VBA Final Project
Write-Up

Scott Henderson

04/13/14

Executive Summary

My wife is the HR manager for a small, but growing company in the valley. They currently have around 60 employees, but anticipate this to grow substantially over the next few years. I found they were currently using a very long and tedious process in order to record and process their payroll. My wife would email each employee asking for their hours. They would then reply (sometimes with text in an email, sometimes by filling out a timecard template in Microsoft Excel and attaching it their return email. She would then take that data, manually convert the hours into decimal format and calculate overtime hours for each employee. This data would then have to be entered manually into a master spreadsheet.

Before the spreadsheet could be sent to the contracted payroll company for payment processing, all of the reported hours needed to be verified by each of the employee's respective supervisor. So again these reports were manually created and sent to each supervisor for approval.

This process is extremely repetitive. It is prone to calculation errors, along with many other minor errors that could cause the payroll to be completed incorrectly. When something has to be done in exactly the same way, multiple times over, it is a sure sign that it should be automated.

The owners of the company did not want to spend a lot of money creating a web portal in-house, nor did they desire to use an external online service. My solution is completely free. Using a custom built google form, and Microsoft VBA I was able to automate the process.

What would usually have taken the HR manager hours can now be done by clicking a button. I have generated a google form where the employees can go to report their hours for the week. These responses are stored on a google spreadsheet which can only be accessed by those authorized. My solution will automatically download this raw data file from google drive, import the raw data, calculate the regular hours, overtime hours and total hours for the week, and generate a summary spreadsheet in the desired format.

At this point a report can be automatically generated for each supervisor, listing all the information for each of the employees they manage. These reports are now able to be sent to the supervisors for validation.

This solution is a much more effective way to gather the information needed fir the company to process their payroll, and effectively saves more than 4 hours of work each week for the HR manager.

Implementation

I adopted and applied many of the skills I acquired throughout the semester as I went about implementing my solution. In this section you will find step by step documentation as to how my solution is composed and exactly how it works.

Overview

Step 1: HR manager sends a link for Employees to enter detailed time card information on my custom google form

Step 2: Once employees have entered their information, theHR manager (using the excel document containing my solution) navigates to the "Payroll" tab on the ribbon and clicks the "Populate Spreadsheet" button.

Step 3: Timecard data is automatically downloaded in excel, calculations made, and summary sheet generated.

Step 4: HR manager clicks the "Generate Supervisor Reports" button on the payroll tab.

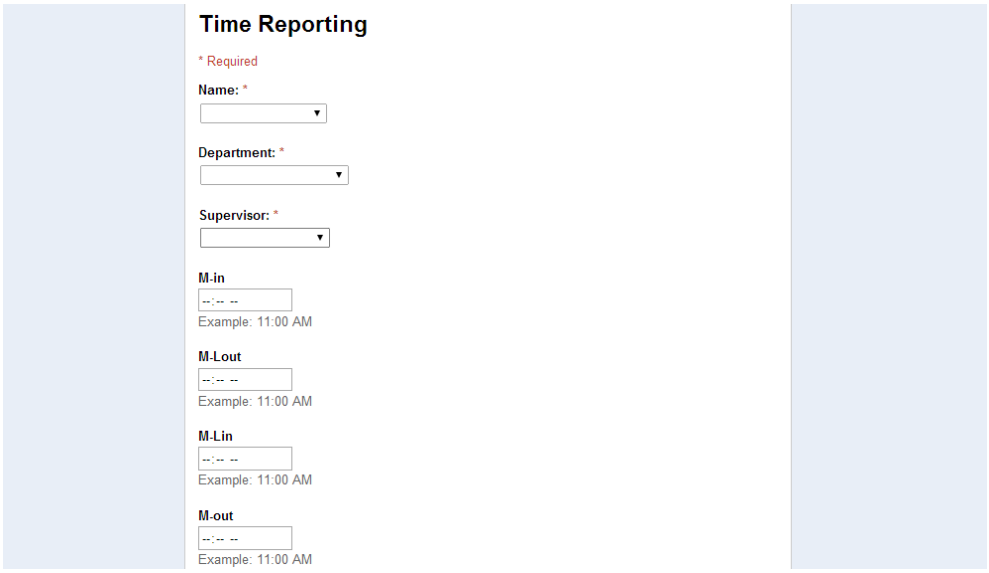
Step 5: A reports is generated for each supervisor in the organization, at which point the HR manager is free to email these reports for verification as per company policy.

Custom Timecard Using Google forms

Using google forms I created a standard form online and made it public to those given the link. The simplicity of google forms did not allow for the customization I desired, and therefore I opted to manipulate the form with custom html and css to give it the look and feel I wanted. The standard format of google forms does not allow the use of html tables. This meant that every single time entry for clocking in and out (four for each day of the week) would have to appear on a separate line; this was not very practical, therefore I manipulated the source code to allow the form to still be submitted via the google form engine, but in a way that allowed a more appealing layout.

Below I have provided two screenshots; one of each of the forms as they appeared in the browser.

Standard Google Form:



Time Reporting

* Required

Name: *

Department: *

Supervisor: *

M-in

--:-- --

Example: 11:00 AM

M-Lout

--:-- --

Example: 11:00 AM

M-Lin

--:-- --

Example: 11:00 AM

M-out

--:-- --

Example: 11:00 AM

My Custom Google Form:

Personal Information

Name:	Select Department:	Select Supervisor:

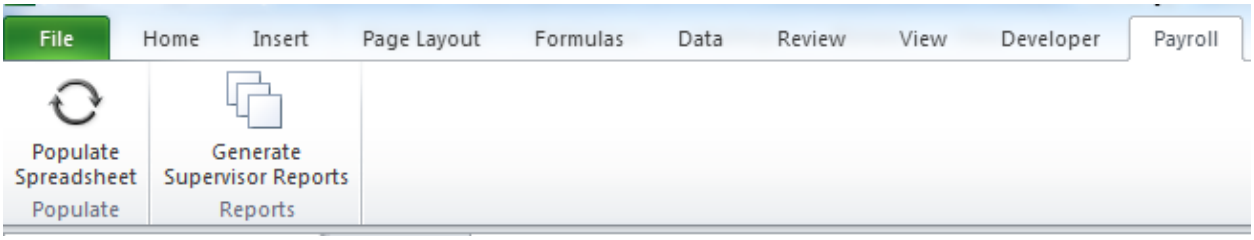
Time Reporting

Date	IN	Lunch OUT	Lunch IN	OUT
Monday	--:-- --	--:-- --	--:-- --	--:-- --
Tuesday	--:-- --	--:-- --	--:-- --	--:-- --
Wednesday	--:-- --	--:-- --	--:-- --	--:-- --
Thursday	--:-- --	--:-- --	--:-- --	--:-- --
Friday	--:-- --	--:-- --	--:-- --	--:-- --
Saturday	--:-- --	--:-- --	--:-- --	--:-- --
Sunday	--:-- --	--:-- --	--:-- --	--:-- --

This process was new to me and took some effort to make sure all of the elements of the form were correctly recorded on the google spreadsheet on the back-end when the form was submitted. The form itself is made public to anyone the HR manager sees fit to send the URL to, whereas the spreadsheet recording the employee submissions is only accessible to those with authorization. This allows for the protection of the private information submitted by each employee.

"Populate Spreadsheet" Button

This is where the bulk of the work is done.

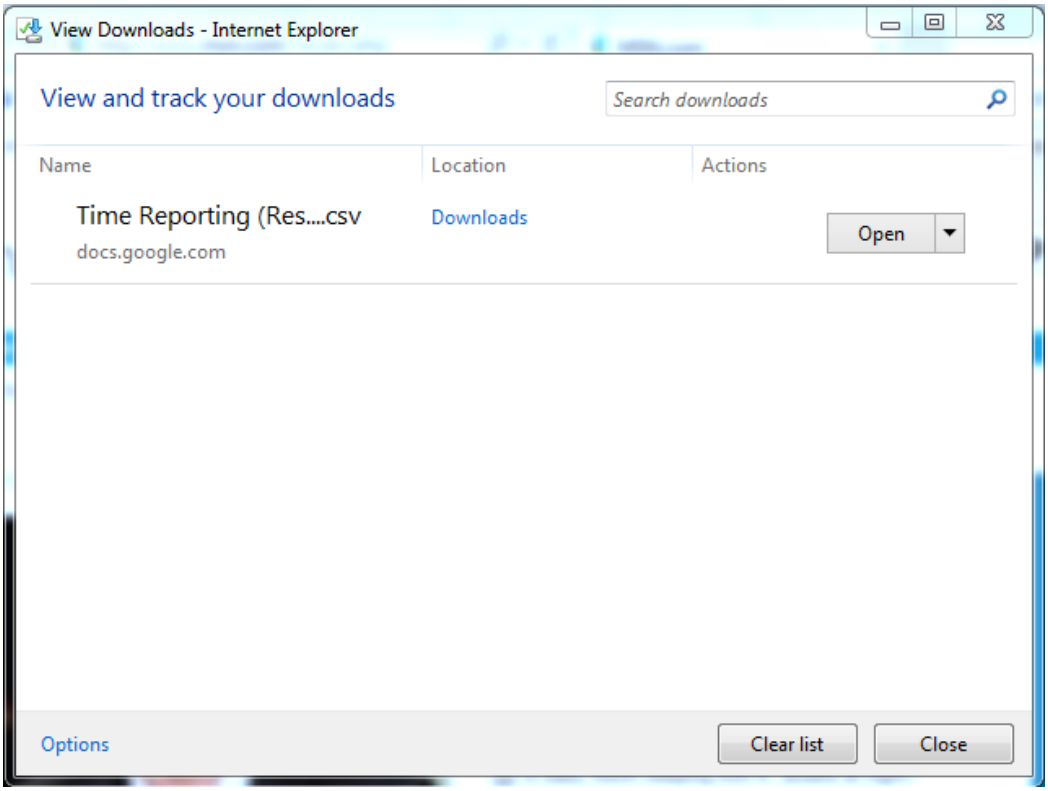


The following code snippet is what is programmed behind the 'Populate Spreadsheet' button:

```
Sub ONE_GenerateSpreadsheet()
   DownloadData
   moveFile
   copyResponseData
   GenerateSummary
End Sub
```

DownloadData

This custom Sub first initiates an internet explorer application with the intention of navigating to the location of the google spreadsheet and downloading the file. It first checks to see whether the browser is already logged in to google drive, and if not it logs in. At this point a download request is immediately initiated and the file is downloaded to the 'downloads' folder. Internet Explorer made this very difficult. The browser is programmed to prevent the automatic download of any file without the user's specific approval. The only way I could bypass this was to use the SendKeys function to confirm the action as proxy for the user that I confirm the download.

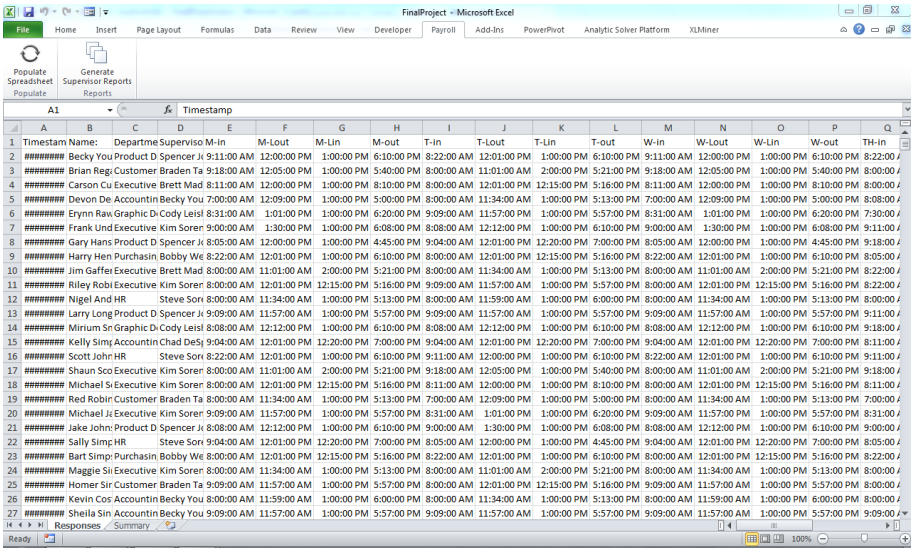


moveFile

Since the file is downloaded directly into the 'downloads' folder I wanted to make sure that this file was immediately moved into the same folder where the solution workbook is located in order to keep track of it. This Sub therefore uses the 'dir' function to loop through the downloads directory, return the file name, which I then use to rename the file path to the location that I desire, while at the same time renaming the file and adding a date and time stamp to it so to keep track of what version of the file it is.

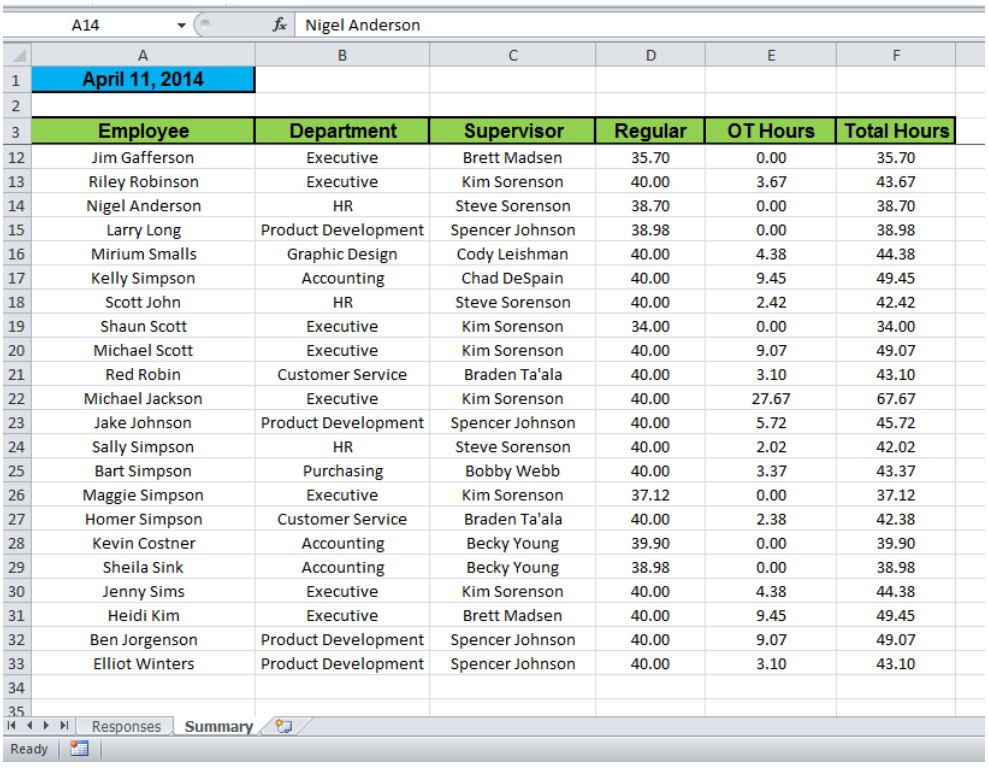
copyResponseData

This Sub then opens the downloaded and relocated file, and imports all of the data into the current worksheet on the tab named "Responses."



GenerateSummary

This Sub then takes all the data, performs all the necessary calculations including the transformation into decimal format, totals the regular hours, overtime hours, and generates a summary tab formatted in the way requested by the payroll company.



A14 | Nigel Anderson

	A	B	C	D	E	F
1	April 11, 2014					
2						
3	Employee	Department	Supervisor	Regular	OT Hours	Total Hours
12	Jim Gafferson	Executive	Brett Madsen	35.70	0.00	35.70
13	Riley Robinson	Executive	Kim Sorenson	40.00	3.67	43.67
14	Nigel Anderson	HR	Steve Sorenson	38.70	0.00	38.70
15	Larry Long	Product Development	Spencer Johnson	38.98	0.00	38.98
16	Mirium Smalls	Graphic Design	Cody Leishman	40.00	4.38	44.38
17	Kelly Simpson	Accounting	Chad DeSpain	40.00	9.45	49.45
18	Scott John	HR	Steve Sorenson	40.00	2.42	42.42
19	Shaun Scott	Executive	Kim Sorenson	34.00	0.00	34.00
20	Michael Scott	Executive	Kim Sorenson	40.00	9.07	49.07
21	Red Robin	Customer Service	Braden Ta'ala	40.00	3.10	43.10
22	Michael Jackson	Executive	Kim Sorenson	40.00	27.67	67.67
23	Jake Johnson	Product Development	Spencer Johnson	40.00	5.72	45.72
24	Sally Simpson	HR	Steve Sorenson	40.00	2.02	42.02
25	Bart Simpson	Purchasing	Bobby Webb	40.00	3.37	43.37
26	Maggie Simpson	Executive	Kim Sorenson	37.12	0.00	37.12
27	Homer Simpson	Customer Service	Braden Ta'ala	40.00	2.38	42.38
28	Kevin Costner	Accounting	Becky Young	39.90	0.00	39.90
29	Sheila Sink	Accounting	Becky Young	38.98	0.00	38.98
30	Jenny Sims	Executive	Kim Sorenson	40.00	4.38	44.38
31	Heidi Kim	Executive	Brett Madsen	40.00	9.45	49.45
32	Ben Jorgenson	Product Development	Spencer Johnson	40.00	9.07	49.07
33	Elliot Winters	Product Development	Spencer Johnson	40.00	3.10	43.10
34						
35						

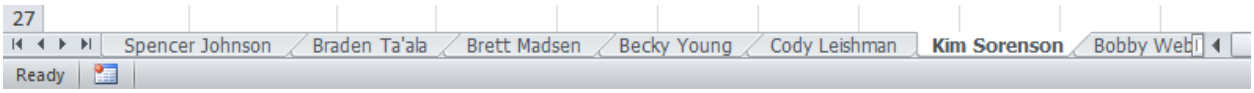
Responses | Summary

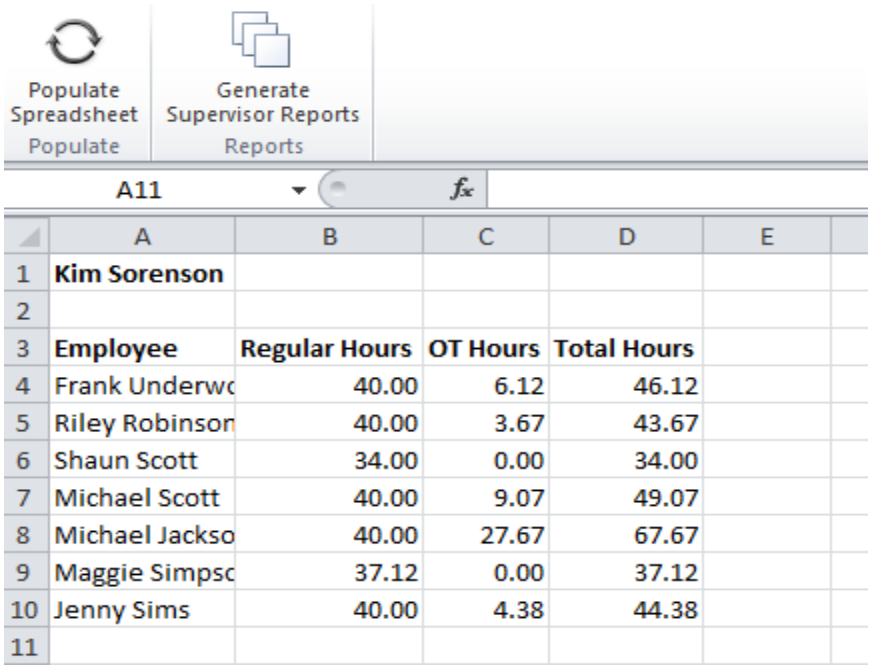
Ready

Once all of the data has been imported correctly and the summary tab created, the HR manager is able to review the figures by eye to see if everything looks good. If any of the employees have not submitted their timesheet at this point, the HR manger can contact them and ask them to do so. Upon confirmation that the said employee has now completed their timesheet the 'Populate Spreadsheet' button can simply be clicked again, and the information will be re-downloaded, imported and formatted again so to represent the most current statistics.

Generate Custom Supervisor Reports

Once all of the employees have submitted their timesheets, the calculations have been performed, and the summary sheet generated, it is time to create the supervisor reports. As per company policy, every supervisor must verify that the submitted by those they manage are accurate. In an attempt to tackle this I decided that I would generate a new workbook names 'SupervisorReports', save it to the same directory as the current file, and populate multiple tabs individually labelled by the name of each supervisor. These reports could then be emailed out to each of the supervisors for verification. This is a lot nicer than having to generate these reports manually.





	A	B	C	D	E
1	**Kim Sorenson**				
2					
3	**Employee**	**Regular Hours**	**OT Hours**	**Total Hours**	
4	Frank Underwc	40.00	6.12	46.12	
5	Riley Robinsor	40.00	3.67	43.67	
6	Shaun Scott	34.00	0.00	34.00	
7	Michael Scott	40.00	9.07	49.07	
8	Michael Jackso	40.00	27.67	67.67	
9	Maggie Simpsc	37.12	0.00	37.12	
10	Jenny Sims	40.00	4.38	44.38	
11					

In determining the best way to generate these reports I decided to research something that had not really been discussed in the course material. I thought it would be nice to create an 'Employee' class within the code. After some research I determined this to be the best way to go about it, with the added bonus of learning something new. It was nice to be able to generate an array of employee objects that I could assign variables to. Each employee object held multiple variables such as name, supervisor name, regular hours, total hours, and overtime hours.

I then generated a loop that looked through all the supervisor names from the supervisor column on the summary tab, and created a fresh tab in a new workbook for every unique supervisor name I found. This then allowed me to look through all the employee objects and place all the employees' information on the tab that matched the 'employee.supervisorName'.

This feature is very useful as it is, however; had I had been allocated more time, and the project required more scope, I would have created another feature that allowed these reports to be emailed out directly through the excel workbook. I could have generated an email based off of the supervisors name and sent it directly from the report. Or without even viewing the reports I could have created a feature that just sent the email out to every specific supervisor.

Learning/Conceptual Difficulty

This final project was extremely beneficial to my learning in this class. It provided me with an opportunity to not only gather all the concepts taught throughout the semester and put them to practical use, but it also provided me with a chance to overcome some more difficult hurdles along the way.

Downloading from IE

The most challenging part of the project was finding a way to make internet explorer download the employees responses file automatically. I wanted this process to be completely automated, and would have seen it as a huge failure had I not been able to figure it out. It is because of this that I spent a great deal of time trying to find out how to accomplish this.

After a lot of research with no success, I spent around an hour or so with Dr. Allen trying to find a way. Dr. Allen researched some more complicated options which ultimately lead us right back where we started. One thing Dr. Allen helped me with, which was vital, was finding the exact URL I needed to navigate to that would immediately fire off the download trigger for the file. We did this using google chromes ability to monitor the network traffic as we manually pressed the download button on the file.

Once I had this, I eventually figured out a way to initiate Internet Explorers download manager directly without having to actually navigate to google drive. Once the download was initiated, I used Sendkeys function to confirm the option of downloading the file. From this point on it was a matter of locating the file and importing the data, which seemed trivial in comparison.

Creating a Class Module

The actual syntax was not the tricky part here, but it was the conceptual thinking behind what needed to be done when creating a custom class, and then how to interact with it in the body of the code. It proved very beneficial to me in the end, and was ultimately worth the struggle. To be essentially able to create your own objects and set variables to them is hugely important. I can particularly see the importance of this when working on much bigger projects. It allow for a much more structured approach to storing the particular information you need.

Using the Wildcard (*)

When the file downloaded from IE, I was frustrated to find that it did not download with the exact same filename each time. It had some sort of code attached to it which I'm assuming was to represent a particular instance of the download. Also I couldn't just simply copy the file directory and look there because I realized that the 'downloads' folder would

not have exactly the same path on everyone's computer running this code. Therefore I had to first of all find the name of the user of the current running machine, and append into the file path, and then use the "*" character to search for something like the filename, since the first part of the filename was always the same. It was a little tricky at first but I eventually figured it out.

Key Learning

Looking back on this project I think the most difficult thing about it is the fact that there are so many variables that need to be considered when writing a program that is intended to work on any machine that uses Microsoft Excel. It is not as easy as writing the program and making it work. I learnt the importance of testing and troubleshooting, it is almost like you have to think about it from every angle, and bulletproof your code so that it is error proof. It helps you be more inquisitive and thorough; in fact it forces you to.

Another key learning is that I realized just how many different things there are that you can do with VBA. There is so much syntax that it is almost impossible to learn it all. So the internet is vital. It is so important to know how to search the web, and test what you find until you find the right answer. Also at the same time you can't just use code you don't understand, because that makes debugging almost impossible. If you don't know what your code does you can't possibly figure out why it isn't working.

Assistance

It is here that I will disclose the level of help I have received from other persons on this project. The only help I have received from another person other than the use of the internet is limited to Dr. Allen. I met with him for around an hour or so as we attempted to resolve the issue I was having regarding the downloading of data from the internet. I have previously discussed this situation earlier in my report, however; I will do so again here.

After struggling to download the particular file I needed from google drive I met with Dr. Allen. He helped me a great deal by finding the direct URL that triggered the download of the file I needed, in the correct format. After getting this far I was then able to personally find a way to fully automate the download and data import process.