2.2.3 Find Total Daily Volume for DQ in 2018

Steve's parents want to know how actively DQ was traded in 2018. They believe that if a stock is traded often, then the price will accurately reflect the value of the stock. If we sum up all of the daily volume for DQ, we'll have the yearly volume and a rough idea of how often it gets traded.

At this point, we have an output worksheet to hold our analysis and know how to write readable code. Now we can start writing more complex code to perform our analyses. In this section, we'll dive into loops, conditionals, code patterns, and how to find coding help on the internet. Let's get started!

Loops

We're going to calculate the total daily volume in 2018 using **loops**.

IMPORTANT

> **Loops** tell a computer to repeat lines of code over and over (and over, and over) again. `for` loops tell the computer to repeat the lines of code a specific number of times. You can think of a `for` loop like telling the computer to "run this code *for* as many loops as I tell you to." There are a few different kinds of loops, but `for` loops are the workhorse of loops. It's entirely possible that you'll never need to use any other kind of loop.

To find the total daily volume, we'll loop through every row in our stock data worksheet and check if the ticker for that row is DQ. If it is, we'll add its daily volume to our total volume.

In VBA, the syntax for a `for` loop has a beginning, middle, and end. The beginning is one line that tells VBA that we're opening a `for` loop; the middle is the block of code to be repeated; and the end is one line that closes the block of code.

The opening line of a `for` loop uses the keyword `For` and an **iterator**. Iterators are named variables that change values over the course of the `for` loop, usually increasing by 1, thus holding the number of times the loop has repeated. For example, if we wanted a `for` loop that would loop exactly 3 times, the opening line would look like this:

```
For i = 1 to 3
```

Inside the code block, we can use the iterator like any other variable. So, if we want to display 3 message boxes in a row, showing the number of times the loop has repeated, our code block would look like this:

```
For i = 1 To 3
    MsgBox (i)
```

Now we need to close the `for` loop with our last line. To tell VBA that the `for` loop has ended, we use the keyword `Next` and the iterator we used: `i`.

NOTE

> We need to specify the iterator because there can be more than one `for` loop going on. These are **nested** `for` loops, which we'll get to later.

So our full `for` loop looks like this:

```
For i = 1 To 3
    MsgBox (i)

Next i
```

Make a new macro and name it `DQAnalysis`, then run the `for` loop to make sure it works.

Because the iterator is treated like any other variable inside of the code block, we can use it to interact with our data. To see this, change the `for` loop to display each of the stock data column names in a MsgBox. There are 8 columns, so change the iterator to go from 1 to 8, and then change the MsgBox to display `Cells(1, i)`.

```
For i = 1 To 8
    MsgBox (Cells(1, i))

Next i
```

Make sure the 2018 sheet is active. There are two ways to do this. The first is to select the correct sheet from the file structure display to the left of your window, then run the macro. The other option is to update your

code by adding `Worksheets("2018").Activate` before the `for` loop. Once ready, execute your macro.

NOTE

> Using `Worksheets("2018").Activate` in this macro will ensure that the analysis is always performed on the correct sheet.

You should get 8 message boxes in a row, each displaying one of the column headers.

Let's continue with our `DQAnalysis` example by inserting the code we have created so far into the lines above our new for loop. Your code should look like this:

```
Sub DQAnalysis()
    Worksheets("DQ Analysis").Activate

    Range("A1").Value = "DAQO (Ticker: DQ)"

    'Create a header row
    Cells(3, 1).Value = "Year"
    Cells(3, 2).Value = "Total Daily Volume"
    Cells(3, 3).Value = "Return"

    Worksheets("2018").Activate
    For i = 1 To 8
        MsgBox (Cells(1, i))

    Next i

End Sub
```

We can also use variables that are defined outside of the `for` loop inside our block of code. If we create a `totalVolume` variable and set it equal to zero, we can add to it inside of the loop; when the loop is done, `totalVolume` will hold the sum of all the volume.

There are 3,013 rows in the 2018 worksheet. We don't want to include the header row, so our iterator will go from 2 to 3013. We could write a loop where we hard-code the numbers for the iterator, like this:

```
totalVolume = 0

Worksheets("2018").Activate
For i = 2 To 3013
    'increase totalVolume

Next i
```

Hard-coded values like these are known as **magic numbers**: if you didn't write this code, it would seem like the numbers were created by magic. Magic numbers should be avoided as much as possible. A better way to use these numbers is to create variables with informative names and assign them the values we want to use.

Make a `rowStart` and a `rowEnd` variable with the values 2 and 3013, respectively, and use those in the opening line of your `for` loop:

```
rowStart = 2
rowEnd = 3013
totalVolume = 0

Worksheets("2018").Activate
For i = rowStart To rowEnd
    'increase totalVolume

Next i
```

NOTE

> `totalVolume`, `rowStart`, and `rowEnd` are all new variables, so you might be asking yourself why we didn't need to initialize them with the `Dim` keyword. In this case, because we're setting values as we initialize

> them, VBA can infer the data type from the value being assigned. Here, all the values are integers, so VBA will create integer variables for us in the background.
>
> Using named variables here allows us to make our code more flexible, because if we update the data, we only have to change the value of the variables to match. However, an even better solution is to use code to find the value dynamically based on the data. We'll explain how to do that in VBA when we're done writing our loop.

Volume is in the 8th column, so we can increase the `totalVolume` by the value in `Cells(i, 8)`:

```
rowStart = 2
rowEnd = 3013
totalVolume = 0

Worksheets("2018").Activate
For i = rowStart To rowEnd
    'increase totalVolume
    totalVolume = totalVolume + Cells(i, 8).Value

Next i
```

To check that `totalVolume` increased, display the value with `MsgBox()`. To do this, adjust your code like so:

```
MsgBox(totalVolume)
```

This is the total volume for all the stocks; we want just the total volume for DQ. To do this, we need another tool in our toolbelt: conditionals. Before we jump into that, let's make sure your code is accurate. So far, your code should look like this:

```
Sub DQAnalysis()
    Worksheets("DQ Analysis").Activate

    Range("A1").Value = "DAQO (Ticker: DQ)"

    'Create a header row
    Cells(3, 1).Value = "Year"
    Cells(3, 2).Value = "Total Daily Volume"
    Cells(3, 3).Value = "Return"

    Worksheets("2018").Activate
    rowStart = 2
    rowEnd = 3013
    totalVolume = 0

    For i = rowStart To rowEnd
        'increase totalVolume
        totalVolume = totalVolume + Cells(i, 8).Value

    Next i

End Sub
```

Please Wait...

Conditionals

IMPORTANT

> **Conditionals** tell the computer that certain lines of code should run only under certain conditions. The workhorse of conditionals is an **if-then**

> **statement**. The if-then statement checks if a condition is true. If it is, then a block of code will be run.

In VBA, the syntax of an if-then statement also has a beginning, middle, and end. The beginning is one line to open the if-then statement; the middle is a block of code to run if the condition is true; the end is a line that closes the if-then statement.

The opening of an if-then statement uses the keywords `if` and `then`, and a condition. The code block can be any number of lines of code. To close the if-then statement in VBA, add the line `End If`.

```
If condition Then
    'Run code if condition is true
End If
```

The condition is a logical expression that is either true or false. For example, if we check that 2 is less than 3, it evaluates to true, so the code will run.

```
If 2 < 3 Then
    'This code will always run
End If
```

However, if we check that 3 is less than 2, it will evaluate to false, so the code block will not run.

```
If 3 < 2 Then
    'This code will never run
End If
```

Let's hop back into `DQAnalysis`. In the `for` loop we wrote in the previous section, if we add a conditional to check whether the ticker is "DQ," we can then only increase `totalVolume` for DQ's volume.

The condition we're checking is that the value of the cell in the first column is "DQ," so our code will look like this:

```
rowStart = 2
rowEnd = 3013
totalVolume = 0

Worksheets("2018").Activate
For i = rowStart To rowEnd
    'increase totalVolume if ticker is "DQ"
    If Cells(i, 1).Value = "DQ" Then
        totalVolume = totalVolume + Cells(i, 8).Value
    End If

Next i

MsgBox(totalVolume)
```

Now we just need to set the value in our output worksheet to show the total volume. To do this, activate the output worksheet, and then use `Cells().value` or `Range().value` to set the value:

```
    Worksheets("DQ Analysis").Activate
    Cells(4, 1).Value = 2018
    Cells(4, 2).Value = totalVolume
```

Let's do a quick code check. Compare your code to the code below to check for errors.

```
Sub DQAnalysis()

    Worksheets("DQ Analysis").Activate
```

```
    Range("A1").Value = "DAQO (Ticker: DQ)"

    'Create a header row
    Cells(3, 1).Value = "Year"
    Cells(3, 2).Value = "Total Daily Volume"
    Cells(3, 3).Value = "Return"

    rowStart = 2
    rowEnd = 3013
    totalVolume = 0

    Worksheets("2018").Activate
    For i = rowStart To rowEnd
        'increase totalVolume if ticker is "DQ"
        If Cells(i, 1).Value = "DQ" Then
            totalVolume = totalVolume + Cells(i, 8).Value
        End If
    Next i

    'MsgBox (totalVolume)

    Worksheets("DQ Analysis").Activate
    Cells(4, 1).Value = 2018
    Cells(4, 2).Value = totalVolume

End Sub
```

Now is a good time to save your work. Don't forget to save your work often! Then run your code to make sure everything is working.

FINDING

Check the total daily volume in the "DQ Analysis" worksheet. You should see that DQ traded 107,873,900 shares in 2018. If your column isn't wide enough to hold the entire number, then it will appear as scientific notation instead. Expanding the column's width will convert the data to the expected number.

ADD/COMMIT/PUSH

If everything is working correctly, save your changes to `green_stocks.xlsm` and upload it again to the "stocks-analysis" repository in GitHub. Saving incremental changes to files on GitHub is called **pushing** to the repository. You may have to exit Excel before pushing to GitHub. If so, save your worksheet and close it, then open it again once the file has been pushed to GitHub.

Patterns

The code that we used to calculate DQ's yearly volume is a simple **design pattern**. Recognizing and utilizing design patterns is crucial for writing good code.

IMPORTANT

> In programming, **design patterns** (or just **patterns)** are templates to solve similar problems. Patterns aren't necessarily code but rather reusable structures to help us write our code.

The pattern we used follows this general structure:

1. Initialize a variable to hold a sum.
2. Set the variable to zero.
3. Start a `for` loop.
4. Use a conditional to increase the sum variable by a value.
5. End the loop.

IMPORTANT

> Design patterns are larger than a single programming language. They offer a way to organize a process so that it can be put into code.
>
> Programmers use **pseudocode** to break down algorithms and processes into a design pattern without being tied to a specific language. Pseudocode can range from writing a list of tasks in natural language to formatting code with indentation and adding simple keywords. The main purpose of pseudocode is to think through the logic of the code before actually writing it.

In pseudocode, the pattern looks like this:

```
totalValue = 0

For i = Start To finish

    If Condition Then
        totalValue = totalValue + currentValue
    End If

Next i
```

You should always be on the lookout for new patterns to use. Every time you solve a single problem, consider whether you can use that same pattern to solve another problem.

The opposite of a good design pattern is an anti-pattern. **Anti-patterns** are common responses to problems, but they may be ineffective, too specialized, or generally counterproductive. Quick and dirty workarounds, called **kludges**, often use anti-patterns.

NOTE

> Anti-patterns and kludges are part of a broader idea of **code smells**, where the code works and solves the problem it's supposed to, but

> something about the code indicates that there is probably a more elegant and productive solution to the problem.

We've already avoided one anti-pattern: an unnamed magic number for the row count. Let's take it even further and, instead of giving our magic number a name, we'll have VBA figure out the value for us. This way, if the data changes, our code will still work.

Research Practice

To calculate DQ's total daily volume, we need to loop through all of the stocks, so we've typed the number of rows into the code itself. What would be even better, though, is to use VBA to find the number of rows to loop over. Unfortunately, VBA doesn't have a nice function or method to figure that out. But we can't be the first person to have this problem; someone must have found a solution. We just need to find it.

Enter Google! Programmers and analysts use Google all the time to find solutions to on-the-job problems and tasks—there's no shame in the googling game. Let's get some hands-on experience with this real-world skill and find a way to get the number of rows with VBA.

Search for something like "VBA get the number of rows with data." Note that searching for the right keywords is way more important than being grammatically correct. Also be sure to include the programming language that you're using—in this case, VBA.

A ton of search results will come up, but most of them will fall into four categories:

1. Official documentation
2. Stack Overflow
3. Quora
4. Expert blog posts

Official Documentation

For VBA questions, the official documentation is published by Microsoft. In general, official documentation is a reference guide published by the creator of the language or software. You might think this is the logical place to check first, but that's not always the case. Because official documentation is meant to be an exhaustive reference, it's often extremely dense and difficult to understand, especially when you're first learning a programming language. In the worst cases, the documentation isn't even up-to-date and therefore contains incorrect information. It's often better to get explanations from people who are actually using the code.

Stack Overflow

Stack Overflow (https://stackoverflow.com/) is almost always the best resource for getting practical solutions. Programmers will half-joke that their job is just googling problems and copying and pasting Stack Overflow code. Stack Overflow works like this: someone asks a question about a problem they're running into and experts will provide answers, almost always with sample code. The expertise level is top-notch; sometimes the person who invented the program will be the one giving the answer! Answers are voted on, so you can get a sense of which answers are more authoritative than others.

Quora

Quora (https://www.quora.com/) has a similar format to Stack Overflow, but the questions and answers tend to be more theoretical. The expertise level is similarly high, but the emphasis is more on understanding concepts rather than just getting some code that works. If you're struggling to do something, go to Stack Overflow; if you're struggling to understand something, go to Quora.

Expert Blog Posts

On the internet, anyone can claim to be an expert, so blog posts vary wildly in quality—especially because there is no mechanism for people to give feedback on the quality of a blog's explanations. However, expert blog

posts tend to blend sample code and theory, so a good post will help a new concept click.

Narrow Your Search

There's no reason to feel awkward about using search engines to figure out problems. Welcome to being a programmer! A big part of any programming job is googling solutions and spending a lot of time on Stack Overflow. Even experienced programmers do this on a daily basis.

Our search "VBA get number of rows with data" gives a number of results, and many look promising. There's a Stack Overflow link, a link to the official documentation, and a handful of blogs and forums. Click on a few links and skim the information to get a feel for how different sites answer questions like this. Here are two curated, potential solutions. (Don't worry about finding the exact links to these. And remember that you might find even better solutions!)

Potential Solution #1: Stack Overflow

An answer on Stack Overflow provides the following code for finding the number of rows with data:

```
lastRow = Cells(Rows.Count, "A").End(xlUp).Row
```

This is fairly complicated, so let's break it down:

- `Cells(Rows.Count, "A")` goes to the bottom cell in column A, which may extend past the last row of data in the sheet.
- `.End(xlUp)` is the same as pressing END and then the up arrow in Excel, which will go to the last cell with data in column A. We use this to move from the bottom of the sheet to the last row of data.
- `.Row` returns the row number.

If your eyes glazed over during the explanation of how that line of code works, don't worry. In a perfect world, we would have time to research everything and understand it completely, but sometimes we just need to get code that works, even if we can't explain exactly how it works. It's a good idea to add a comment that explains where the code came from, especially if we're not entirely sure why it works. Linking back to where the code came from can help anyone reading the code understand why it is being used.

```
'rowEnd code taken from https://stackoverflow.com/questions/18088729/row-cou
rowEnd = Cells(Rows.Count, "A").End(xlUp).Row
```

Potential Solution #2: Blog Post from an Excel Expert

The previous Stack Overflow solution works (and spoiler alert: it's what we're going to use in our code), but here's another possible solution to demonstrate that there are usually multiple ways to solve a problem. This one is from an Excel expert. It's more involved than the previous solution, so don't worry if there are some things that don't make sense yet.

```
LastRow = Cells.Find("*", searchorder:=xlByRows, searchdirection:=xlPrevious
```

As you probably noticed, we haven't used the `Find` method before. It has many options, so figuring out how it works can be confusing. Let's dissect what's happening here.

- The first argument, `"*"`, says to look for anything. Asterisks are often used as a wildcard that means to "match anything." In this case, it means to match anything except a blank cell.
- `searchorder` says to search by rows, not columns.

- `searchdirection` set to `xlPrevious` means to start from the end of the worksheet range and work backward until the `Find` method finds something that matches, which will be in the last row.
- Finally, the `Find()` method returns a single cell. That cell has a row property, which is the number of the row that it is in. This is the number that we want, so we access it with the `.Row` property.

Keep in mind that these solutions are not the only ones out there; you may have found a different way to find the number of rows based on your search results. That's perfectly fine! Just be sure to test it out to make sure it works in your workbook. If it doesn't, go back to your Google results and find another solution.

Let's take a moment to do another code check. So far, your code should look like the code below. If not, go ahead and update it to match.

```
Worksheets("DQ Analysis").Activate

Range("A1").Value = "DAQO (Ticker: DQ)"

'Create a header row
Cells(3, 1).Value = "Year"
Cells(3, 2).Value = "Total Daily Volume"
Cells(3, 3).Value = "Return"

Worksheets("2018").Activate

rowStart = 2
'DELETE: rowEnd = 3013
'rowEnd code taken from https://stackoverflow.com/questions/18088729/row-cou
rowEnd = Cells(Rows.Count, "A").End(xlUp).Row
totalVolume = 0

For i = rowStart To rowEnd
    'increase totalVolume
    If Cells(i, 1).Value = "DQ" Then
        totalVolume = totalVolume + Cells(i, 8).Value
    End If

Next i
```

```
Worksheets("DQ Analysis").Activate
Cells(4, 1).Value = 2018
Cells(4, 2).Value = totalVolume
```