1. OVERVIEW OF PROJECT

In this module, I will be helping my friend Steve, who just got his degree in Finance. Steve's parents are so proud of him that they decided to become his first clients. Steve's parents are passionate about green energy. They believe that as fossil fuels get used up, there will be more and more reliance on alternative energy production. However, Steve's parents haven't done much research and have instead decided to invest all their money into Daqo New Energy Corporation, a company that makes silicon wafers for solar panels.

Steve decided to contact me, an Excel power user, to help him automate tasks using Visual Basic for Applications (VBA).

PURPOSE: I will help Steve build his financial analisis automating tasks with VBA, reading and writing into cells and worksheets, making calculations to use complex logic to perform analysis and saving this code for later reuse with any stock and reducing the chances of accidents and errors.

2. RESULTS

Steve's promised to look into Daqo stock for his parents, but he's concerned about diversifying their funds. He wants to analyze a handful of green energy stocks in addition to Daqo's stock. For this, he shared with an Excel file containing the stock data he wanted to analyze. Data included the following information regarding 12 different stocks for two different consecutive years, 2017 and 2018, (see ***Image 2.1*** Overview of *"Green Stocks"* Data):

- Date
- Open
- High
- Low
- Close
- Adj. Close
- Volume

Image 2.1 Overview of *"Green Stocks"* Data

Ticker	Date	Open	High	Low	Close	Adj Close	Volume
AY	2017-01-0	19.49	19.64	19.24	19.47	16.80219	309500
AY	2017-01-0	19.58	19.76	19.5	19.67	16.97478	525300
AY	2017-01-0	19.76	19.9	19.54	19.79	17.07834	403300
AY	2017-01-0	19.88	20.18	19.69	19.99	17.25094	457600
AY	2017-01-0	20.08	20.23	19.78	20.06	17.31134	437600
AY	2017-01-1	20.11	20.4	20.01	20.29	17.50983	776000
AY	2017-01-1	20.25	20.281	19.56	19.73	17.02656	546900
AY	2017-01-1	19.75	19.81	19.52	19.53	16.85396	351200
AY	2017-01-1	19.58	20.03	19.58	19.81	17.0956	993600
AY	2017-01-1	19.83	21.34	19.81	21.28	18.36418	854500
AY	2017-01-1	21.33	21.47	21.07	21.35	18.42458	446000
AY	2017-01-1	21.35	21.66	21.24	21.32	18.3987	587500

I decided to tackle this problem by refactoring some code I already had on hand. Although the code I already had worked well for a dozen stocks, it might not work as well for thousands of stocks. I want Steve to be able to analyze the entire stock market over the last few years at the click of a button taking into consideration that I don't want it to take a long time to execute.

Refactoring is a key part of the coding process. When refactoring code, you aren't adding new functionality; you just want to make the code more efficient—by taking fewer steps, using less memory, or improving the logic of the code to make it easier for future users to read.

I decided to refactor my original code implementing the following steps:

1. Create a ticker index
2. Create three output arrays (tickerVolumes, tickerStartingPrices, and tickerEndingPrices)
3. Create a for loop to initialize the tickerVolumes to zero.
4. Loop over all the rows in the spreadsheet.
5. Increase volume for current ticker
6. Check if the current row is the first row with the selected tickerIndex.
7. Check if the current row is the last row with the selected ticker. If the next row's ticker doesn't match, increase the tickerIndex
8. Loop through your arrays to output the Ticker, Total Daily Volume, and Return.

The main difference between the refactored code and the original code is that for the original code an array was created containing each of the twelve stock tickers and two for loops were created to iterate over the tickers and then iterate over all the rows of data. In contrast, for the refactored code I included the *"tickerIndex"* variable as a counter helping me to implicitly move through indexing along the array of stock tickers instead of explicitly having two for loops. **My hypothesis was that two for loops use more computer power in relation to indexing.**

Table 2.1 Response Time in Seconds Original vs. Refactored Code

YEAR	ORIGINAL	REFACTORED	% CHANGE	AVERAGE % CHANGE
2017	0.984375	0.28125	-71%	-69%
2018	1.078125	0.359375	-67%	

As seen on ***Table 2.1*** Response Time in Seconds Original vs. Refactored Code, at first glance, refactoring the code confirmed my original hypothesis. Indeed, moving through indexing along the array of stock tickers instead of explicitly having two for loops is computationally more efficient, in both cases (2017 and 2018), reducing response time by 71% and 67%, respectively, and 69% on average!

3. SUMMARY

A. ADVANTAGES

In this case, the main obvious advantage is the reduction in response time that implementing only one for loop has. People always work under limitations, budgets, human resources, time, and equipment are some of the constraints people face in their day-to-day life. For this reason, implementing more efficient code can help teams and organizations better adapt to time and equipment constraints, e.g., analysts have to work with 8 GB RAM equipment and face pre established deadlines which means they have to make the best use of their resources to meet their goals.

B. DISADVANTAGES

Although response time reduction is great in and on itself, it can have some disadvantages. For example, having only one for loop and using indexing to move through an array can be less intuitive, especially if code is not properly commented and commit comments are not descriptive enough. Also, if there is a team who work under certain style of code and some else implements new changes it can cause communication / misunderstandings among the team, possibly impacting the work pipeline and having to set everything back to one coding style to standardize work.