Programming case: A methodology for programmatic web data extraction

John N. Dyer
Georgia Southern University

ABSTRACT

Web scraping is a programmatic technique for extracting data from websites using software to simulate human navigation of webpages, with the purpose of automatically extracting data from the web. While many websites provide web services allowing users to consume their services for data transfer, other websites provide no such service(s) and it is incumbent on the user to write or use existing software to acquire the data. The purpose of this paper is to provide a methodology for development of a relatively simple program using the Microsoft Excel Web Query tool and Visual Basic for Applications that will programmatically extract webpage data that are not readily transferable or available in other electronic forms. The case presents an overview of web scraping with an application to extracting historical stock price data from Yahoo's Finance® website. The case is suitable for students that have experience in an object-oriented a programming course, and further exposes students to using Excel and VBA, along with knowledge of basic webpage structure, to harvest data from the web. It is hoped that this paper can be used as a teaching and learning tool, as well as a basic template for academicians, students and practitioners that need to consume website data when data extraction web services are not readily available. The paper can also add value to student's programming experience in the context of programming for a purpose.

Keywords: Data Extraction, Web Scraping, Web Query, Web Services

INTRODUCTION

Increasingly, many individuals and organizations have the need to extract massive amounts of data from the web. The basic technique for extracting the data is web scraping, which can be loosely defined as a computer program to extract data from a website. Wed scraping is commonly used to facilitate online price comparisons, contact scraping, online product catalog scraping, weather data monitoring, economic/demographic/statistical data extraction, and web mashups, among other uses. Web scraping is also a subset of People-Oriented programming, which empowers individuals to program web-based self-fashioned tools that ultimately suit the user's own needs (International Journal of People-Oriented Programming (IJPOP), n.d.).

Often times a company releases its application program interface (API) to the public so that software developers can design products that are powered by its service for extracting data (Roos, 2007). In many cases an API doesn't exist and the developers must write their own. Web scraping is commonly used to either extract data from a legacy system (which has no other mechanism to transfer data), or to extract data from a website which does not provide a more convenient API (Data scraping, 2015).

Since most webpages are designed for human end-users, navigation and data extraction are not necessarily easily automated. As such, web scraping is typically considered a "last resort" tool, with high programming and processing overhead. It focuses on acquiring HTML formatted data from a webpage and storing the data in a variety of formats, including a text file, a worksheet, or a database. Since websites are built using HTML or XHTML, web scraping utilizes software to simulate human exploration and extraction of data from the web, pulling the data directly out of the HTML. The program then goes through all available pages and collects data and images as a human would do manually using mouse clicks and copy-and-paste. As such, any content that can be viewed on a webpage or is contained in the source code can be scraped. It is for this reason that web scraping tools programmatically automate data extraction/acquisition from a website.

The focus on this case study is on developing a methodology for programmatic web data extraction when an API or other web service is not available. Note that although there are software programs and web-browser add-ins that facilitate web scraping (paid subscription and free-ware), this paper illustrates writing one's own program. A quick Internet search will reveal many of the available web scraping programs, as well as a highly-rated Google Chrome browser extension named Web Scraper® (Web Scraper, n.d.) and a program named Data Toolbar® (Web Data Extraction Software Made Simple, n.d.). A good overview of web scraping can be found at Brody, H. (2012), while a good instructional resource can be found from Brody, H. (2013).

To visualize web scraping, consider an online product catalog wherein a certain category of products will result in many items displayed on a webpage, but the catalog will display only a subset of all the items per webpage (usually displayed in an HTML table element). That is, a single web page may display 20, 50 or 100 items per page, with paging links allowing navigation across all subsequent webpages. Such is the case with auction sites such as eBay®, wherein a search may result in 50 items per page across multiple webpages. Although many websites allow the user to choose a fixed number of items per page, few websites offer a single page view of unlimited size.

Now consider the task of downloading the data for each item across each webpage. If a single page view of all the items were available the task would simply be that of a few keystrokes allowing one to select the desired data, copy the data, and then paste the data into

document file or worksheet. But, it is easy to imagine a website wherein thousands of items are displayed across many webpages, hence one is limited to a small subset in each webpage view. Without some type of automation, it would require one to navigate through each webpage and linked page, selecting the desired data, copying, and then pasting each of the webpage's contents. This is obviously not a practical approach, especially if one wanted to acquire the complete contents of a large website on a frequent basis.

Fortunately, if a website is structured appropriately, Excel contains all the tools necessary to automate the process of paging through and acquiring all the desired data; relatively quickly and efficiently. An advantage of using Excel is its relative ease of use and a high degree of familiarity among many business professionals and programmers, and relative familiarity among students. Excel is an excellent tool for analyzing data, including charts, sorting, filtering, and data modeling, among many used. Additionally, Excel has VBA programming capabilities that allow one to use a subset of Visual Basic to automate and manipulate Excel and other Microsoft applications, including access to the COM and ActiveX Objects, as well as a multitude of built-in functions.

This case describes the methodology for programmatically extracting hundreds and thousands of historical stock price data over hundreds of webpages from Yahoo's Finance website using the Excel Web Query tool. The stock price data example is being used since the website structure accommodates use of Excel in the manner described, and the task is very similar to how one would go about extracting data from numerous other structurally similar websites. Additionally, Yahoo has no express policies preventing programmatic data extraction. As such, Section 2 describes the manual technique for extracting a single page of records, while Section 6 describes the fully automated programmatic approach returning thousands of records across hundreds of webpages. Section 3 discusses webpage structure which is essential in the automation process, and Section 4 discusses discovery in the context of Yahoo's Finance webpage structure. Section 5 provides the necessary overview of Excel and VBA required to automate the data extraction. One should note that the Yahoo Finance historical prices webpage has a downloadable csv file containing all the data over the specified period of data. Nevertheless, this case is to illustrate a program when no such file or other data extraction technology is readily available.

EXTRACTING DATA USING THE EXCEL WEB QUERY TOOL

The Excel Web Query (WQ) tool facilitates bringing data from a website into an Excel worksheet. Web queries are an easy, built-in way to bring data into Excel from the Web. The WQ tool allows the user to point a web query at an HTML document that resides on a Web server and pull part or all of the contents into your spreadsheet. The WQ tools can also retrieve refreshable data that is stored on the Internet, such as a single table, multiple tables, or all of the text on a webpage (Import external data from a complex web site into Excel, n.d.). The tool is based on discovering HTML tables on the webpage and allowing the user to select the table(s) containing the data that is desired to download. In its simplest deployment, one initiates a new web query in Excel and enters a uniform resource locator (url) into the WQ address field. The WQ navigates to the webpage and displays an icon image beside all HTML tables in the webpage. The user can select one or more tables by clicking the icon images. The text from all selected tables is downloaded and displayed in the Excel worksheet. A more extensive description of the WQ tool is available by Rice, F. (2004). As a quick example we will use a web

query to download one page of historical stock price data for Wal-Mart® from Yahoo Finance. The Wal-Mart historical prices url is http://finance.yahoo.com/q/hp?s=WMT+Historical+Prices. The reader is encouraged to engage the steps below. Perform the following steps.

Step 1. Open a new Microsoft Excel blank workbook and position the curser in cell A1.

Step 2. Follow the Excel menu path given as Data > Get External Data > From Web. The web query tool will open.

Step 3. When the WQ tool opens type the url given above into the "Address" field and click the "Go" button. It is common that a Script Error dialogue box will open, but the user can continue to click the "Yes" button until it disappears. Note the screen as shown in Figure 1 (Appendix A). Pay particular attention to the yellow boxes with red arrows beside the tables that have been detected.

Step 4. Click the yellow box beside the column labeled "Date" as shown with the arrow in Figure 1 (Appendix A).

Step 5. Click the "Import" button at the bottom of the WQ tool.

Step 6. When prompted by the dialog box named "Import Data," keep the default selections (Figure 2, (Appendix A)) and click "OK." The data from the tables is imported into the worksheet as shown in Figure 3 (Appendix A).

The result is 66 rows of historical stock prices, one for each of the latest 66 trading days. If one observes the actual webpage there are navigation buttons (First, Previous, Next and Last) to allow displaying prices over additional periods.

Unfortunately, the WQ tool used in this manner requires manual user intervention to enter the url, set the options and properties, and execute the data import. This paper illustrates the programmatic automation of this process for multiple urls obtained via sequential paging (pagination). In this case it will automate the above steps for every webpage of price data; not just the latest 66 days of trading. Although the Internet is full of tutorials on using the WQ tool for simple webpage retrieval (as described above), extensive searches have found no literature or tutorials for fully automating the WQ tool, including sequential pagination required for web scraping. For more discussion and example of simple web queries, the interested reader is encouraged to see the following; Get external data from a Web page (n.d.); Pieterse, J. (2006); Pull data into Microsoft Excel with Web queries – TechRepublic (2006); Wittwer, J. (n.d.).

WEBPAGE STRUCTURE

Determining the structure of the webpage, the HTML tables, and the actual data is often the first step in the programmatic approach. This step is called discovery and it may include several different phases. Some discovery is manual in nature, like clicking through a website, observing url patterns and paging parameters, noting table names and structures, viewing the category and paging link urls, viewing the page's source code, etc. Other discovery can be accomplished with a few button clicks in Excel. Some discovery might use a combination of both.

The specific software being used to display webpages on a particular website often determines the approach one will use to programmatically page through the website and extract the data. Since urls are an important component of the program, it is important to determine if server-side programming or client-side scripts are making the calls to the actual database server. One might observe that on some websites each subsequent webpage has a unique url in the browser address bar, while others show no change in the url, regardless of the webpage. In the former case, it is most likely that client-side scripts are calling the database and displaying the results in a webpage's tables using the GET method of an HTML form element. In the latter case, one has likely encountered server-side processing common in ASP and ASP.Net applications, wherein the script on the page executes a server-side program that makes a call to the database and then binds the data to the same webpage, hence no change in url. In this case the HTML form element uses the POST method. A quick overview of the GET and POST method is available through a standard Internet search, including HTTP methods: GET vs. POST (n.d.), while a more descriptive overview is available by Korpela, J. (2003).

A good way to determine if the GET method is being used is to observe the url over several sequential catalog pages. If the url contains "?", "&" and/or "=" symbols, and sequential paging results in subtle changes in some of the url's string, then the GET method is being used. The "?" indicates a query, the "&" indicates concatenation of a query string, while "=" sets parameters in the query string to query values. Another common design may use either method, but then nests the webpages within frames, so that the visible url remains the same, and only the main frame contents seem to change.

Typically, the method used (GET versus POST) determines how the tables are fetched into Excel, what the data structure and format will be, and the required parameter settings for the WQ url. For example, if each webpage's url structure and parameters are known, one can simply code a For-Next loop in a VBA subroutine to use WQ to fetch each page's desired data according to the url's changing parameters. In any case, it is important to have some knowledge of data retrieval using the HTML DOM since HTML formatted data is being extracted. A comprehensive overview of the HTML DOM can be located at Document Object Model (2015) and The HTML DOM Document Object (n.d.).

Regardless of the webpage structure, the main objective of the program is to acquire the desired data from the tables in each of multiple webpages. The program may also facilitate automated cleaning of dirty data, editing the data, and structuring the data for upload to a database or distribution. The following section demonstrates the required discovery of the structure of the Yahoo Finance website from which we wish to extract the historical stock price data across many pages of data. Again, this website is chosen because it closely resembles the structure of many other websites that might be of interest, and it is also among the least complicated of websites from which to extract data. The reader is encouraged to engage with the discussion provided below to more clearly comprehend the environment and processes.

THE DISCOVERY PROCESS FOR THE YAHOO FINANCE WEBSITE

The Yahoo Finance page located at http://finance.yahoo.com allows a user to manually input a stock symbol to return summaries, charts, and historical price data (among other data) for the selected stock. The historical prices are of interest in this discussion. To obtain the historical prices for Wal-Mart, one would navigate to the above url and then enter the Wal-Mart stock symbol for a quote; WMT. When the subsequent webpage for Wal-Mart stock data opens, one

clicks the hyperlink titled "Historical Prices" in the left pane of the webpage, as shown in Figure 4 (Appendix A). The result is a webpage with a form that allows the user to select the date range as well as an option for daily, weekly, or monthly stock data (prices and volume). The webpage is initially loaded with the historical prices of the last 66 trading days. Figure 5 (Appendix A) is a partial screen-shot of the date range and options fields, as well as stock price data displayed in a table. Note the dates shown in Figure 5 (Appendix A), as this is the time period over which the data will be returned.

As an example of discovery via webpage navigation, if one observes the initial url in the web browser address field and the urls of subsequent pages (using the "Next" navigation hyperlink) it is obvious the url structure that will be required to programmatically access each subsequent webpage of data. The "Last" navigation hyperlink is also important, as it displays the total number of records of historical price data over the selected date range. Using the default dates in the "Set Date Range" fields of the webpage and clicking the "Get Prices" button, one can view the initial url in the browser's address field. Note the initial url below, and that the default "End Date" is Oct 23, 2015, the date on which this example was written.
http://finance.yahoo.com/q/hp?s=WMT&a=07&b=25&c=1972&d=09&e=23&f=2015&g=d

The initial url path is also part of the url for all subsequent pages, reflecting the selected stock symbol query parameter (s=WMT), concatenations (&), date query parameters and values reflecting the dates shown in Figure 6 (Appendix A), (a=7, b=25, c=1972, d=10, e=15, f=2015), and the optional parameter for daily prices (g=d). The parameters a, b, and c are the starting month, day and year parameters, while the parameters d, e, and f are the ending month, day, and year values. Note that due to indexing set by Yahoo that each month value is one month behind. For example, "a=07" in the above url is August, and "d=09" is October. When one clicks on the "Next" navigation button three times, the subtle change in the subsequent urls include the following paths appended with the initial url as shown below. The last url shown below results from clicking the "Last" navigation button.
Page 2 url: &z=66&y=66, Page 3 url: &z=66&y=132, Page 4 url: &z=66&y=198, Last Page url: &z=66&y=10824

The obvious structure is that each webpage displays 66 historical stock price records in the price table. The parameter **z** indicates the number of records per table, and **y** indicates the ending value of the next set of 66 records. If one divides the last page url value of y=10824 by z=66, the resulting value is 164, indicating 164 additional pages beyond the initial page; 165 pages total. Additionally, the initial url can have the following appended without affecting the url; **&**z=66&y=0.

The key here is to first determine the number of pages of historical price data for any specified stock symbol, hence enabling the programmatic looping required for the WQ tool to acquire the table on each webpage. For example, in this case we need a loop to iterate from i = 0 to 164 in increments of i*66, emulating urls ending in y=0, y=66, y=132, y=198, … y=10824. The static url is as follows.
http://finance.yahoo.com/q/hp?s=WMT&a=07&b=25&c=1972&d=09&e=23&f=2015&g=d&z=66&y=
So, the url for input into the WQ tool is the static URL**,** while the dynamic path component is simply an appended value of **y**, like &y=0, &y =66, … &y=10824. Obviously, a different quote symbol and different dates will have different parameter values.

Another method of discovery involves viewing the page source for the webpage, which provides a view of the webpage's source code. Although the initial results appear as an

entanglement of undecipherable source code, there is valuable information in the code. For example, after displaying the historical prices webpage for WMT, one can right-click anywhere on the page and select "View page source." Navigating down to around line number 230 one can observe the urls for the "Next" and "Last" pages of historical data, as shown below.
href="/q/hp?s=WMT&a=07&b=25&c=1972&d=09&e=23&f=2015&g=d&z=66&y=66
href="/q/hp?s=WMT&a=07&b=25&c=1972&d=09&e=23&f=2015&g=d&z=66&y=10824"

Note that although not shown in the code, the url of the first page ends in "&y=0". Again, dividing 10824 records by 66 records per page results in 164 pages, plus the additional first page; a total of 165 pages of records. This information will later be programmatically extracted for automation of the WQ tool. Additionally, one can use the browser's "Inspect element" tool to determine the next and last webpage urls. For example, while in webpage view, one may right-click on the "Next" navigation button, select "Inspect element," and notice the same url as shown above; likewise, for inspecting the "Last" navigation button. The following section provides an overview of Excel macros and the VBA programming language in regards to automating the WQ tool for data extraction from the Yahoo finance website.

UNDERSTANDING EXCEL MACROS AND VBA

The ultimate goal of the automation is to, for a given stock symbol, programmatically extract every historical price summary across every webpage and store them in an Excel worksheet. This involves several programmatic steps, including, extracting, and recording the necessary parameters and values from the source code urls, calculating the number records, records per page, and of pages of data, automating the WQ tool to use the parameters to download the desired data, and then clean the data of unwanted data (dirty data).

This paper illustrates the implementation of the WQ tool using Microsoft Excel 2016, but there are no notable differences that would affect the implementation using previous versions of Excel, as far back as 2007. High-end users of Excel may be more familiar with recording macros and programming VBA in the Visual Basic Editor (VBE), but basic knowledge in each of these is required to automate Excel in the fashion described in this paper. A more extensive treatment of using VBA in Excel can be found at Korol, J. (2014) and Walkenbach, J. (2013). At a minimum, the reader must be familiar with Excel macros and VBA, so we define the following in laymen's terms.

- Macro - A macro is a program that store a series of commands that one might execute in Excel. It is the simplest form of automation, showing Excel the steps to follow to accomplish various tasks. A macro runs within Excel and automates repetitive tasks within the spreadsheet environment. Macros can be created by using Excel's built-in recording tool (Macro Recorder) or written using the VBE in Excel. Macros are created with the programming language VBA.

- VBA – VBA is an implementation of Microsoft's event-driven programming language, Visual Basic 6, enabling building functions and subroutines to automate processes in Excel (and other Microsoft applications). It can be used to control many aspects of the host application (Excel in this case), including manipulating worksheets, ranges, cells, and user interface features such as menus and toolbars, working with custom user forms or dialog boxes, interacting with other Microsoft and Windows applications, and interacting with the

web (among other aspects). VBA runs within Excel rather than as a stand-alone program (Visual Basic for Applications, n.d.).

In general, macros are recorded by the user, while subroutines are written using VBA. When a macro is recorded, the instructions are automatically created inside of a module using a VBA subroutine. In either event, the VBE allows one to edit existing macros or to create subroutines within the module. The only real difference between a macro and a subroutine is how they are created; either recorded in Excel or written by the user. Following the creation of a macro and/or subroutine, it can be run (executed) using a variety of methods, including key-board commands, button clicks, dialogue boxes, user forms, etc. They can also be run directly from the VBE using the run icon or the F5 key.

The Excel 2016 menu contains main menu tabs that house groups of related tools in a ribbon. In the case of macros and the VBE, the menu tab named "Developer" houses the required ribbons and tools. Since it is a less common tab option it is not typically included in the default tab list when Excel is installed. In this case, the user must customize the main menu to include the Developer tab. Appendix B describes the steps to include the Developer tab and provide access to macros and the VBE. As previously stated, all VBA code is written within a module in the VBE. Appendix C describes using the VBE to create a module and start programming a subroutine. The following section discusses programmatic automation of the WQ tool to enable data extraction.

AUTOMATING THE WEB QUERY FOR DATA EXTRACTION

To prepare any web data extraction program, a user specifies a starting URL, a crawling rule (pagination) and content or page HTML elements to collect. The program then goes through all available pages and collects data as a human would do manually using mouse clicks and copy-and-paste. This paper describes the programmatic web query to return all historical stock price summaries across all webpages of data. As such, the program includes five tasks as shown below, with each task consisting of a VBA subroutine. Note that the dates in the urls and figures are those when the paper was written and will change daily.

Task 1: Extract Webpage URLs

Task 2: Extract Page Navigation URLs

Task 3: Extract URL Parameters and Values

Task 4: Run the Web Query and Download Stock Price Data

Task 5: Clean the Data

Following completion of all five programming tasks, the subroutines are combined into one subroutine that calls the five subroutines and runs them in sequence by clicking a button on an Excel worksheet. Although all five subroutines could easily be combined and written into only one, they remain separate in this paper for the sake of clarity of each task. A discussion of all five tasks follows. Note that for the sake of brevity the VBA code does not follow best practices of input data validation, the explicate declaration of variables, code for exception and

error handling, or code commenting. Instead, each subroutine is shown in a table with a description of the subroutine discussed in context of the task, while a subsequent table contains code descriptions. Furthermore, in many cases, what would normally be separate lines of code are combined into one line using the colon symbol "**:**" to break the code execution apart but consolidate it on one line.

Before writing the subroutines one must first setup the Excel workbook. To do so complete the following steps. Note that in step 1 the worksheet must be saved as shown else the VBA code will not run.

Step 1. Open a new Excel workbook, save with the name "StockDataWebQuery," and save as an "Excel Macro-Enabled Workbook."

Step 2. Create and name three worksheets using the names "IO," "URLs" and "Prices." Worksheet names can be changed by right-clicking the worksheet tab and selecting "Rename."

Step 3. Format the first worksheet named "IO" exactly as shown in Figure 6 (Appendix A), and save the workbook.

Note that the worksheet named "IO" is for initial input/output associated with first four subroutines, "URLs" is used to store and manipulate the required urls used in the first three subroutines, and the "Prices" worksheet will store the final historical stock price data records generated by the forth subroutine (which is the actual automated WQ). The only user input required in the worksheet is typing a stock symbol into Cell "B1" on the "IO" worksheet. The other cells are set programmatically and will provide required inputs to various subroutines. The cell data will be described in the context of the discussion of each task and related subroutine.

Task 1: Extract Webpage URLs

Recall from the section on the discovery process that one can determine the url that contains the parameters and values needed to run the web query by either of visually inspecting the address bar url or using the "Inspect Element" tool for the **Last** navigation button, or by viewing the webpage source code. The url provides the query parameters (a, b, c, …y) and well as the values (07, 25, 1972, …10824). Note that these parameter values will change daily. The required url follows.
http://finance.yahoo.com/q/hp?s=WMT&a=07&b=25&c=1972&d=09&e=23&f=2015&g=d&z=66&y=10824

The first task then is to programmatically extract the url containing the parameters and values from the Yahoo Finance webpage for the selected stock. By viewing the source code for the webpage one would notice many urls in the code (over 100). The required url above is just one of these. Furthermore, one would notice that each url in enclosed in a tag, given by <a>. Fortunately, the HTML DOM "getElementsByTagName" method is available to programmatically extract all urls into a worksheet. The method will extract all tags named "a" which is an anchor object in the HTML DOM (HTML DOM getElementsByTagName() Method, n.d.). Again, discovery of the tag name was required by viewing the source code of all urls on the initial webpage; for example <a href="http:\\ ….

So, the program must first determine the initial url of the desired webpage, open an Internet Explorer browser session, navigate to the webpage, and then extract all the urls into the worksheet named "URLs." Note that before running the subroutine the user must first enter a stock symbol in the "IO" worksheet in cell "B1." In this example the code is extracting price data for Wal-Mart, whose stock symbol is WMT. After writing the code, entering the stock symbol and running the code, the user can observe the initial url in the "IO" worksheet (among the other extracted urls in the "URLs" worksheet). The subroutine can be run in VBE by pressing the F5 key or by clicking the run icon on the VBE toolbar. Figures 7 shows the "IO" worksheet with the initial url programmatically generated and stored "B2." Figure 8 (Appendix A) shows rows 109 through 114 of the "URLs" worksheet, wherein the shaded region contains the actual urls that will later be extracted. Note that the month/day/year parameter/values are reversed, but this is immaterial. The code for the first task is shown in Table 1 (Appendix A), while the description of the code is shown in Table 2 (Appendix A).

Task 2: Extract Page Navigation URLs

The urls resulting from Task 1 (Figure 8 (Appendix A)) will later be used to allow extraction of the required parameters and values in Task 3. Instead of directly pulling out the urls though, it is easier to delete the unwanted urls. Again, it is important to note by inspection that all four of the urls in Figure 8 (Appendix A) have the common url component "http://finance.yahoo.com/q/hp?s=". Hence, the subroutine in Task 2 will simply delete every row on the worksheet where the url doesn't have the same first 30 characters from the left shown in the url above. Again, the code is shown in Table 3 (Appendix A) and described in Table 4 (Appendix A). The result is the four shaded urls shown in Figure 8 (Appendix A).

Task 3: Extract URL Parameters and Values

The primary objective of Task 3 is to create the static url component required in the automated web query in Task 4. The url below is the required static url component.
http://finance.yahoo.com/q/hp?s=WMT&a=07&b=25&c=1972&d=09&e=23&f=2015&g=d&z=66&y=
The result is the static url component using the extracted parameters and values, as well calculations for the total number of records, the number of records per page, and the total number of pages of records to be downloaded. These calculated values are also used in Task 4.

Notice that the parameter **y** in the static url above has no assigned value (as shown in Task 1), as it will be dynamically appended to the web query in the code. Also, note that of the four urls remaining after Task 2, the first and third urls are the only ones that are actually needed to extract the required parameters and values. The first and second are identical while the third and fourth are identical. The first url will be retained to extract the parameters (a, b, c, …y), while the third url will be retained to extract the parameter values. As such, the code first uses the VBA "RemoveDuplicates" method to remove the unwanted duplicate urls, and then uses the VBA "TextToColumns" method to isolate both the parameter/value combinations After isolating the combinations, a VBA "Replace" method is used to replace unwanted characters with an empty string, leaving only the desired parameters and values. These parameters and values are copied from the "URLs" worksheet and pasted into the appropriate cells in the "O" worksheet. They are then used to create the static url. The code is shown in Table 5 (Appendix A) and

described in Table 6 (Appendix A). Figure 9 (Appendix A) reflects the" URLs" worksheet after the code in line 2 executes, while Figure 10 (Appendix A) reflects the worksheet following execution of lines 3-7. Finally, Figure 11 (Appendix A) reflects the "IO" worksheet after the remaining code has been executed.

Task 4: Run the Web Query and Download Stock Price Data

Task 4 will use the static component of the url in Figure 11 (Appendix A) with the dynamically changing value of **y** and the total number of pages to programmatically iterate through the webpages and download the data from each webpage. Recall from discussion in the discovery process how the WQ tool was used to download the first set of records. If a macro is recorded using the WQ tool, the VBA WQ output will appear as shown in Appendix D, Table 12 (Appendix A). Note that the actual WQ is executed in lines 1 and 2, while the only other required VBA code is in lines 18, 24 and 25. The other lines of code are option, are outside of the purview of this paper, and are omitted from the code. It should be noted that including line 3 will create a run-time-error and prevent the code from running. Hence, the code for Task 4 uses only the required lines of the WQ code. Note also that line 18 reflects that the selected HTML table is "15." If one were to view the source code of the webpage and count the tables, the desired data is indeed in table 15. The code is shown in Table 7 (Appendix A) and described in Table 8 (Appendix A).

The VBA for the WQ requires both a url and the destination cell for the output, as shown in line 9 of Table 7 (Appendix A). Both the url and destination will change for each iteration (i) of the For-Next loop. The variable "WqURL" in line 5 is simply a static string that is set programmatically, while the variable "DynamicWqURL" is the programmatically set connection string url required in the WQ, which is a dynamic string resulting from concatenating the static string with the dynamically changing value of y. Recall that y is the next set of 66 records. For example, when i=0, the WQ navigates to the url ending in y=0*66=0 (the first page). When i=1, the url ends in y=1*66=66 (the second page), and so on, until i reaches its upper bound (total number of pages -1). Recall, the total number of pages is stored in the "IO" worksheet, cell "J7." For subroutine testing purposes the upper bound variable can be replaced with a static value (like 2) to extract only a few pages of data. For example, line 7 can be changed to "For i = 0 To 2 'TotalPages – 1" where the apostrophe following the value 2 is simply commenting out the upper bound.

Regarding the destination, this is the cell in the worksheet where the stock price output is to be stored. A variable named "LastRow" is used to determine the last used row in the worksheet to know where to store the next set of records. For example, for iteration i=0, the LastRow value is 1, and the code in line 12 tells the WQ to store the records starting in cell "A2," which in the code is the row and column given by "row=LastRow+1", "column=1." Figure 12 (Appendix A) reflects the truncated output after only 2 iterations; i=0 and i=1. Note that the row 1 is empty, row 2 contains the output headers, row 60 is non-trading data regarding a dividend distribution (dirty data), row 70 is the footnote for the asterisk in the last column (dirty data), and row 71 is header for the second set of records (dirty data). After a set of records is downloaded, the "LastRow" value is recalculated returning the row number containing the footnote, and the row is deleted (the footnote row). For the next iteration, "LastRow" is calculated again and the next set of records is stored in the first available row with no data (LastRow +1). Note that the other dirty data issues will be addressed in Task 5.

Task 5: Clean the Data

The final task is to clean the worksheet of the unwanted rows of data. The result is that the row 1 (which is empty) will be deleted, making the headers the first row, and every row that contains indications of a dividend or unwanted additional headers will be deleted. This is quite easy considering that all the desired data in the worksheet is either a date or a value, and unwanted data will be text. Hence, the subroutine removes row 1 (making row 2 headers the new row 1), iterates across each row (from "LastRow" value up to row 2), and uses an "If" statement to determine if the data in column B of the selected row contains text. If it contains text, then the entire row is deleted. Column B is checked since the dividend data is always contained in column B, and the unwanted additional headers will always exist in column B. The code is shown in Table 7 (Appendix A) and described in Table 8 (Appendix A).

Finally, the 5 separate subroutines are combined into a single subroutine as shown in Table 11 (Appendix A). Although the subroutine can be run directly from the VBE using the run icon or pressing F5, a command button can be added to the "IO" worksheet to run without the VBE being open. Appendix E provides the steps to add the button and assign the subroutine are provided below.

ADDITIONAL CONSIDERATIONS

Several other considerations should be noted regarding the method demonstrated in this paper. The main consideration is that if the program is structurally dependent, then it will not run if the structure of the url changes. In the case of the program in this paper, if the Yahoo Finance url changes then the user must discover the new url and parameters and modify the code in several places. Additionally, this method depends heavily on the data being structured in an HTML table element, as was seen in this example where the webpage had many tables but we wanted to only extract data from table 15. There are cases (such as eBay) wherein the entire page is encapsulated in a single table and parsing and cleaning the desired data requires additional VBA code. The WQ tool is also browser dependent, meaning the WQ uses Microsoft Internet Explorer as the browser application, and the latest version should be installed on the user computer.

There are many cases where it is not feasible to automate the WQ tool. If a webpage stores session variables, then the variable values are usually different each time the webpage is called. Session variables are parameters that are stored on the server and not displayed into the source of the webpages. Hence, it may be impossible to create a web query unless the user knows the value of the session variables ahead of time, which is not usually the case. And while websites using the GET method retrieves webpages based on the url concatenated with the parameter query string (as is the case in this paper), the POST method requires a different process that requests the webpage by sending the query string parameters to the server, which may not always readily be accomplished using the WQ tool without significant modification to the code. In this case the user may have to create and automate an internet query (iqy) file (How to Create Web Query (.iqy) Files, 2003). Additionally, there is no frames support for the WQ tool, and sites requiring authentication and passwords present additional challenges of writing workaround code.

There are often ethical and legal considerations to be made when web scraping in general. Some websites may strictly prohibit it, and while others may not, it may create an undue heavy burden on the server from which webpages are extracted. FaceBook® and Google® strictly prohibit any programmatic approach to extract data from their webpages and will take measures such as closing an account or blocking an IP to prevent web scraping. There are also many legal cases regarding the use of web scraping (Web scraping, 2015).

CONCLUSION

Web scraping allows extraction of data from the web. There are many techniques available for website data extraction, including website provided APIs, web services, RSS feeds, and file downloads, among others. Web scraping software is also available as freeware or paid subscription, and one can even developed their web scraping software. This paper has shown that when other technologies and software are not readily available, the automation of the Excel WQ tool can facilitate data extraction from the Web. Additionally, using Excel as the data repository and VBA allows editing and structuring of the extracted data, as well as cleaning of dirty data.

APPENDIX A

Figure 1. Web Query Dialogue

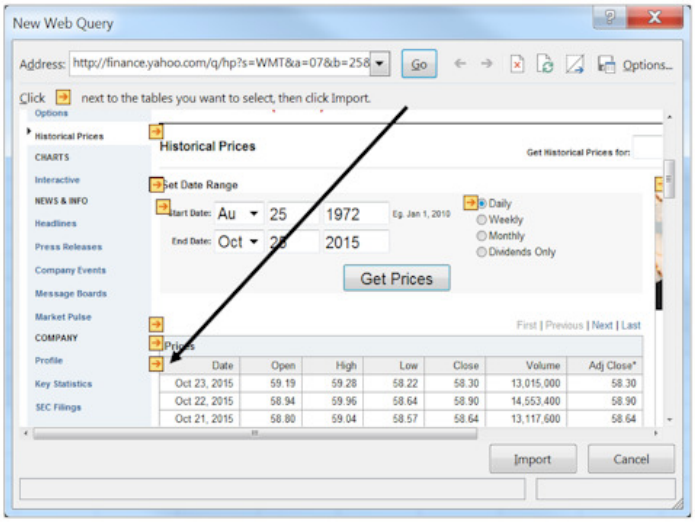


Figure 2. Import Data Dialogue

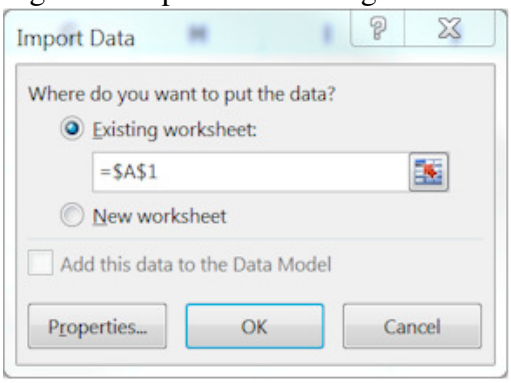


Figure 3. Imported Data

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Volume	Adj Close*
2	23-Oct-15	59.19	59.28	58.22	58.3	13,015,000	58.3
3	22-Oct-15	58.94	59.96	58.64	58.9	14,553,400	58.9
4	21-Oct-15	58.8	59.04	58.57	58.64	13,117,600	58.64
5	20-Oct-15	58.86	59	58.57	58.75	10,264,200	58.75
6	19-Oct-15	58.79	59.3	58.5	58.85	17,685,800	58.85
7	16-Oct-15	59.47	59.49	58.37	58.89	25,860,500	58.89
8	15-Oct-15	59.7	60.47	58.61	59.33	45,746,400	59.33
9	14-Oct-15	66.61	67.95	60.02	60.03	80,604,500	60.03
10	13-Oct-15	66.62	66.94	66.26	66.73	8,803,100	66.73
11	12-Oct-15	66.67	67	66.58	66.93	5,629,300	66.93
12	9-Oct-15	66.94	67.02	66.51	66.69	6,659,100	66.69
13	8-Oct-15	66.23	66.99	66.15	66.88	5,939,400	66.88

Figure 4. Historical Prices

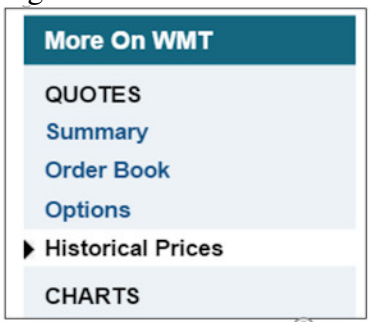


Figure 5. Date Ranges and Options

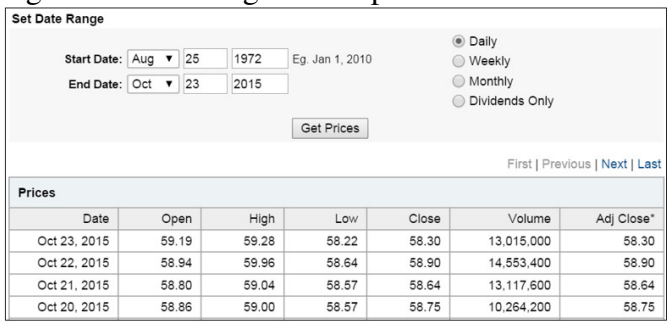


Figure 6. Formatted IO Worksheet

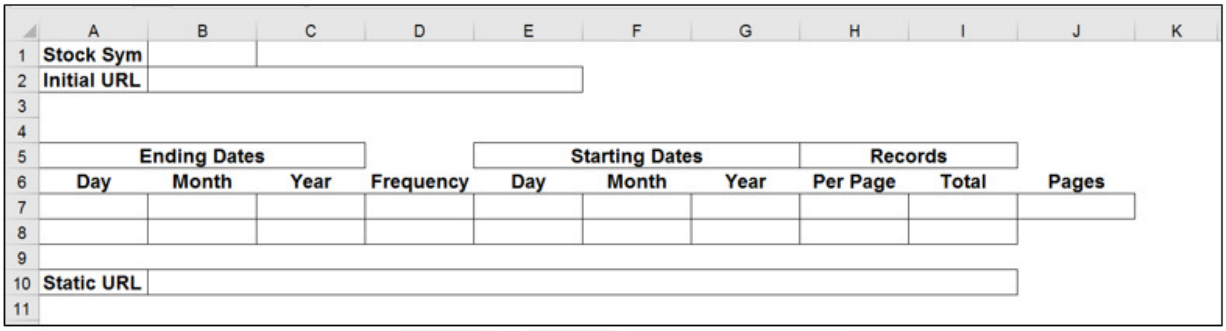


Figure 7. IO Worksheet with Initial URL

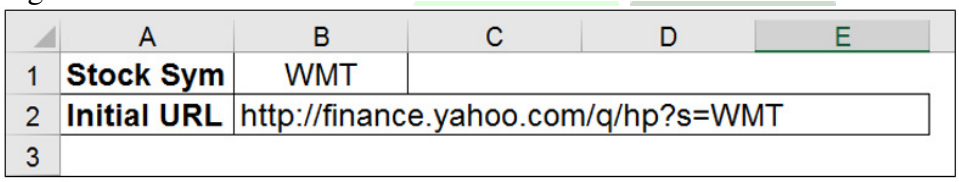


Figure 8. URLs Worksheet

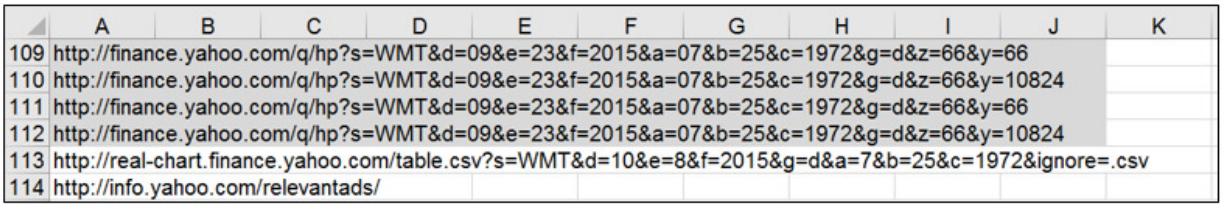


Figure 9. URLs Worksheet After Line 2 Code Execution

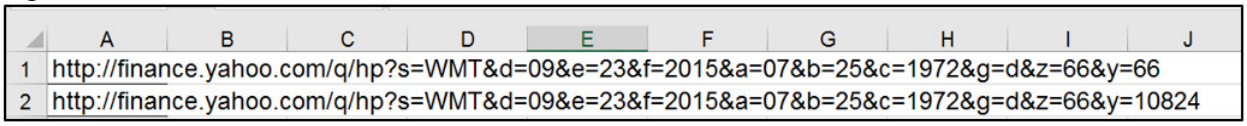


Figure 10. URLs Worksheet After Lines 3-7 Code Execution

	A	B	C	D	E	F	G	H	I	J
1	d	e	f	g	a	b	c	z	y	
2	9	23	2015	d	7	25	1972	66	10890	

Figure 11. IO Worksheet After All Lines of Code Executed

	A	B	C	D	E	F	G	H	I	J
1	**Stock Sym**	WMT								
2	**Initial URL**	http://finance.yahoo.com/q/hp?s=WMT								
3										
4										
5	**Ending Dates**				**Starting Dates**			**Records**		
6	**Day**	**Month**	**Year**	**Frequency**	**Day**	**Month**	**Year**	**Per Page**	**Total**	**Pages**
7	d	e	f	a	b	c	g	z	y	165
8	9	23	2015	7	25	1972	d	66	10824	
9										
10	**Static URL**	http://finance.yahoo.com/q/hp?s=WMT&d=9&e=23&f=2015&a=7&b=25&c=1972&g=d&z=66&y=								

Figure 12. Truncated Output After 2 Iterations

	A	B	C	D	E	F	G
1							
2	**Date**	**Open**	**High**	**Low**	**Close**	**Volume**	**Adj Close***
3	23-Oct-15	59.19	59.28	58.22	58.3	13,015,000	58.3
60	5-Aug-15	0.49 Dividend					
61	4-Aug-15	72.42	72.8	71.93	72.25	5,873,400	71.76
62	3-Aug-15	71.84	72.38	71.84	72.18	5,131,800	71.69
63	31-Jul-15	72.44	72.49	71.66	71.98	7,928,400	71.49
64	30-Jul-15	72.03	72.53	71.78	72.16	4,604,500	71.67
65	29-Jul-15	72.25	72.63	72.09	72.23	4,939,700	71.74
66	28-Jul-15	71.53	72.39	71.23	72.1	8,592,300	71.61
67	27-Jul-15	71.38	71.65	71.01	71.38	6,197,600	70.9
68	24-Jul-15	72.53	72.56	71.5	71.58	5,951,100	71.09
69	23-Jul-15	73.07	73.22	72.42	72.51	4,252,900	72.02
70	* Close price adjusted for dividends and splits.						
71	**Date**	**Open**	**High**	**Low**	**Close**	**Volume**	**Adj Close***

Table 1. Task 1 VBA Code

Line	Code
1	Sub Extract_Webpage_URLs()
2	Sheets("URLs").Cells.Clear: Sheets("Prices").Cells.Clear
3	Sheets("IO").Select: Range("A7:J8").ClearContents: Range("B10").ClearContents
4	Range("B2") = "http://finance.yahoo.com/q/hp?s=" & Range("B1")
5	InitialURL = Range("B2")
6	Sheets("URLs").Select
7	Dim WebPage As Object
8	Set WebPage = CreateObject("InternetExplorer.Application")
9	WebPage.Visible = False
10	WebPage.NAVIGATE InitialURL
11	Application.Wait (Now() + TimeValue("0:00:10"))

12	NumURLs = WebPage.document.getElementsByTagName("a").Length
13	For i = 1 To NumURLs - 1
14	Cells(i, 1) = WebPage.document.getElementsByTagName("a").Item(i)
15	Next
16	WebPage.Quit: ActiveWorkbook.Save
17	End Sub

Table 2. Task 1 VBA Code Description

Line	Code Description
1	Create the subroutine named **Extract_Webpage_URLs**.
2	Clear the previous contents of the **URLs** and **Prices** worksheets.
3	Select the **IO** worksheet and clear the contents of previous cells and ranges.
4	Set the Initial URL in cell **B2** to the URL "http://finance.yahoo.com/q/hp?s=" concatenated with the stock symbol in cell **B1**.
5	Name a new variable **InitialURL** and set it to the url stored in cell B2.
6	Select the **URLs** worksheet.
7	Declare and name a new object **WebPage**.
8	Set the WebPage object to Internet Explorer (**IE**) browser application.
9	Hide the IE application (browser session) when it opens.
10	Navigate to the Initial URL (**InitialURL**) that was set in line 5.
11	Wait 10 seconds while the page loads (otherwise the application may time out).
12	Name a new variable **NumURLs** to count how many urls are in the webpage using the HTML DOM method (*getElementsByTagName*("a")) and the *Length* method.
13	Set a **For-Next** loop to iterate through all the urls in the webpage; from i = 1 to NumURLs-1.
14	For each iteration (i), use the HTML DOM method (*getElementsByName*) to extract each url from the webpage and copy the extracted urls into the sequential rows of column A in the **URLs** worksheet.
15	Continue the Next iteration; i.
16	Close the IE browser application and save the workbook.
17	End the subroutine.

Table 3. Task 2 VBA Code

Line	Code
1	Sub Extract_Page_Navigation_URLS()
2	Sheets("URLs").Select
3	With ActiveSheet: LastRow = .Cells(.Rows.Count, "A").End(xlUp).Row: End With
4	For i = LastRow To 1 Step -1
5	If Left(Cells(i, 1), 30) <> "http://finance.yahoo.com/q/hp?" Then
6	Rows(i).EntireRow.Delete
7	End If
8	Next
9	ActiveWorkbook.Save
10	End Sub

Table 4. Task 2 VBA Code Description

Line	Code Description
1	Create the subroutine named **Extract_Page_Navigation_URLS**.
2	Select the **URLs** worksheet.
3	Name and set a new variable **LastRow** and use a VBA method (*Cells(.Rows.Count, "A").End(xlUp).Row*) to count the number of rows in the worksheet.
4	Set a **For-Next** loop to iterate through all the urls in the webpage; from i = LastRow to 1 (reverse order) in increments of 1.
5-7	For each iteration, use an **If** function to delete the rows using a VBA method (Rows(i).EntireRow.Delete) for any url that does not equal "http://finance.yahoo.com/q/hp?" when counting from the left of the url, using the VBA Left function.
8	Continue the **Next** iteration; i.
9	Save the workbook.
10	End the subroutine.

Table 5. Task 3 VBA Code

Line	Code
1	Sub Extract_Parameters_Values()
2	Range("A:A").RemoveDuplicates Columns:=1
3	Range("A:A").Select
4	Selection.TextToColumns Destination:=Range("A1"), Other:=True, OtherChar:="&"
5	Rows(1).Select: Selection.Replace What:="=*", Replacement:=""
6	Rows(2).Select: Selection.Replace What:="*=", Replacement:=""
7	Columns(1).EntireColumn.Delete
8	Range("A1:I2").Copy: Sheets("IO").Select
9	Range("A7").Select: Selection.PasteSpecial Paste:=xlPasteValues
10	Range("J7") = (Range("I8") / Range("H8")) + 1
11	ParameterString = ""
12	For i = 1 To 8
13	ParameterString = ParameterString & "&" & Cells(7, i) & "=" & Cells(8, i)
14	Next
15	Range("B10") = Range("B2") & ParameterString & "&y="
16	Range("B1").Select: ActiveWorkbook.Save
17	End Sub

Table 6. Task 3 VBA Code Description

Line	Code Description
1	Create the subroutine named **Extract_Parameters_Values**.
2	Select the **URLs** worksheet and remove duplicate rows using the **RemoveDuplicates** method.
3	Select column A.
4	Use the **TextToColumns** method to parse parameter/value combinations into adjacent columns using the “&” delimiter.
5	Select row 1 and replace text containing the equal sign and anything to the right of it with an empty string. This results in the parameters.

6	Select row 2 and replace text containing the equal sign and anything to the left of it with an empty string. This results in the parameter values.
7	Delete the first column (column A).
8	Copy the range of data in the **URLs** worksheet and select the **IO** worksheet.
9	Paste the data in the range in the **IO** worksheet.
10	Calculate the total number of pages as the total records divided by records per page.
11	Name a new variable **ParameterString** and set it to an empty string.
12	Set a **For-Next** loop to iterate through the 8 parameters and values; i = 1 to 8.
13	For each iteration, concatenate the parameter/value combinations from (row 7, column i) and (row 8, column i).
14	Continue the Next iteration; i.
15	Create the Static URL in cell B10 using the Initial URL in cell B2, concatenated with the complete parameter string from lines 11-13, concatenated with y=. Select cell B1 and save the worksheet.
16	End the subroutine.
17	

Table 7. Task 4 VBA Code

Line	Code
1	Sub Get_Stock_Data()
2	Application.ScreenUpdating = False: Sheets("IO").Select
3	RecordsPerPage = Range("H8"): TotalRecords = Range("I8")
4	TotalPages = Range("J7"): InitialURL = Range("B10")
5	WqURL = "URL;" & InitialURL
6	Sheets("Prices").Select: Cells.Clear
7	For i = 0 To TotalPages – 1
8	y = i * RecordsPerPage
9	DynamicWqURL = WqURL & y
10	With ActiveSheet: LastRow = .Cells(.Rows.Count, "A").End(xlUp).Row: End With
11	With ActiveSheet.QueryTables.Add(Connection:=DynamicWqURL, _ Destination:=Cells(LastRow + 1, 1))
12	.WebTables = "15"
13	.Refresh BackgroundQuery:=False
14	End With
15	With ActiveSheet: LastRow = .Cells(.Rows.Count, "A").End(xlUp).Row: End With
16	Rows(LastRow).EntireRow.Delete
17	Next
18	Application.ScreenUpdating = True: ActiveWorkbook.Save
19	End Sub

Table 8. Task 4 VBA Code Description

Line	Code Description
1	Create the subroutine named **Get_Stock_Data**.
2	Stop updating the screen and select the **IO** worksheet.
3	Name and set new variable **RecordsPerPage** to the value in cell **H8**, and name and set a new variable **TotalRecords** to the value in cell **I8**.

4	Name and set new variable **TotalPages** to the value in cell **J7**, and name and set a new variable **StaticURL** to the url stored in cell **B10**.
5	Name and set a new variable **WqURL** using the text string "URL;" concatenated with the **StaticURL** from line 4.
6	Select the **Prices** worksheet and clear all previous data.
7	Set a **For-Next** loop to iterate through the TotalPages value; i = 0 to TotalPages-1.
8	Name and set a new dynamic variable **y** to the iteration number (i) multiplied by the RecordsPerPage value.
9	Name and set a new dynamic variable **DynamicWqURL** to the WqURL concatenated with the dynamic value of variable y.
10	Name and set a new variable **LastRow** and use a VBA method (*Cells(.Rows.Count, "A").End(xlUp).Row*) to count the number of used rows in the worksheet.
11-14	Execute the web query using the DynamicWqURL from line 10, downloading the data into the destination cell of the worksheet located at the first cell below the last used row (calculated in line 10).
	Recalculate the **LastRow** variable
15	Delete the row located at the location of the last used row in the worksheet.
16	Continue the **Next** iteration; i.
17	Update the screen and Save the workbook
18	End the subroutine.
19	

Table 9. Task 5 VBA Code

Line	Code
1	Sub Clean_Data()
2	Rows(1).EntireRow.Delete
3	With ActiveSheet: LastRow = .Cells(.Rows.Count, "A").End(xlUp).Row: End With
4	For i = LastRow To 2 Step -1
5	If WorksheetFunction.IsText(Cells(i, 2)) = True Then
6	Rows(i).EntireRow.Delete
7	End If
8	Next
9	ActiveWorkbook.Save
10	End Sub

Table 10. Task 5 VBA Code Description

Line	Code Description
1	Create subroutine named **Clean_Data**.
2	Delete the first row in the **Prices** worksheet.
3	Name and set a new variable **LastRow** and use a VBA method (*Cells(.Rows.Count, "A").End(xlUp).Row*) to count the number of used rows in the worksheet.
4	Set a **For-Next** loop to iterate through all the rows in the webpage; from i = LastRow to 2 (reverse order) in increments of 1.
5-6	For each iteration, use an **If** function to delete the rows containing text using a VBA method (WorksheetFunction.IsText).
7	End the If function statement.

8	Continue the **Next** iteration; i.
9	Save the workbook.
10	End the subroutine.

Table 11. Single Subroutine Combining 5 Subroutines

Line	Code
1	Sub Stock_Date_WQ()
2	Call Extract_Webpage_URLs
3	Call Extract_Page_Navigation_URLS
4	Call Extract_Parameters_Values
5	Call Get_Stock_Data
6	Call Clean_Data
7	Range("A1").Select
8	MsgBox "Finished"
9	End Sub

APPENDIX B

To display the **Developer** tab in the Excel menu ribbon, perform the following steps.

Step 1. Open Excel 2016.
Step 2. Click on the "File" tab as shown in Figure 13.
Step 3. In the window that opens, navigate the far left pane to last menu item and click on "Options."
Step 4. In the Excel Options dialogue box opens, navigate the far left pane and click on "Customize Ribbon."
Step 5. In the far right side of the window, inside the "Main Tabs" section, select the "Developer" box as shown in Figure 14.
Step 6. Click the OK button.
Step 7. Return to the main menu and click on the "Developer" tab and note the far left group named "Code," as shown in Figure 15.

Figure 13. Menu Bar Displaying File Tab

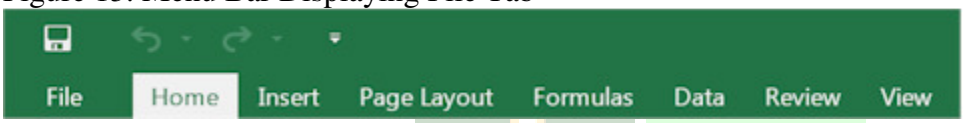


Figure 14. Excel Options Dialogue

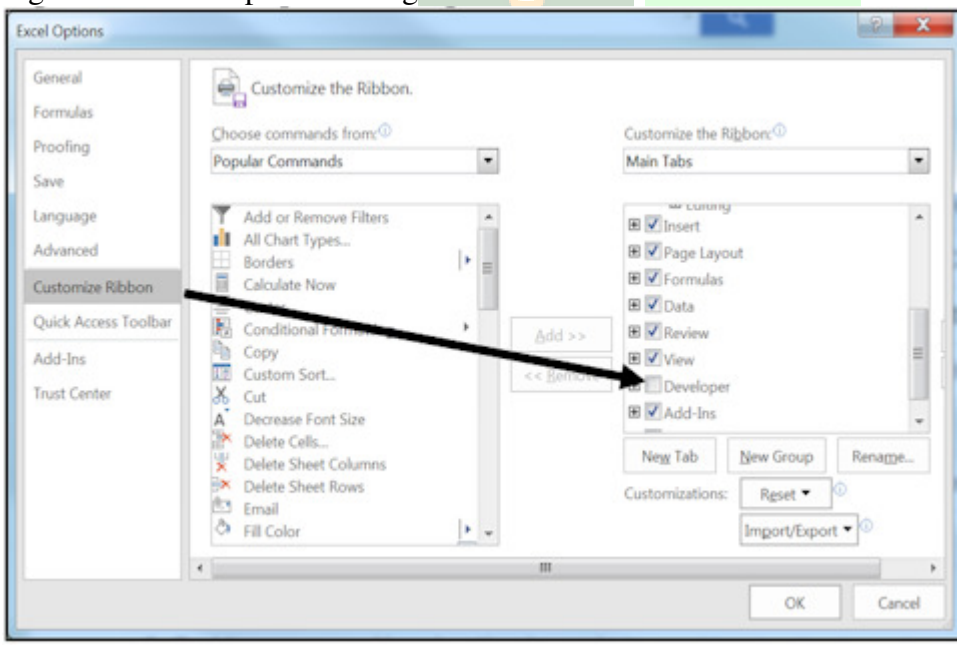
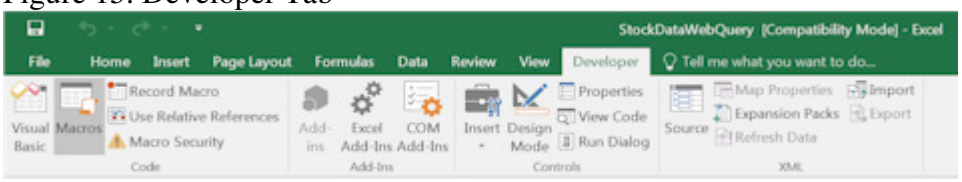


Figure 15. Developer Tab



APPENDIX C

To open the VBE, insert a new module and write a VBA subroutine, perform the following steps (with Excel open).

Step 1. Click on the **Developer** tab, as shown in Figure 6.

Step 2. Click on the icon labeled **Visual Basic** in the **Code** group (far left icon), also shown in Figure 6 (Appendix A). The Visual Basic Editor will open as shown in Figure 16 below. Note that previous versions of Excel will appear slightly different, showing 3 worksheets instead of 1. Also note the VBE can be opened using **Alt+F11**.

Step 3. To insert a new module, click the **Insert** menu option and select **Module**, as shown in Figure 17. Note the module's VBA programming pane opens to the right, and the new module is named **Module1**, as shown in Figure 18.

Figure 16. Visual Basic Editor

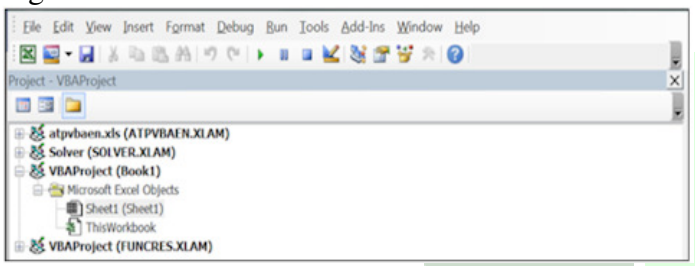


Figure 17. VBE Module

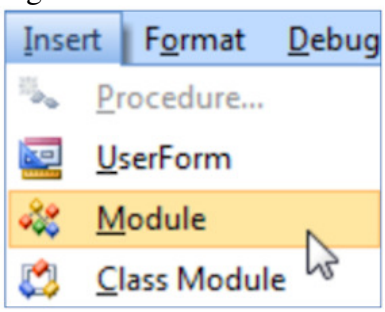
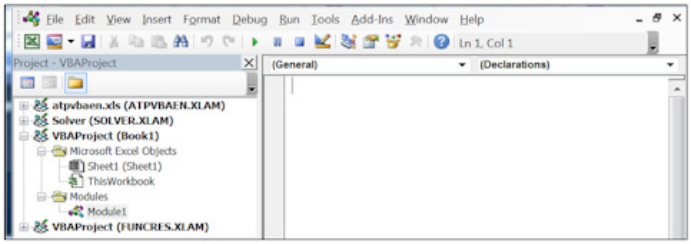


Figure 18. Visual Basic Editor Programming Pane



APPENDIX D

Table 12. Web Query Macro VBA Code

Line	Code
1	**With ActiveSheet.QueryTables.Add(Connection:= _**
2	**"URL;http://finance.yahoo.com/q/hp?s=WMT", Destination:=Range("A1"))**
3	.CommandType = 0
4	.Name = "hp?s=WMT"
5	.FieldNames = True
6	.RowNumbers = False
7	.FillAdjacentFormulas = False
8	.PreserveFormatting = True
9	.RefreshOnFileOpen = False
10	.BackgroundQuery = True
11	.RefreshStyle = xlInsertDeleteCells
12	.SavePassword = False
13	.SaveData = True
14	.AdjustColumnWidth = True
15	.RefreshPeriod = 0
16	.WebSelectionType = xlSpecifiedTables
17	.WebFormatting = xlWebFormattingNone
18	**.WebTables = "15"**
19	.WebPreFormattedTextToColumns = True
20	.WebConsecutiveDelimitersAsOne = True
21	.WebSingleBlockTextImport = False
22	.WebDisableDateRecognition = False
23	.WebDisableRedirections = False
24	**.Refresh BackgroundQuery:=False**
25	**End With**

APPENDIX E

Add the Command Button to the Worksheet

Step 1. Select the **IO** worksheet.

Step 2. Click the **Developer** tab

Step 3. Under the **Controls** group, select the **Insert** dropdown list, and select the button icon (first icon on the left under the **Form Controls** section.

Step 4. Click anywhere on the **IO** worksheet and the Assign Macro screen opens. Select the Macro name **Stock_Data_WQ** and click OK as shown in Figure 19.

The button text can be edited by right-clicking the button, highlighting the existing text ("**Button**") and typing over the text, like "Run." Likewise, the button can be relocated on the worksheet by right-clicking it, dragging it to the desired location, releasing it and selecting the option Move Here. Figure 20 reflects the button located over cell C1 in the worksheet.

Figure 19. Assign Macro Dialogue

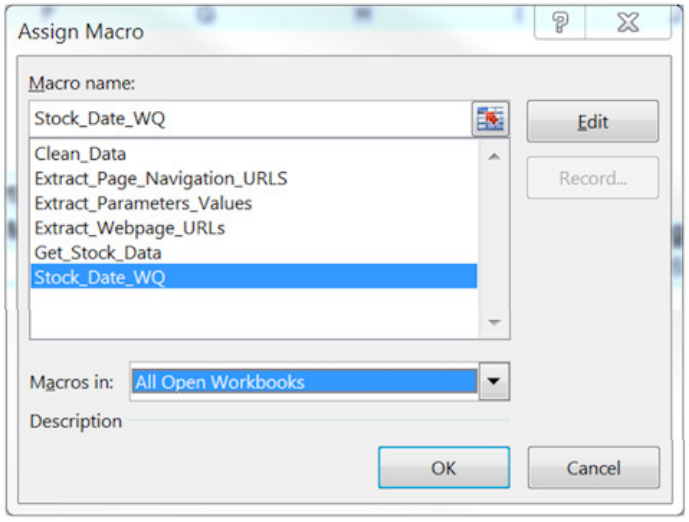
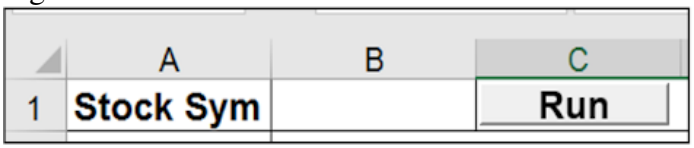


Figure 20. IO Worksheet Run Button



REFERENCES

Brody, H. (2012). I Don't Need No Stinking API: Web Scraping For Fun and Profit. Retrieved November 5, 2015, from https://blog.hartleybrody.com/web-scraping/

Bordy, H. (2013). *Ultimate Guide to Web Scraping*. Lean Publishing.

Data Scraping. (2015). In Wikipedia, The Free Encyclopedia. Retrieved October 19, 2015, from https://en.wikipedia.org/w/index.php?title=Data_scraping&oldid=679724055

Document Object Model. (2015). In Wikipedia, The Free Encyclopedia. Retrieved October 26, 2015, from https://en.wikipedia.org/w/index.php?title=Document_Object_Model&oldid=685393325

Get external data from a Web page. (n.d.). Retrieved October 19, 2015, from https://support.office.com/en-us/article/Get-external-data-from-a-Web-page-708f2249-9569-4ff9-a8a4-7ee5f1b1cfba

How to Create Web Query (.iqy) Files. (2003). Retrieved November 5, 2015, from https://support.microsoft.com/en-us/kb/157482

HTML DOM getElementsByName() Method. (n.d.). Retrieved October 29, 2015, from http://www.w3schools.com/jsref/met_doc_getelementsbyname.asp

HTML DOM getElementsByTagName() Method. (n.d.). Retrieved October 29, 2015, from http://www.w3schools.com/jsref/met_element_getelementsbytagname.asp

HTML DOM innerHTML Property. (n.d.). Retrieved October 20, 2015, from http://www.w3schools.com/jsref/prop_HTML_innerHTML.asp

HTTP methods: GET vs. POST. (n.d.). Retrieved October 20, 2015, from http://www.w3schools.com/tags/ref_httpmethods.asp

Import external data from a complex web site into Excel. (n.d.). Retrieved November 5, 2015, from http://datatoolbar.com/Import-external-data-from-a-complex-web-page-into-Excel.html

International Journal of People-Oriented Programming (IJPOP). (n.d.). Retrieved November 11, 2015, from http://www.igi-global.com/journal/international-journal-people-oriented-programming/41021.

Korpela, J. (2003). methods GET and POST in HTML forms - what's the difference? Retrieved October 20, 2015, from http://www.cs.tut.fi/~jkorpela/forms/methods.HTML

Korol, J. (2014). *Microsoft Excel 2013 programming by example with VBA, XML, and ASP*. Dulles, Va.: Mercury Learning and Information.

Levitt, J. (n.d.). From EDI To XML And UDDI: A Brief History Of Web Services - InformationWeek. Retrieved October 15, 2015, from http://www.informationweek.com/from-edi-to-xml-and-uddi-a-brief-history-of-web-services/d/d-id/1012008?

Pieterse, J. (2006). Using Parameters With Web Queries. Retrieved October 19, 2015, from http://www.jkp-ads.com/articles/webquery.asp?AllComments=True

Pull data into Microsoft Excel with Web queries - TechRepublic. (2006). Retrieved October 19, 2015, from http://www.techrepublic.com/article/pull-data-into-microsoft-excel-with-web-queries/

Rice, F. (2004). Different Ways of Using Web Queries in Microsoft Office Excel 2003. Retrieved October 20, 2015, from https://msdn.microsoft.com/en-us/library/office/aa203721(v=office.11).aspx

Roos, D. (2007). "How to Leverage an API for Conferencing." HowStuffWorks.com. Retrieved October 19, 2015, from http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing.htm.

The HTML DOM Document Object. (n.d.). Retrieved October 26, 2015, from http://www.w3schools.com/jsref/dom_obj_document.asp

Visual Basic for Applications. (n.d). In *Wikipedia, The Free Encyclopedia*. Retrieved October 22, 2015, from https://en.wikipedia.org/w/index.php?title=Visual_Basic_for_Applications&oldid=682899862

Walkenbach, J. (2013). *Excel 2013 power programming with VBA*. Hoboken, N.J.: Wiley.

Web Data Extraction Software Made Simple. (n.d.). Retrieved November 6, 2015, from http://datatoolbar.com/

Web Scraper. (n.d.). Retrieved October 19, 2015, from http://webscraper.io/

Web scraping. (2015). In *Wikipedia, The Free Encyclopedia*. Retrieved November 5, 2015, from https://en.wikipedia.org/w/index.php?title=Web_scraping&oldid=689058873

Web service. (n.d.). In Wikipedia, The Free Encyclopedia. Retrieved October 15, 2015, from https://en.wikipedia.org/w/index.php?title=Web_service&oldid=685767160

Wittwer, J. (n.d.). Excel Web Query Secrets Revealed. Retrieved October 19, 2015, from http://www.vertex42.com/News/excel-web-query.HTML

XML Web Services. (n.d.). Retrieved October 15, 2015, from http://www.w3schools.com/xml/xml_services.asp