**# Stock Analysis – Refactored VBA Code Christine Mitchell**

**## Overview of Project**

Our client, Steve, is interested in analyzing green energy stocks to help his parents invest in a company that not only they can believe in to better the future, but to also make a return on their investment. Steve’s parents were interested in the green energy stock, DQ. Upon analysis, we found that it did not perform well in 2018. So, Steve asked for our help to evaluate eleven additional stocks. He appreciated the workbook we created and how with the click of a button he could quickly analyze the twelve stocks he selected. But now, he would like the potential to expand the analysis to all stocks over the last few years.

**### Purpose**

Our current VBA (Visual Basic for Applications) code was effective to evaluate the designated twelve stocks, but by Steve expanding his scope to over 1000 stocks we may need to reevaluate and edit our code so that it runs more efficiently. This fine-tuning process is called refactoring. So, the next step in our service to Steve is to go through and refactor the code by improving the logic of the code and minimize the steps to reduce the amount of memory necessary to run the code. These updates should help to make the subroutine run faster and efficiently.

**## Results**

**### Compare the Stock Performance between 2017 and 2018**

Let’s take a moment and evaluate the selected stocks and their return rate in 2017 and 2018. While DQ certainly performed well in 2017 with a 199.4% return rate, in 2018 it plummeted to a -62.6%. So, it was wise for Steve to ask us to look at few more stocks. As you can see in the tables below, 2017 saw many successes where DQ was the most successful and there was only one stock, TERP, that had a negative return at -7.2%. However, in 2018, this trend did not continue for ten of the twelve stocks. We did see that RUN jumped from a 5.5% return rate to an 84% successful return rate. RUN performed 15x greater in 2018, that is a 1427.3% percent change in 2018 compared to 2017. Yet, the most impressive stock in this dataset is ENPH where it had a 129.5% and 81.9% return rate in 2017 and 2018, respectively. I would certainly advise Steve’s family to consider these two stocks because they both yielded positive return rates where RUN has seen a marked improvement and ENPH has seen consistent successful return rates over the last two years. It is wise to continue to trend both though, especially ENPH. While ENPH has been consistently green, the return is going down. From 2017 to 2018, the percent change is -37%. Therefore, I understand Steve’s rationale to evaluate all stocks and not just limit himself to the twelve originally requested.

Table

Description automatically generated

![2017v2018](https://github.com/ChristineMitchell/VBA\_HW/blob/main/Resources/2017v2018StockPerformance.png)

**### Compare Execution Times between the Original Script and Refactored Script**

As mentioned above, while our original code met the needs of the request to evaluate the return rate at the click of a button for a dozen stocks selected, it did take a relative long time to run the code as it looped through the nested loop in dataset. So, we updated the code to see just how long it took to run it with a timer and found that it averaged .9141 seconds to run through 12 stocks. That does not sound that bad, however when you want to consider evaluating 1000 stocks, that is just not fast enough. This breaks down to .0758 seconds per stock, so if we multiply this by 1000 stocks, we are looking at the subroutine taking 75.8 minutes to run! That is over an hour and 15 minutes. Not only is that not efficient, we run the risk of a run time error and the process never completing.

So, it really was essential to streamline our VBA code. We updated the code to not have a nested loop, and instead we looped an array.

Graphical user interface, application, Word

Description automatically generated

![OriginalCodeSnipits](https://github.com/ChristineMitchell/VBA\_HW/blob/main/Resources/OriginalCodeScreenPrints.png)

We set a tickerIndex to hold the input variable stock ticker array, and the three output array variables needed to provide the total volume and to calculate the return rate for each stock ticker.

Graphical user interface, application

Description automatically generated

![RefactoredCodeSnipits](https://github.com/ChristineMitchell/VBA\_HW/blob/main/Resources/RefactoredCodeScreenPrint.png)

This refactored code improved our processing time dramatically. As seen in the images below, the 2017 original code took .9375 seconds to run, then the refactored code only took .1797 seconds to run that is an 81% improvement in the performance.

A screenshot of a computer

Description automatically generated with medium confidence

![RunTime2017](https://github.com/ChristineMitchell/VBA\_HW/blob/main/Resources/VBA\_Challenge\_2017.PNG)

Respectively, we saw a similar improvement in processing the 2018 data where the original code took .8906 seconds to run while the refactored code took .1641 seconds, this time an 82% improvement in performance. I ran the macro multiple times for both years and continued to see similar results.

Graphical user interface, application, table, Excel

Description automatically generated

![RunTime2018]( https://github.com/ChristineMitchell/VBA\_HW/blob/main/Resources/VBA\_Challenge\_2018.PNG)

**## Summary**

**### What are the advantages and disadvantages of refactoring code?**

The greatest advantage of refactoring code is to improve the efficiency of a subroutine yet there are other advantages as well like getting a chance to dive deeper into your code to understand what it is happening at each step. Then, you get the opportunity to learn new methods to improve the efficiency. Of course, the greatest disadvantage is that this takes time. Time that you may not have. And of course, another issue is that you may get lost in your code and lose even the original functionality as you try to improve upon it.

**### How do these pros and cons apply to refactoring the above code?**

I appreciated the opportunity to learn the basics of VBA coding by taking steps to write the code with the module. And more importantly, it helped to teach me the basics of loops that are used in all programming languages not just VBA. When learning the original looping process, I was thrilled how that made the calculations of the data set magically populate in the designated cell with the push of the button, however, it was even more impressive to learn that there was a more efficient method. I totally appreciated how we could measure that efficiency by building a timer into the code. There were times that I got lost in refactoring the code, but that was me trying to understand the array concept. I had more debugging snags with the refactored code. The one downside on a personal note, is that I feel like as I am learning to code, I may not always know the best method and my novice abilities will be clearly seen as others review it in a real-world environment. I always want my work to be just right, so hopefully I can stop myself from going down the rabbit hole of questioning my abilities, “is this code good enough?” Not to mention the other end, what if I do not catch on that my code is not efficient and it causes issues with larger data sets. I just need to keep in mind that as I continue to learn, I will build on my skills and confidence. Obviously, this was an excellent learning process to learn how to code, and then recode. Not only did I learn that there are multiple ways to program, but that some methods are more optimal by utilizing tools like arrays to reduce the steps and the memory it takes to process the subroutines.