```

2. **get_data_portfolio.py** is triggered by the macro RunPythonScript2 and takes as input a time-period, 10 stock' tickers and 10 stock' weights. The file outputs prices and returns data of that portfolio as a csv as well as the SP500 prices and returns over the same period.

   ```
   Inputs example for the Python script:
   - period: "1y"
   - tickers: "CARR/GOOGL/KO/HPE/MCD/GS/V/CMG/NKE/A"
   - weights: "0/0,1/0,05/0,15/0,4/0,1/0,2/0/0/0"
   ```

   We chose "/" as a separator as "," was ambiguous when dealing with decimals. With python strings' built-in methods *split* and *replace* we were able to transform the strings' inputs into lists. We then iterate over the list of tickers and use our previously-built function *get_stock_returns* to store the dataframe for each ticker. We then multiply by the ticker' weight the entries of the matrix element-wise. Finally, we stack on top of each other the dataframes which allow us to recover the portfolio' dynamics.

3 Challenges and Limitations

3.1 Challenges

The main challenge for this project consisted in coding a consistent communication channel between Excel and Python. In fact, working from terminal directly from Excel is not an easy task, especially due to the lack of documentation available online. In particular, we started developing our VBA scripts on Excel for MacOS, but after struggling to make the code run, we realized that the main function to interact with the MacOS terminal has been deprecated and now requires a daunting number of additional steps. This is why we moved the project to a Windows machine in the hopes of continuing the development.
Luckily, Windows PowerShell can interact with Excel through the **Shell object** and the **objShell.Run + cmd** function. This enabled us to pass the parameters required by the Python script in string format and to execute from the Windows terminal.

3.2 Limitations

Although our dashboard works, there are some improvements that could be made in order to develop a more complete product.
Firstly, although we allow the user to select the weights for their portfolio, we could integrate a Monte Carlo simulation directly in the Python script to find the tangent portfolio and produce optimal weights according to Markowitz's theory.
Moreover, we could integrate Machine Learning techniques and prediction analysis to infer over the portfolio performance over the next few days in order to advise the user on whether an investment is feasible. This analysis would take into consideration financial statements, balance sheets and past returns of the selected stocks.
Furthermore, we are not yet satisfied with the graphical results of the analysis given our lack of excel experience. With more time, we could improve the figures with better scaling and added completeness.
Lastly, our data includes only companies listed in the SP500 index for simplicity, but it would be wise to include a larger number of companies as to give the user more freedom over their analysis and data.

The final improvement that would be interesting to implement would be to find an automated way for VBA to retrieve the user's specific **Python.exe** path (which is required by the Shell object) and to specify the working directory. The latter works, but VBA requires inverted backslashes ("\") instead of the regular backslashes provided by the VBA function ("/").

4 Demo

We report a graphical example of a portfolio, with resulting figures and performance indicators. We also report the total return of the portfolio over the selected period and the return of the SP500 as a benchmark. Note that the weights of the sample portfolio do not have any financial intuition behind them and were selected for demonstration purposes (as reflected by the performance being lower than the SP 500)

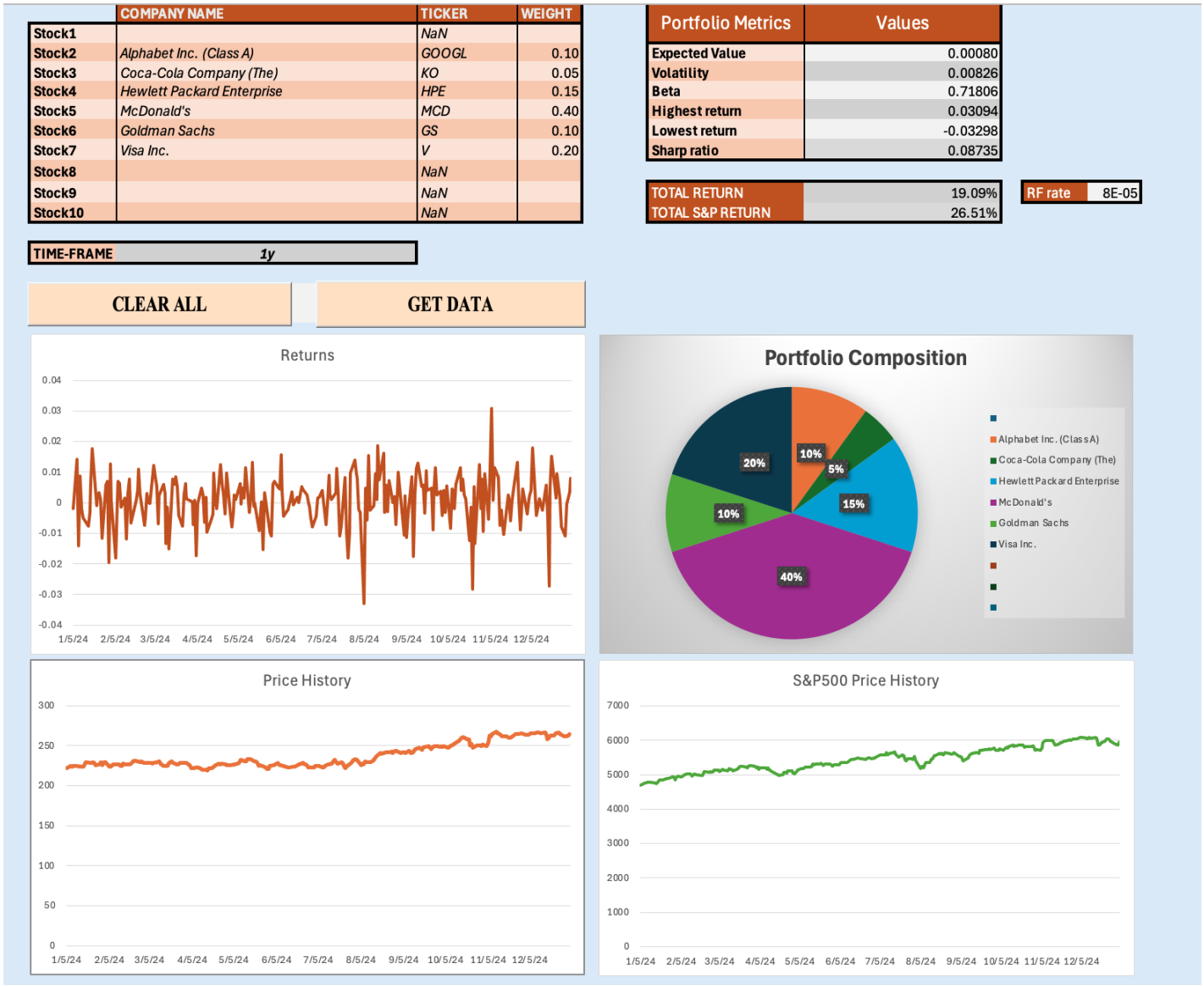

Figure 1: Sample portfolio to show the graphical results

5 Conclusions

We provide a user-friendly excel interface to get stock information and simulate historical portfolios. This project could be adjusted to respond to the users' specific needs. Getting an alternative data source in a ready-to-work environment can prove to be useful given the cost of alternatives (e.g., Bloomberg). With a few added features we believe our dashboard could truly prove useful to retrieve financial data and advise investment.

Moreover, we had the opportunity to get a first hands-on experience with VBA and its interactions with Python scripts. We noticed the low collaborative nature between the two, finding it hard to run python scripts and to provide inputs in a dynamic fashion. We also had difficulties in debugging in VBA as the error logs lack clarity.