2.4.2 Conditional Formatting

The table is a lot easier to read now, but it's still difficult to determine at a glance which stocks performed well and which ones did not. Let's format our data so that Steve can determine stock performance at a glance.

If we make positive returns green and negative returns red, it'll be a lot easier to determine which stocks did well and which ones didn't. Let's add some formatting based on the values of the returns. We'll loop through each of the returns, and if the return is positive, we'll make the background color green; if the return is negative, we'll make the background red; otherwise, we'll clear the background color.

REWIND

Earlier, we used Excel's built-in conditional formatting to accomplish this kind of highlighting. However, since we're now building the output sheet programmatically, we're going to handle the conditional formatting ourselves with code.

Color Formatting

The color of a cell belongs to the `Interior` property of the cell. We'll need to access the `Color` property of the `Interior` to change the color of the cell. There are a few ways to refer to colors in VBA, but the simplest and easiest to read are the VBA standard colors. For most basic color names, you can put `vb` before the color name, and VBA will refer to that color. For example, the standard VBA color green is `vbGreen`. However, the color of an empty cell isn't set to white, even though it's usually displayed as white. To clear the color, it should be set to `xlNone`.

Therefore, to set the color of the cell at row 4, column 3 to green, we write:

```
Cells(4, 3).Interior.Color = vbGreen
```

To set the color of that cell to red, we write:

```
Cells(4, 3).Interior.Color = vbRed
```

And to clear the cell, we write:

```
Cells(4, 3).Interior.Color = xlNone
```

Conditionals Revisited

We could use separate if-then statements for each condition, but we want to cover all cases, including ones we may not have thought of. For example, what if the formula returns an error? How would this be formatted?

If-then statements have a couple more tricks up their sleeve, namely, the else statement and the elseif statement.

Just as the if-then statement creates code that runs if a condition is true, the **else statement** runs a separate block of code if a condition is false. The **elseif statement** introduces another condition: it runs if the first condition is false and the second condition is true.

To see an example, let's apply conditional formatting to the cell in row 4, column 3. First, to color the cell green, create an if-then block that checks if the value is greater than zero.

```
If Cells(4, 3) > 0 Then
    'Color the cell green
    Cells(4, 3).Interior.Color = vbGreen

End If
```

Then, add an elseif block to check if the cell is less than zero.

```
If Cells(4, 3) > 0 Then
    'Color the cell green
    Cells(4, 3).Interior.Color = vbGreen
ElseIf Cells(4, 3) < 0 Then

    'Color the cell red
    Cells(4, 3).Interior.Color = vbRed
End If
```

Finally, add an else block to clear the cell if the value is neither positive nor negative.

```
If Cells(4, 3) > 0 Then
    'Color the cell green
    Cells(4, 3).Interior.Color = vbGreen
ElseIf Cells(4, 3) < 0 Then

    'Color the cell red
    Cells(4, 3).Interior.Color = vbRed

Else
    'Clear the cell color
    Cells(4, 3).Interior.Color = xlNone

End If
```

Set Up the Loop

Now that we know how to conditionally format one cell, we can conditionally format all the cells we need with a loop. The data starts on row 4 and goes to row 15, so we'll create two new variables to hold those values—remember, we don't want unnamed magic numbers in our code—and have the `for` loop iterate through each row.

```
    dataRowStart = 4
    dataRowEnd = 15
    For i = dataRowStart To dataRowEnd

    Next i
```

Inside the loop, we'll use the same code from the previous section but replace the row number with the iterator.

First, check if the cell is greater than zero. If so, change the color to green.

```
    dataRowStart = 4
    dataRowEnd = 15
    For i = dataRowStart To dataRowEnd
```

```
        If Cells(i, 3) > 0 Then

            'Change cell color to green
            Cells(i, 3).Interior.Color = vbGreen

        End If

    Next i
```

Then, check if the cell is less than zero. If so, change the color to red.

```
    dataRowStart = 4
    dataRowEnd = 15
    For i = dataRowStart To dataRowEnd

        If Cells(i, 3) > 0 Then

            'Change cell color to green
            Cells(i, 3).Interior.Color = vbGreen

        ElseIf Cells(i, 3) < 0 Then

            'Change cell color to red
            Cells(i, 3).Interior.Color = vbRed

        End If

    Next i
```

Finally, if neither condition is true, clear the cell color.

```
    dataRowStart = 4
    dataRowEnd = 15
    For i = dataRowStart To dataRowEnd

        If Cells(i, 3) > 0 Then

            'Color the cell green
            Cells(i, 3).Interior.Color = vbGreen

        ElseIf Cells(i, 3) < 0 Then

            'Color the cell red
            Cells(i, 3).Interior.Color = vbRed

        Else

            'Clear the cell color
            Cells(i, 3).Interior.Color = xlNone

        End If

    Next i
```

Let's get some added practice with `for` loops, conditionals, and formatting.

SKILL DRILL

Using nested `for` loops and conditionals, format an 8 x 8 square of cells with a checkerboard pattern.

Hint: The mod function returns the remainder of the division between the two numbers. If the remainder is `0`, then the number if even. If the remainder is not `0`, then the number is odd.

It's always a good idea to save your work.

ADD/COMMIT/PUSH

Save your changes and push to the "stocks-analysis" repository in GitHub.