ANTHROPOLOGICAL DATASETS: AN EXAMINATION OF CSV FILE DATA USING SHELL SCRIPTING AND A COMPARISON TO R

CS/IT 3530 – UNIX Operating Systems

TOPIC

I've chosen to use some anthropological datasets to more closely examine how CSV files can be dealt with using shell scripting, and how the results I get from that might compare to the results obtained from a language primarily used to deal with statistical data, R. I won't be going into any depth with the statistical analysis side since my project is centered around handling the data itself, but I wanted to compare against something people in the industry would actually use.

I chose this project for two reasons: I wanted to do a comparison of reading/using datasets in shell scripting with languages I was already somewhat familiar with; and I wanted to get a better handle on R. While we did some shell scripting in class, most of the content was focused on executing commands in a sequence to accomplish something, not parsing through data, so I felt like this would be a good fit. I chose R because I had to use it for one of my previous anthropology classes, but I never really understood it. I thought this might help.

The website the datasets are from was D-Place, described on their webpage as a "Database of Places, Language, Culture, and Environment", and are from their Standard Cross-Cultural Sample, a 1969 study of 186 societies (1 below). With this comes inherent similarities in the structure of the CSV files; each dataset on the site has the same categories/fields in the same order, so I can easily combine them. The only line that won't have the same structure as the rest is the first one, which contains labels for the fields. My code reflects this, but the basic principles (of combining and interpreting CSV data) would remain the same even if I were to use CSV files from another site. Figure 1 is a snippet of what the datasets I downloaded look like.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	society_id	society_name	society_xd_id	language_glottocode	language_name	language_family	variable_id	code	code_label	focal_year	sub_case	comment	
2	SCCS113	Atayal	xd732	atay1247	Atayal	Austronesian	SCCS655	1	Absence of	1930	Tribe (excluding Sedeq)		
3	SCCS115	Manchu	xd646	manc1252	Manchu	Tungusic	SCCS655	1	Absence of	1915	Aigun District		

Figure 1: Sample CSV

RESEARCH

(Note: since the topic of this project is so hands-on, the majority of the content will be in APPLICATION instead of RESEARCH; that being said, APPLICATION also holds quite a bit of research.)

Shell scripting is usually used for "system level operations" such as automating processes or administrative roles (e.g. setting up employee profiles to automatically have group permissions based on their role in the company), according to Uses of shell scripting - https://www.educba.com/uses-of-shell-scripting/. While it's not easy to do everything – my attempt at graphing in APPLICATION can attest to that – it gives the user a lot of flexibility, allowing everything from running a sequence of commands to, as I did here, combining multiple datasets and finding the frequency of certain values. This flexibility is, in part, due to how it allows the use of terminal commands in one's code. One can easily take any output from a command and stick it in a file, same as the method for doing so in a terminal, but one can also interface with users and error check, same as a programming language. Since Unix systems are so widely used, shell scripting is also fairly universal; one can download a script from anywhere and run it on their machine, and if it worked originally it should also work on the new device (below).

Large amounts of data frequently take the form of CSV files. Since they contain nothing but data separated by delimiters (usually commas), CSV files can be used in innumerable programs. An example (from CSV file background - https://www.businessinsider.com/what-is-csv-file) mentions that one "can save contact information" as a CSV file in Excel and import it into their Outlook. In searching for topics for this project, I found several CSV files holding COVID-19 statistics in Missouri. In the past, I've used CSV files mainly in my anthropology classes; they are an easy way to export a large amount of data, and they are simple for programs and people to read.

APPLICATION

For the sake of brevity, I'm leaving some comments in my script and am not including screenshots of all of its content; covered below are the topics I found important in creating the file and the sections I thought needed an explanation.

I approached this project step-by-step, beginning with shell scripting:

1. Open a CSV file.
2. See if the user would like to combine its contents with a second file.
    a. If so, combine the contents of the two files.
3. Parse through the data.
4. List any important results (e.g. an average or a common trend).
5. Try to graph the data.
6. Repeat in R

Step 1 was mostly completed in a previous lab (lab 10, with the context menu and a lot of options), so I referenced that to create anything handling file opening, namely for printf statements and reading values from the user but also for concatenation and checking to see if files exist.

Step 2, actually combining the contents of two files, trickier. I set up my script to create a new file to hold the combined data, but I didn't want it to overwrite any pre-existing files. Instead, I chose to add a number to the filename. While this wasn't difficult, it did take me a minute to figure out. I ended up creating a loop and incrementing a variable (Formatting the incrementing variable in the file creation loop - https://linuxize.com/post/bash-increment-decrement-variable/) to append to the end of the new filename until there was no file with that name (

```
29                  until  [[ $new == "y" ]]
30                  do
31                          if [[ -e $filename ]]
32                          then
33                                  filename="$combinedName$i.csv"
34                                  i=$(($i + 1))
35                          else
36                                  new="y"
37                          fi
38                  done
```

Figure 2). I then used Sorting CSV files - https://www.geeksforgeeks.org/sort-command-linuxunix-examples/ to sort the two files the user wanted to combine (by their "society_id" category, which is

unique for every society within the sample I was using). I chose to do this using version-sort for my own readability, but ultimately as long as both files were sorted in the same manner, it didn't particularly matter; version-sort simply allowed me to see each "society_id" in numerical order (Figure 3). Actually joining the two tables was rather difficult and required outside help (Merging two CSV files (including combining their lines if they have the same content) - https://unix.stackexchange.com/questions/113898/how-to-merge-two-files-based-on-the-matching-of-two-columns); as seen in Figure 4, it is quite a long line, involving a lot of flags. The difficult part was getting the files to combine fully; this involved adding "-a1" and "-a2" to allow for non-matching lines to also be added to the combined file, but also "--nocheck-order" because a sorting error kept popping up. Applied, the whole thing joins two existing files together and outputs only the lines listed after the "-o" flag, since some of the content of the files was duplicated.

```
29                until  [[ $new == "y" ]]
30                do
31                        if [[ -e $filename ]]
32                        then
33                                filename="$combinedName$i.csv"
34                                i=$(($i + 1))
35                        else
36                                new="y"
37                        fi
38                done
```

Figure 2: Creating the Combined File

```
47              if [[ -e $file1 ]]
48              then
49                      file1S="$file1No-Sorted.csv"
50                      sort -t ',' -k1 --version-sort -o $file1S $file1
51                      echo Sorted $file1
```

Figure 3: Sorting File1

```
                    $(join -j 1 -t ',' --nocheck-order -a1 -a2 -o 1.1,1.2,1.
3,1.4,1.5,1.6,1.7,1.8,1.9,2.8,2.9,1.10,1.11 $file1S $file2S > $filename)
```

Figure 4: Joining the Files

Since the most important sections of the datasets I chose are section 8 in the case of a single file and sections 8 and 10 for the merged file (i.e. those labelled with "code" back in Figure 1), that's the data I chose to parse through in Step 3. This combined with step 4 in my code, since I was using commands like "tail" and "cut" to parse the data – it seemed easier to just print the results then and there. This line is shown in Figure 5, with the first three sections (the segments using "tail", "cut", and "uniq") parsing the data and the last section (with "awk") handling printing the data so the user can interpret it.

```
101             code=$(tail -n +2 $file1 | cut -d',' -f 8 | uniq -c | awk '{printf("\t%s
    %d\n", $2, $1); }')
```

Figure 5: Parsing and Interpreting Data

```
Number of occurrences of each code in File 1:
         1 16
         2 45
         3 45
         4 24

Number of occurrences of each code in File2:
        1 81
        2 24
        3 17
        4 9
```

Figure 6: Results

Unfortunately, I was unable to complete step 5 (hence the "try" in the list of steps). I can't quite tell if I just can't figure out how to use gnuplot correctly or if there's something else at play, but I attempted to create a bar graph of the data in the "code" variables. Nothing worked, so it's neither in the script nor pictured here.

With that done, I moved to graphing the contents of the CSV files in R. I used an IDE called RStudio to do so. Here, I've provided all the code I used, since replication seems off the table. Obviously, this is much simpler, since this language (and IDE) are built to read and manipulate data. Lines 1 and 3 in Figure 7 are all it takes to open the CSV files, and line 5 merges the two by the category "society_id" (the same column I merged by).

```
> theoriesOfSorcery <- read.csv("C:/Users/Kylee/Downloads/theoriesOfSorcery.csv")
>   View(theoriesOfSorcery)
> theoriesOfWitchcraft <- read.csv("C:/Users/Kylee/Downloads/theoriesOfWitchcraft.csv")
>   View(theoriesOfWitchcraft)
> merged <- merge(theoriesOfSorcery, theoriesOfWitchcraft, by="society_id")
```

Figure 7: Opening CSVs and Merging Them

Using R, I graphed the frequency of each code in the datasets. In Figure 8 below is the code required to do so followed by the graphs of each in Figure 9. I used Bar graphs (on the official R site) - https://www.statmethods.net/graphs/bar.html as a guide on how to do so. The data graphed there is the same data I got from my shell script.

```
> counts <- table(merged$code.x)
> countss <- table(merged$code.y)
> barplot(counts, main="Sorcery", xlab="Code (increasing importance)")
> barplot(countss, main="Witchcraft", xlab="Code (increasing importance)")
```

Figure 8: Coding the Graphs

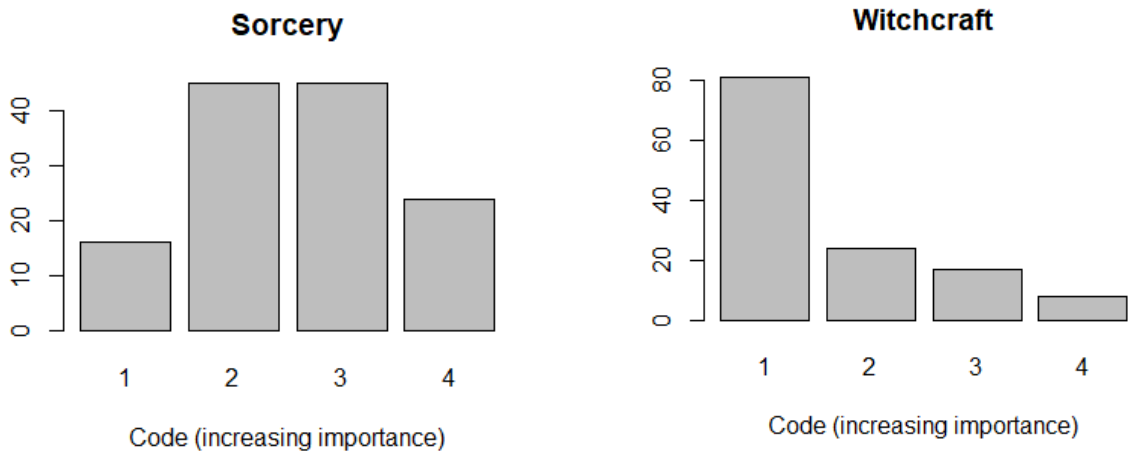

Figure 9: Bar Graphs

I also combined the two plots in a graph. I'm not entirely sure I can interpret this graph correctly, but it was still simple to combine the two (again using Bar graphs (on the official R site) - https://www.statmethods.net/graphs/bar.html as a guide, but also using Labeling the legend - https://r-graphics.org/recipe-legend-label-text to determine how to create the legend/key).

```
> countsss <- table(merged$code.x, merged$code.y)
> barplot(countsss, main="Sorcery and Witchcraft: Level of Importance", xlab="Code (Increasing
 Importance)", col=c("darkblue", "red"), legend=c("Sorcery", "witchcraft"))
> |
```

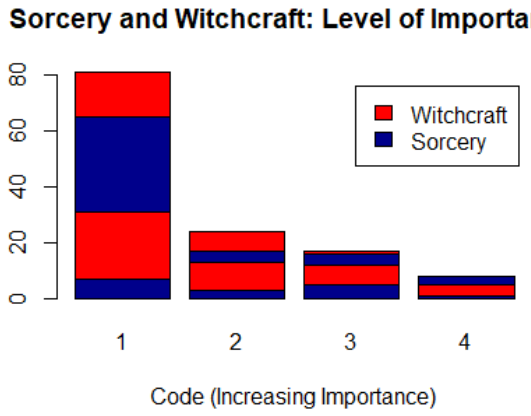

Figure 10: Combined Graph

In the end, I feel as though I've learned quite a bit about Bash, not so much about R. I expected to have more of a comparison with graphs, but in the end, I've accomplished what I set out to do (read through files, combine CSV files, and interpret some of the data within CSV files). The data I obtained, though not in the same format as the R data, did end up being the same; it was far more complex for me to obtain said data with shell scripting than with R, but the simplicity that R has doesn't actually show me anything happening behind the scenes.

SOURCES

1. Datasets from D-Place:
   a. Complete list of the Standard Cross-Cultural Sample - https://d-place.org/contributions/SCCS
   b. Theories of Witchcraft [SCCS656] - https://d-place.org/parameters/SCCS656#1/14/150
   c. Theories of Sorcery [SCCS655] - https://d-place.org/parameters/SCCS655#1/14/150

2. Sites that helped me with Bash content:
   d. Sorting CSV files - https://www.geeksforgeeks.org/sort-command-linuxunix-examples/
   e. Merging two CSV files (including combining their lines if they have the same content) - https://unix.stackexchange.com/questions/113898/how-to-merge-two-files-based-on-the-matching-of-two-columns
   f. Formatting the incrementing variable in the file creation loop - https://linuxize.com/post/bash-increment-decrement-variable/
   g. Tail formatting - https://linuxhint.com/bash_head_tail_command/

3. Sites that helped me with R content:
   h. Bar graphs (on the official R site) - https://www.statmethods.net/graphs/bar.html
   i. Labeling the legend - https://r-graphics.org/recipe-legend-label-text

4. Research resources:
   j. Uses of shell scripting - https://www.educba.com/uses-of-shell-scripting/
   k. CSV file background - https://www.businessinsider.com/what-is-csv-file
   l. Bash advantages - https://www.fosslinux.com/42541/bash-script-examples.htm