# Analysis of Financial Stock Data via VBA programming and a Retro Factor of the Code.

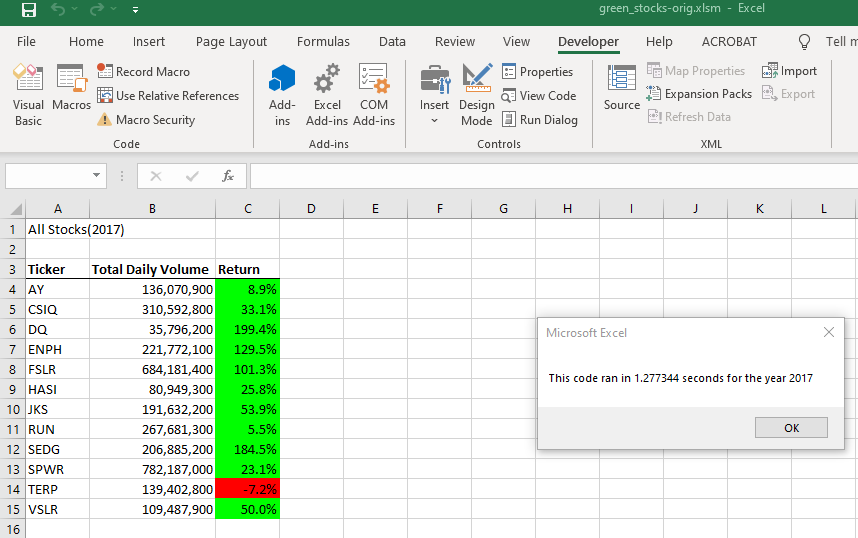
## Overview of Project:

Statistical data for 12 stocks has been collected over a period of two years (2017, 2018) on days that the stock market is open. The data consists of daily start price, end price and total volume of the respective stocks. The analysis was performed using VBA code and provides a one-line summary of each stock with its first stock market day of the year and last stock market day of the year and total volume. From there the Return can be derived. This analysis provides valuable input to an investor who has these stocks or is looking to include these stocks. The analysis is then further refined through a refactoring of the original VBA code to make it more efficient and more user friendly.

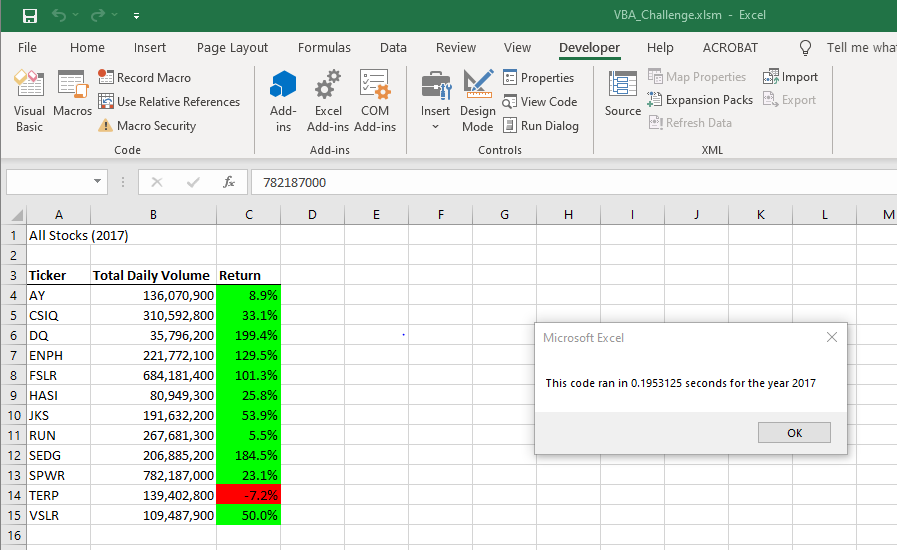
*Disclaimer*: This analysis is only provided for a statistical and analytical viewpoint. We request all users to perform their own research with due diligence prior to purchasing any of these stocks.

## Results

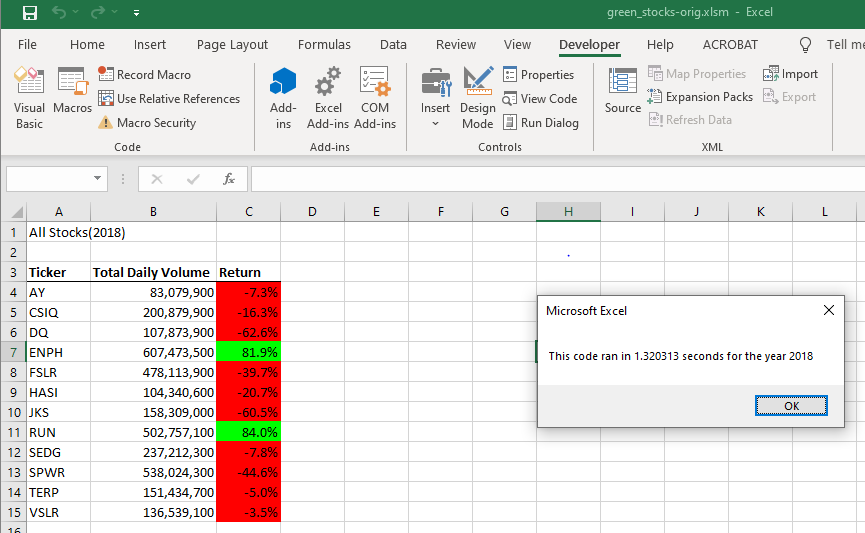
### Image for 2017 - below shows the original code



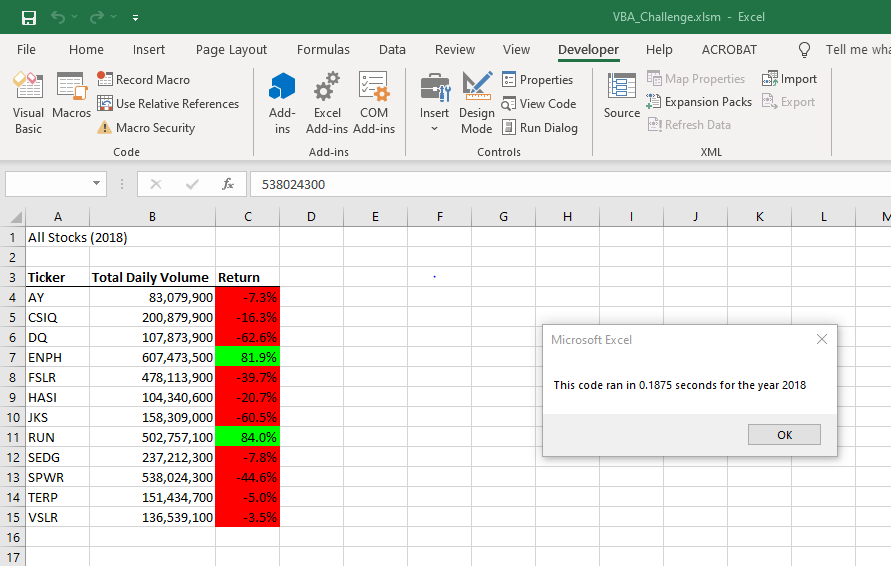
### Image below is from refactored code



### Image below for 2018 – below shows the original code



### Image below is for 2018 – below shows refactored code



### Comparing stock performance over the two-year period

## From an investor perspective investing in the stocks ENPH and RUN should be validated more closely as they show positive results over the two years of analysis.

## Code snippets for comparison – User input

The refactored code validates the user’s input for the years 2017 and 2018 only. If the user hits the Cancel button, the program exits gracefully.

User input is not validated in original code

**yearValue = InputBox("What year would you like to run the analysis on?")**

User input is validated in refactored code and workflow is managed using labels

**initialProcessing:**

**yearvalue = InputBox("What year would you like to run the analysis on?")**

**If (yearvalue = "2017" Or yearvalue = "2018") Then**

**GoTo startProcessing**

**ElseIf yearvalue = "" Then**

**MsgBox ("Exiting the application")**

**Exit Sub**

**Else**

**MsgBox ("you must type 2017 or 2018 to obtain the analysis. Please try again")**

**GoTo initialProcessing**

**End If**

**startProcessing:**

**startTime = Timer**

## Code snippets - Use of arrays for processing to coordinate a write operation rather than reading and writing line by line to the output Analysis sheet.

Writing into a file is an expensive operation. In the original code, one analyzes the data from thee 2017/2018 sheet and then write the result to the Analysis tab. So, the sheet is opened for writing 12 times (based on number of tickers)

|  |
| --- |
| '6) Output data for current ticker  Worksheets("All Stocks Analysis").Activate  Cells(4 + i, 1).Value = ticker  Cells(4 + i, 2).Value = totalVolume  Cells(4 + i, 3).Value = endingPrice / startingPrice - 1 |

In the refactored code the data is collected in arrays and then written to the analysis sheet in one operation.

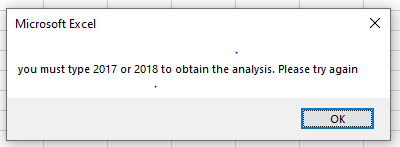
|  |
| --- |
| 4) Loop through your arrays to output the Ticker, Total Daily Volume, and Return.    Worksheets("All Stocks analysis").Activate  For i = 0 To 11  Cells(4 + i, 1).Value = tickers(i)  Cells(4 + i, 2).Value = tickerVolumes(i)    Cells(4 + i, 3).Value = tickerEndingPrices(i) / tickerStartingPrices(i) - 1    Next i |

Removing the nested FOR loops with this piece of code

|  |
| --- |
| check if the current row is the last row with the selected ticker  If Cells(j + 1, 1).Value <> tickers(tickerIndex) And Cells(j, 1).Value = tickers(tickerIndex) Then    tickerEndingPrices(tickerIndex) = tickerEndingPrices(tickerIndex) + Cells(j, 6).Value    ' Since this is the last row of aticker index, we can increment the value and go to the next ticker symbol    tickerIndex = tickerIndex + 1    End If    If tickerIndex > 12 Then  'Since all the tickers have been analyzed, we can exit now the for loop  GoTo ContinueProcessing  End If |

## Summary - Advantages or Disadvantages of Refactoring Code

Code is refactored primarily to obtain a positive result or outcome. In this case the code was refactored to provide a better user This was a benefit and probably the outcome of feedback from usability testing. The UI should be friendly so as to not frustrate the user. In this case the user is guided by labels and warning if a date or value other than 2017 or 2018 is added in the refactored code.



Most importantly execution time was reduced by 90%.

|  |  |  |
| --- | --- | --- |
| **Year** | **Original Code** | **Refactored Code** |
| 2017 | 1.27 sec | .19 sec |
|  |  |  |
| 2018 | 1.32 sec | .18 sec |
|  |  |  |
|  |  |  |

**Advantages of Refactoring**

Increase in Performance and efficiency

The original program in “*green\_stocks-orig*” had nested *for* loops. The refactored one “*VBA\_Challenge*” removed the nested *for* loops. When the last row of the ticker was reached, then the program incremented the tickerindex by one to move to the next ticker symbol. When all are complete, it uses the goTo LABEL to continue processing

This change in logic caused the execution time to reduce by a factor of approximately 6.

Refactoring code also allows a second pair of eyes review the code, thus allowing expertise to come in from different developer perspectives to enhance the code. As a result, the code gets cleaner and easier to read.

In the refactored code, the use of labels provided for faster code execution. This was another perspective.

In the process of refactoring any dormant bugs could be uncovered, thereby increasing the quality of the software. In this case we had an advantage. We could run the original code and compare the outputs in the Stock analysis tab. We knew our refactored code was accurate when the two matched. Our test case helped here.

**Disadvantages of refactoring**

Disadvantages include testing to make sure the refactored code has not broken any existing code. It requires robust test cases to be developed so they can be validated against the refactored code. Our test case to validate the accuracy was quite wholesome. We could just compare the outputs of the original program v/s the new refactored program to verify our refactored code was done correctly.

Refactoring involves understanding the logic and flow of the original code. Then only can programming techniques be utilized to enhance the code. In larger sets of code this can be daunting.

However, in the original code in “*green\_stocks-orig*”, we observed that the stocks were grouped and within each group the dates were chronologically ordered. Therefore, it was evident that when the last row was reached, we could insert logic to move to the next ticker for processing. Understanding the original flow was required to refactor effectively. Yes, it worked for this program.