**Stock Analysis with VBA Code**

**Module 2 Challenge**

**Overview of Project:**

After completing a VBA code that allows Steve to analyze a handful of stocks from the 2017 and 2018 stock data set provided, he now would like to be able to expand the number of stocks analyzed and would like the code “refactored” to improve its efficient to accommodate the great number of stocks included in his analysis.

**Process:**

The original VBA code include a list of 12 stocks that Steve was analyzing. The code looked at each line individually for each of the 12 stocks, pulling the required information and listing it to the output sheet. To improve the speed and efficiency of the code I used the Array / Index capabilities in VBA to capture the data in indexed arrays and then pulled the required information for output from the arrays.

To accomplish that I created a “ticker Index” and set it to zero:

**Ticker Index:**



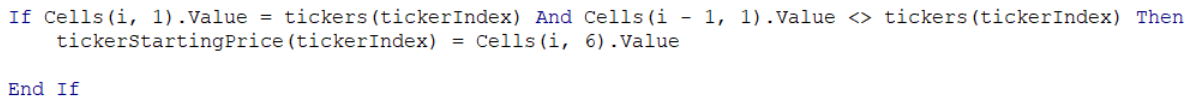
After creating arrays for the three required data fields for the analysis (Volume, Starting & Ending Prices) and setting the Volumes for each stock ticker to zero. I used a “For” loop to run through all the rows of data set using the *tickerIndex* to fill out the three arrays. Below is the code for pulling the volume information:

**Stock Volume:**

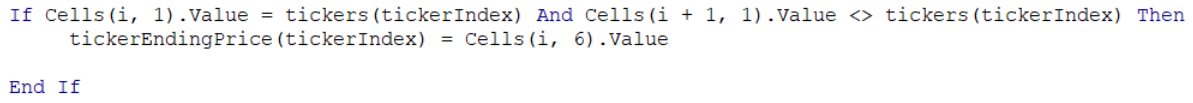


I then used the same *tickerIndex* in “If loops to the Starting and Ending stock prices used to calculate the return and to increase the stock ticker to go through all the stocks to be analyzed:

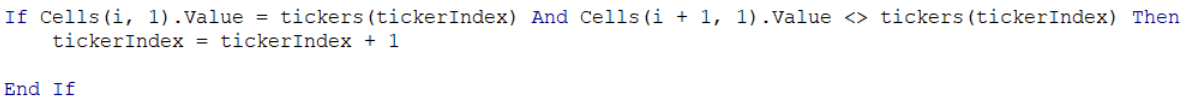
**Starting Stock Price:**



**Ending Stock Price:**

****

**Increase Stocks Ticker Index:**

****

After pulling the required information from the data set into the array, I used another “For loop” to print data from the arrays:

**Printing Output:**

Text

Description automatically generated

**Results:**

By measuring the elapsed time to run the analysis for the original 12 stocks for the original code vs the refactored code we can see how the changes have impacted the speed of processing for the analysis. As you can see below these changes significantly improved the speed of running the analysis by roughly a .76 sec reduction from .88 to .12 sec (see below):

**Year 2017**

**Original code:**

**Graphical user interface, application, table, Excel

Description automatically generated**

**Refactored code:**

**Graphical user interface, application, table, Excel

Description automatically generated**

**Year 2018**

**Original code:**

**Graphical user interface, application, table, Excel

Description automatically generated**

**Refactored code:**

**Graphical user interface, application, table, Excel

Description automatically generated**

**Summary:**

The process of refactoring programming code is beneficial for sophisticated coding programs, to improve efficiency and clarity of the code. Since there are many ways to accomplish any give task the process of refactoring provides the opportunity for more effective method to be used. As a developer creates a code the first method might work just fine but there could be more efficient methods possible. Once the code is complete there might also be patterns that could be consolidated to improve the overall process that were not noticeable as the code was originally created. Another thought would be that by having someone other than the original developer refactor the code, the fresh set of eyes and ideas might bring addition option that the original developer might not consider.

In this challenge the process of developing the original code one step at a time through the lesson modules was beneficial in understanding the process as well as teaching the VBA coding. That original code was easier to read and understand what each step of the process was doing. However, it did make it less efficient as you demonstrated with our analysis. The refactored code using the indexes makes it less legible to the average person but one you understand how indexes work can see that the code is cleaner and has less lines of code.