Journal and Assignments

Week 44

29.10: Having no experience with this sort of thing I've never been so unsure what to expect from a course, but the coding introduction on Thursday did clear things up substantially. Regex is certainly graspable, though might take some tinkering if I actually have to accomplish something above-basic. Have downloaded the programs.

31.10: Looked into some spreadsheets that may be usable for a project. Think I may have to settle for 'doable' this time around.

Assignment 1: The full expression for extracting all dates from the text bit is:

(\d+).(\d+)..?(\d{4})

You need to highlight the full dates (rather than just each individual number), hence the '+'. The '?' is necessary in order to catch the space separating the comma and the year in the first date. But the '?' will also catch the first numbers from the year in the other dates. Thus in the last expression you need to set the {value} at 4 so as to catch the entire number.

The parentheses are necessary for rearranging the dates, as they group the individual expressions and allow you to swap them in the substitution field:

$3-$1-$2

Assignment 2: Converting Voyant to R you just substitute the line break (\n) with ", ". You'll need to manually insert the first and last "s. Converting R to Voyant follows the same procedure in reverse. Substitute ", " with \n. If you want to catch the line breaks in the original text (as between "kasseoverskuddet" and "uafhængiges") in your expression, you can insert |", \n", but it might be less work just correcting those details manually.

Week 45

02.11: Looking into OpenRefine. Seems manageable enough. Usable if I end up needing to rearrange or clean up data one way or another.

03.11: Did the first peer-review. Certainly informative to see how other people approach and convey the assignments. Still looking for data.

04.11: Not having a ton of luck with the sources suggested in Monday's lecture so trying for looser searches. Found a site called Kaggle, which has a bunch of datasets. Found a set containing full transcripts of Trump rally speeches, which I've downloaded. Think I might settle on those for my project.

08.11: Idea for a project concerning those Trump speeches. If I could make a program that could sort through all those speeches for one particular term (and deviations of it), this could be useful. Like if I were to find out how Trump treats Europe in his rallies, I could use the program to sort through all occurrences of Europe (and Europe's, European, etc.) so I don't have to do it manually.

Assignment 1: The most important general rule is probably consistency, making sure you always type in your data in one particular way to make it as readily readable and manageable as possible, for people and computers both. Make sure the spreadsheet doesn't turn into a notebook but rather reflects organized data manageable by a computer, and if possible format the spreadsheet so that each column represents one single observation. Also keep useable metadata so as not to lose track of the information in the spreadsheet (and so others can understand the data). Finally one should remember that a computer doesn't read things like we do, so we should avoid actions that might confuse it. One shouldn't include more than one table per spreadsheet, and one must be aware of how computers read information and lack hereof (a zero is not the same as a blank space, for instance).

Assignment 2: No. Sorting and filtering changes the organization of the data, but the data as such remain unaltered.

Assignment 3: After transforming the data (removing brackets, quote marks and spaces) I was able to make a facet showing the counts for each entry. The answer to the question is October and September.

Assignment 4: By sorting the latitude and longitude columns by smallest numbers, we see that 49 is grouped with the village Chirodzo. Sorting interviews by date yields the same result, so it's a fair assumption that this is in fact the village in question.

Week 46

10.11: Worked through some of the Shell stuff today. Give up on trying to get it all done. Not exactly difficult, but tiresome, and some way in I lost track of what it was all about. I'll definitely need to keep the guide handy if I'm going to use this stuff.
12.11: Read feedback on the last assignment. Have to be more detailed in my assignment solutions.
13.11: Wrote Adela about my project idea. Seems you can pretty much do what I aim to do in Voyant, so need to approach it differently if it is to make much sense. Second take: Rather than making a program, I could try using stopword lists in R or Voyant to isolate Trump's key issues and then try and analyze these. This would seem like it could be helpful in getting to understand Trump as a political phenomenon. Made first half of this week's assignment and tried to be as detailed as possible this time.

Assignment 1: I go to the terminal and write cd Desktop/HW to get into the relevant directory. I want to use the wc command to create a word count of all files in the directory: wc * (* means it counts all files). But I specifically need to find the *largest* file. I think the safest way to account for file size is to count the bytes, so I add an option –c (counts the amount of bytes in each file). The full command goes:

```
wc –c *
```

I don't just want the list, though. I want it sorted so that the files are listed according to file size. I do this by adding a pipe | (which lets me add a second command relying on the first one). I want to sort the list by number (–n) and in reversed order (-r) so that the highest number comes first. So far, the entire command goes:

```
wc –c * | sort –nr
```

I don't want this entire list, but only the three largest files (the first three). I make another pipe and use the head command. This command specifies that only a certain number of lines from the top shall be shown. Notice that the list starts with an entry counting the total amount of bytes among all the files, so in order to see the first three specific files I actually need the first 4 (not 3) entries on the list. The final command for making a list of the three largest files thus goes:

```
wc –c * | sort –nr | head –n 4
```

Were I a perfectionist and wanted to remove that total count at the top, I could add another pipe with a command called tail –n 3. This will make the list show only the last three entries (based on the previous pipe of course).

```
wc –c * | sort –nr | head –n 4 | tail –n 3
```

If finally I wanted to put this in a document which I could send to my supervisor, I'd do like this:

```
wc –c * | sort –nr | head –n 4 | tail –n 3 > jerk.txt
```

This would create said .txt file in the working directory. Don't forget to check the content of the file by using the cat command.

```
cat jerk.txt
```

Assignment 2: As to finding the empty files, I'll assume the find command may come in handy. I want to find the files in the current folder with a size of 0 bytes, so I make the following command:

```
find . –size 0
```

The punctuation narrows the search to current folder, the hyphen signals an option specifying the search criterion (size), and the 0 further specifies that I want only files of this particular size. I hereby get a list of all empty photo files in the directory. Now, I'm not completely sure what's meant by 'count them'. The above command does provide a list of all the files, so there's that. If I want to know how many files the list contains, I'll start by making the computer read the list generated by the find command as an actual list using the ls command.

```
ls $(find . –size 0)
```

the $( ) makes it so the ls command uses the contend of the parenthesis as its input. Each file makes up a single line in this list, which means that by counting the amount of lines on it, we in effect get a count of the size 0 files. I use a pipe to add the wc command:

```
ls $(find . –size 0) | wc –l
```

It says there's 73 lines, which means there should be 73 files. There might be more elegant ways of accomplishing this. Anyway, to make a list of the file names, I simply add a > command to the list command:

```
ls $(find . –size 0) > whatevs.txt
```

I've now created a document containing a list of the files.

Week 47

16.11: Playing around with the Trump speeches on Voyant, which apparently was used another semester. Got to figure out what I can do with it.

17.11: Looking into R, but with my computer not being able to run it properly and there being a monthly time limit on the online version, I think I'd better stick to Voyant as far as my project is concerned.

I think I've landed on a problem worth pursuing: How does Trump depict the U.S.'s global position and its role as a global actor?

20.11: Started on my paper. Noticed the syllabus states the resources should be those used in class, so I should probably clear up whether I can in fact stick to Voyant.

Assignment 1: I start by opening RStudio and here I create the vector 'rooms' with the value listed in the assignment: [rooms <- c(bunch of numbers)].

I then want to omit the NA entries. For this I'll use the na.omit function. I create a new vector called rooms_no_na and assign it the value using the na.omit function combined with the rooms value:

```
rooms_no_na <- na.omit(rooms)
```

I check the output for rooms_no_na, and sure enough, the NA's are removed. Next I want to figure out how many elements in the vector are greater than 2. I try out the following command:

```
rooms_no_na > 2
```

This produces a list of TRUEs and FALSEs indicating whether each entry in rooms_no_na is above 2 or not. Which is not what I wanted. In order to extract values from a vector, I need to subset the vector (indicated by brackets). I single out the vector, and then in the bracket I indicate what part of the vector I need. Looks like this:

```
rooms_no_na[rooms_no_na > 2]
```

The program needs you to type in the vector name twice to understand that this is the sum it's operating on. Now of course we want this new info in its own vector:

```
rooms_over_2 <- rooms_no_na[rooms_no_na > 2]
```

The rooms_over_2 vector contains a list of all room entries above 2. Now in order to figure out how many elements this vector contains, I simply use the 'lenghts' function, which tells me exactly this:

```
length(rooms_over_2)
```

The answer is 8.

Assignment 2: Now I want to find the average number of rooms. For this I'll use the 'mean' function and my trusty rooms_no_na vector again. No reason to remove the NA's again. For good measure, I try using the mean function with the original rooms vector.

```
mean(rooms)
```

Predictably, the output is NA, because all elements in the vector aren't the same type of data, upsetting the program. So of course I use the aforementioned rooms_no_na:

```
mean(rooms_no_na)
```

The answer is 2.318182.

Assignment 3: I want to find out what data type is contained within the rooms vector. For this I simply use the 'class' function with the vector, which shows me just this:

```
class(rooms)
```

The answer is 'numeric'.

Assignment 4: Screenshot has been submitted to my github repository.
https://github.com/Digital-Methods-HASS/au451808_Adam_Isaksen

Week 48

23.11: Adela has indeed confirmed that I need to use R in order to meet course requirements; so will have to work around the monthly time limit. Looked into text mining with R and turns out there's a package called 'tm' designed for the purpose.

24.11: Been figuring out the 'tm' package. Luckily found a good introduction by some Kailash Awati. Think I can pretty much do what I want just by sticking to the functions he introduces.

25.11: Been tinkering with my data in R. It's all worked out pretty easily, no real head scratchers. Hopefully it'll all keep going smoothly.

26.11: Having gotten the programming working, it's pretty much just writing from this point on.

Week 49

01.12: So apparently Adela wasn't just making suggestions when she said to make a comparison rather than focus on just one corpus. I've found a bunch of democrat and republican speeches, may be able to compare discourses.

02.12: In so far as I'm going to use the democrat speeches in my comparison, I've decided it indeed makes more sense to compare them to the republican speeches than the Trump ones, partly because they share a more similar format, partly because Trump's language is rather inconclusive (would you believe it). Hopefully I will still be able to use most of what I've written, assuming I won't have to change my methodical approach. Adjusted my problem to the new data, making it a general comparison of the two parties' discourses.

03.12: Think I just got the point of that final_ver2 cartoon. Assuming there's no more hiccups, it's *really* just down to writing from this point on.