**Refactoring Code for Stock Analysis**

* 1. **Overview of the project**
  2. **Background**

Stock analysis is a method evaluation of stocks for investor to make a buying or selling decision. The current analysis is using VBA excel to analyze green stock data ( green\_stock.xlsm) in 2017 and 2018. It provides Total Daily Volume and Return of each stock in selected year. Return calculated by comparing the difference between starting and ending price to the starting price.

The existing code works well for calculate the current data since the data size is moderate. However, to calculate larger data size the code needs improvement. Refactoring is a disciplined technique for improving and restructuring the design of an existing code base without changing its external behavior. This project will refactor the existing code to improve the performance of the code.

* 1. **Purpose**
* To refactor the existing stock analysis code, therefore the code will be more efficient and maintainable.
* Comparing the existing code to the refactoring code to find the difference result performance.

1. **Results**

The original dataset in the excel contains of two sheet which is 2017 and 2018. Each sheet holds the stock data of the year. Table 1 below describe the summary of data used for analysis. It has 12 Ticker with 251 row data on each ticker. On the dataset there are 8 columns which are ticker, date, open, high, low, close, adj close and volume. Table 2 below will show us the mapping of data column and its usage in analysis, if the column is not present in the table, then it is not used in analysis.

Table 1 Summary of stock Analysis Data

|  |  |  |  |
| --- | --- | --- | --- |
| **2017** | | **2018** | |
| **Stock Name(Ticker Name)** | **Data Count** | **Stock Name(Ticker Name)** | **Data Count** |
| AY | 251 | AY | 251 |
| CSIQ | 251 | CSIQ | 251 |
| DQ | 251 | DQ | 251 |
| ENPH | 251 | ENPH | 251 |
| FSLR | 251 | FSLR | 251 |
| HASI | 251 | HASI | 251 |
| JKS | 251 | JKS | 251 |
| RUN | 251 | RUN | 251 |
| SEDG | 251 | SEDG | 251 |
| SPWR | 251 | SPWR | 251 |
| TERP | 251 | TERP | 251 |
| VSLR | 251 | VSLR | 251 |
| **Grand Total** | **3012** | **Grand Total** | **3012** |

Table 2 Column Usage in Analysis

|  |  |
| --- | --- |
| **Column Name** | **Usage in Analysis** |
| Ticker | Ticker Name |
| Close | \* Starting price if it’s the first data row of each ticker \* Ending Price if it’s the last data row of each ticker |
| Volume | To calculate total Volume |

The Subroutine AllStocksAnalysis() will be used to calculate and enter the data result in excel sheet. The main difference between the existing code and the refactoring code is the usage of arrays to hold volume, starting price and ending price value in the refactoring code instead of implementing nested loop to find the value.

We used nested loops to calculate volume In existing code, as well as defining the starting and the ending price.

For i = 0 To 11

ticker = tickers(i)

totalVolume = 0

Worksheets(yearValue).Activate

For j = 2 To RowCount

' increase totalVolume if the ticker value (row A) is DQ

'To count totalvolume

If Cells(j, 1) = ticker Then

totalVolume = totalVolume + Cells(j, 8).Value

End If

'To count starting price for Yearly Return

If Cells(j - 1, 1) <> ticker And Cells(j, 1) = ticker Then

startingPrice = Cells(j, 6).Value

End If

'To count Ending price for Yearly Return

If Cells(j + 1, 1) <> ticker And Cells(j, 1) = ticker Then

EndingPrice = Cells(j, 6).Value

End If

Next j

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

Next i

The First loop ( For i = 0 to 11)  is used to loop through the tickers. Tickers is an array that holds all the ticker names. The second loop (for j = 2 to RowCount) is nested from the fist loop, to loop through rows in the data to find total volume, starting price, and ending price of the ticker from the first loop.

For the refactoring code, arrays will be used to hold total volume, starting price, and ending price of each ticker using ticker index.  There are no nested loops in the refactoring code. First the code creates a loop to initialize ticker volume to zero, then loop through the row using TickerIndex to access the correct index of each ticker to count total volume and define starting and ending price. The ticker index value will be added when the next row ticker does not match with the current ticker.

'1a) Create a ticker Index

TickerIndex = 0

'1b) Create three output arrays

Dim tickerVolumes(12) As Long

Dim tickerStartingPrices(12) As Single

Dim tickerEndingPrices(12) As Single

''2a) Create a for loop to initialize the tickerVolumes to zero.

For j = 0 To 11

tickerVolumes(j) = 0

Next j

''2b) Loop over all the rows in the spreadsheet.

For i = 2 To RowCount

'3a) Increase volume for current ticker

tickerVolumes(TickerIndex) = tickerVolumes(TickerIndex) + Cells(i, 8).Value

'3b) Check if the current row is the first row with the selected tickerIndex.

'If Then

If Cells(i - 1, 1) <> tickers(TickerIndex) Then

tickerStartingPrices(TickerIndex) = Cells(i, 6).Value

'End If

End If

'3c) 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.

'If Then

If Cells(i + 1, 1) <> tickers(TickerIndex) Then

tickerEndingPrices(TickerIndex) = Cells(i, 6).Value

'3d Increase the tickerIndex.

TickerIndex = TickerIndex + 1

'End If

End If

Next i

'4) Loop through your arrays to output the Ticker, Total Daily Volume, and Return.

For i = 0 To 11

Worksheets("All Stocks Analysis").Activate

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

The performance of the code will be measured by how long it takes to execute the code. Figure 1  and Figure 2 will show us times to execute existing code for 2017 and 2018 data. While Figure 3 and Figure 4 show execution times for the code after refactoring. The comparison between existing and refactoring code of the execution time is shown in Table 3.

Graphical user interface, application, table, Excel

Description automatically generated

Figure 1 Execution times for 2017 data in Existing Code

Graphical user interface, application, table, Excel

Description automatically generated

Figure 2 Execution times for 2018 data in Existing Code

Graphical user interface, application, table, Excel

Description automatically generated

Figure 3 Execution times for 2017 data in Refactoring Code

Graphical user interface, application, table, Excel

Description automatically generated

Figure 4 Execution times for 2018 data in Refactoring Code

Table 3 Comparison of Execution Times

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Year** | **Existing** | **Refactoring** | **Difference (Existing-Refactoring)** | **Reduction in time (Existing- Refactoring)/Existing x100%** | **Increase in Performance (Existing- Refactoring)/Refactoring x100%** | **Multiplication (Existing/Refactoring)** |
| 2017 | 2.836 | 0.375 | 2.461 | 86.777 | 656.267 | 7.563 |
| 2018 | 2.930 | 0.195 | 2.735 | 93.345 | 1402.564 | 15.026 |

*\*The values show in this table are roundup to 3 decimals*

From Table 3, it can be concluded that there are positive improvements for the code with refactoring. The reduction in time for executing the code is 86.777% and 93.345%. Furthermore, the increment in performance and the multiplication suggests that the code is becoming more efficient. There are 656.267% and 1402.564% increase in performance and the code becomes 7.563 and 15.026 faster compared to the existing code.

1. **Summary**
   1. **Advantages and Disadvantages of Refactoring Code**

Advantages:

* The code is easier to enhance and maintain in the future
* Less complex and easier to read
* Prevents many future defects
* May increase performance

Disadvantages:

* May introduce bugs
* Takes times and expensive in budget
* Risky if the application is too large and there is not proper test case in existing code
  1. **Pros and Cons of Refactoring in This Project**

Based on the result above, it is clarified that refactoring will contribute positively to the existing code in this project. It increases the performance and generates faster execution time. The factor that may lead to the improvement is the usage of array instead of nested loops for calculation. The disadvantage of refactoring, however, is the possibility of bugs occurrence and not working as expected, since we only have two dataset (2017 and 2018) to compare and no further test case is available.