STAT 231: Problem Set 5A

Evan Daisy

due by 10 PM on Monday, March 22

In order to most effectively digest the textbook chapter readings – and the new R commands each presents – series A homework assignments are designed to encourage you to read the textbook chapters actively and in line with the textbook's Prop Tip of page 33:

"**Pro Tip**: If you want to learn how to use a particular command, we highly recommend running the example code on your own"

A more thorough reading and light practice of the textbook chapter prior to class allows us to dive quicker and deeper into the topics and commands during class. Furthermore, learning a programming lanugage is like learning any other language – practice, practice, practice is the key to fluency. By having two assignments each week, I hope to encourage practice throughout the week. A little coding each day will take you a long way!

Series A assignments are intended to be completed individually. While most of our work in this class will be collaborative, it is important each individual completes the active readings. The problems should be straightforward based on the textbook readings, but if you have any questions, feel free to ask me!

Steps to proceed:

1. In RStudio, go to File > Open Project, navigate to the folder with the course-content repo, select the course-content project (course-content.Rproj), and click "Open"
2. Pull the course-content repo (e.g. using the blue-ish down arrow in the Git tab in upper right window)
3. Copy ps5A.Rmd from the course repo to your repo (see page 6 of the GitHub Classroom Guide for Stat231 if needed)
4. Close the course-content repo project in RStudio
5. Open YOUR repo project in RStudio
6. In the ps5A.Rmd file in YOUR repo, replace "YOUR NAME HERE" with your name
7. Add in your responses, committing and pushing to YOUR repo in appropriate places along the way
8. Run "Knit PDF"
9. Upload the pdf to Gradescope. Don't forget to select which of your pages are associated with each problem. *You will not get credit for work on unassigned pages (e.g., if you only selected the first page but your solution spans two pages, you would lose points for any part on the second page that the grader can't see).*

1. Text as Data

NOTE: For this chapter, we'll be working with Chapter 19 in the 2nd edition of the textbook.

a.

In Section 19.1.1, the `str_subset`, `str_detect`, and `str_which` functions are introduced for detecting a pattern in a character vector (like finding a needle in a haystack). Explain what the 6 returned records indicate in each case of these three cases:

- `head(str_subset(macbeth, "  MACBETH"))`
- `head(str_which(macbeth, "  MACBETH"))`
- `head(str_detect(macbeth, "  MACBETH"))`

(Yes, the textbook explains the differences in these commands/calls to these commands, but it can be helpful if you run the lines yourself as well to be sure they work as you'd expect and to inspect the results.)

> ANSWER: str_subset returns a list of strings (lines that have the text " MACBETH" in them), whereas str_which returns a vector of numbers of the lines that contain " MACBETH" in them and str_detect returns a vector of Booleans that describe whether or not a certain line contains " MACBETH" (one boolean for each line in the text file, for example the first 6 are 'FALSE').

```
# defining "macbeth" object
macbeth_url <- "http://www.gutenberg.org/cache/epub/1129/pg1129.txt"
Macbeth_raw <- RCurl::getURL(macbeth_url)
data(Macbeth_raw)
#Macbeth_raw

# strsplit returns a list: we only want the first element
macbeth <- stringr::str_split(Macbeth_raw, "\r\n")[[1]]
class(macbeth)
```

```
## [1] "character"
```

```
length(macbeth)
```

```
## [1] 3194
```

```
### finding literal strings
head(str_subset(macbeth, "  MACBETH"))
```

```
## [1] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [2] "  MACBETH. So foul and fair a day I have not seen."
## [3] "  MACBETH. Speak, if you can. What are you?"
## [4] "  MACBETH. Stay, you imperfect speakers, tell me more."
## [5] "  MACBETH. Into the air, and what seem'd corporal melted"
## [6] "  MACBETH. Your children shall be kings."
```

```
head(str_which(macbeth, "  MACBETH"))
```

```
## [1] 228 433 443 466 478 483
```

```
head(str_detect(macbeth, "  MACBETH"))
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

b.

Section 19.1.1 also introduces regular expressions. Why do the two lines below differ in their results?

- `head(str_subset(macbeth, "MACBETH\\."))`
- `head(str_subset(macbeth, "MACBETH."))`

(Yes, the textbook explains the differences, but it can be helpful if you run the lines yourself as well to be sure they work as you'd expect and to inspect the results.)

> ANSWER: The results of these two lines differ in that the first returns a subset of lines containing the text "MACBETH.", whereas the second returns a subset of lines containing the text "MACBETH" followed by any other character. This is because the symbol "." in a regular expression is like a wild card; it can represent any symbol. In the first line, this is overridden by the two forward slashes, which in R ensures that the period is searched for and treated as a symbol in itself.

```
head(str_subset(macbeth, "MACBETH\\."))
```

```
## [1] "  MACBETH. So foul and fair a day I have not seen."
## [2] "  MACBETH. Speak, if you can. What are you?"
## [3] "  MACBETH. Stay, you imperfect speakers, tell me more."
## [4] "  MACBETH. Into the air, and what seem'd corporal melted"
## [5] "  MACBETH. Your children shall be kings."
## [6] "  MACBETH. And Thane of Cawdor too. Went it not so?"
```

```
head(str_subset(macbeth, "MACBETH."))
```

```
## [1] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [2] "  LADY MACBETH, his wife"
## [3] "  MACBETH. So foul and fair a day I have not seen."
## [4] "  MACBETH. Speak, if you can. What are you?"
## [5] "  MACBETH. Stay, you imperfect speakers, tell me more."
## [6] "  MACBETH. Into the air, and what seem'd corporal melted"
```

c.

The following three commands look similar, but return different results.

- `head(str_subset(macbeth, "MAC[B-Z]"))`

- `head(str_subset(macbeth,"MAC[B|Z]"))`
- `head(str_subset(macbeth, "^MAC[B-Z]"))`

In words, explain what overall pattern is being searched for in each of the three cases (i.e., what do the patterns "MAC[B-Z]", "MAC[B|Z]", and "^MAC[B-Z]" indicate?)?

> ANSWER: The first line is searching for "MAC" followed by any letter between B and Z (not A), whereas the second line is searching for "MAC" followed by either "B" or "Z" and the third line is searching for "MAC" followed by any letter between B and Z but it has to be at the beginning of a line.

```
head(str_subset(macbeth, "MAC[B-Z]"))
```

```
## [1] "MACHINE READABLE COPIES MAY BE DISTRIBUTED SO LONG AS SUCH COPIES"
## [2] "MACHINE READABLE COPIES OF THIS ETEXT, SO LONG AS SUCH COPIES"
## [3] "WITH PERMISSION.  ELECTRONIC AND MACHINE READABLE COPIES MAY BE"
## [4] "THE TRAGEDY OF MACBETH"
## [5] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [6] "  LADY MACBETH, his wife"
```

```
head(str_subset(macbeth,"MAC[B|Z]"))
```

```
## [1] "THE TRAGEDY OF MACBETH"
## [2] "  MACBETH, Thane of Glamis and Cawdor, a general in the King's"
## [3] "  LADY MACBETH, his wife"
## [4] "  MACBETH. So foul and fair a day I have not seen."
## [5] "  MACBETH. Speak, if you can. What are you?"
## [6] "  MACBETH. Stay, you imperfect speakers, tell me more."
```

```
head(str_subset(macbeth, "^MAC[B-Z]"))
```

```
## [1] "MACHINE READABLE COPIES MAY BE DISTRIBUTED SO LONG AS SUCH COPIES"
## [2] "MACHINE READABLE COPIES OF THIS ETEXT, SO LONG AS SUCH COPIES"
```

```
# (optional) other patterns to test out:
# head(str_subset(macbeth, "^MAC[B|Z]"))  # should return character(0) (nothing)
# head(str_subset(macbeth, ".*MAC[B-Z]"))
# head(str_subset(macbeth, ".MAC[B-Z]"))
# head(str_subset(macbeth, "more$"))
```

d. OPTIONAL

In section 19.2.2, the `wordcloud` package is used to create a word cloud based on text in abstracts from Data Science articles in arXiv (which is "a fast-growing electronic repository of preprints of scientific papers from many disciplines"). I've provided some code below to get you started coding along with the example. *This part (d) will not be graded, but is included to encourage you to test and explore some of the code in the extended example.* What words are included in `tidytexts`'s `stop_words` dataset? Do you think all of these words should be considered stop words (i.e. excluded from analysis) in all scenarios? Are there any that might be useful in some contexts?

#"'{r} # note that the tidytext, aRxiv, and wordcloud packages were loaded in the # setup code chunk at the top of the program

arxiv_search() is from the aRxiv package

DataSciencePapers <- arxiv_search(query = ‘ “Data Science” ’, limit = 20000, batchsize = 100)

glimpse(DataSciencePapers)

DataSciencePapers <- DataSciencePapers %>% mutate(submitted = lubridate::ymd_hms(submitted) , updated = lubridate::ymd_hms(updated) , field = str_extract(primary_category, “[1]+”) , Field = ifelse(field == “cs”, “Computer Science”, “Other discipline”))

stop words dataset provided by the tidytext package

stop_words <- tidytext::stop_words

the unnest_tokens function is from the tidytext package

arxiv_words <- DataSciencePapers %>% unnest_tokens(output = word, input = abstract, token = “words”) %>% anti_join(stop_words, by = “word”) %>% select(word, id) %>% filter(word != “the”)

arxiv_word_freqs <- arxiv_words %>% count(id, word, sort = TRUE) %>% select(word, n, id)

the wordcloud function is from the wordcloud package

you may also need to install the “tm” package in order to use the function

set.seed(1962) wordcloud(DataSciencePapers$abstract, max.words = 30, scale = c(8, 1), colors = topo.colors(n = 30), random.color = TRUE) “‘

[1] a-z,-