Hw6

Cody

November 6, 2017

Stat 547M Homework 6:

Goal: Pick (at least) two of the six (numbered) topics below and do one of the exercise prompts listed, or something comparable using your dataset of choice.

```
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
library(stringr)
```

1. Character data

Read and work the exercises in the Strings chapter or R for Data Science.

Exercises 1

1. In code that doesn't use stringr, you'll often see paste() and paste0(). What's the difference between the two functions? What stringr function are they equivalent to? How do the functions differ in their handling of NA?

A: paste() and paste0() have a different default value set for the 'sep' arguement.

```
paste("hello","world") #Default sep=" "
```

```
## [1] "hello world"
```

```
paste0("hello","world") #Default sep=""
```

```
## [1] "helloworld"
```

The stringr function is str_c()

```
str_c("hello","world")
```

```
## [1] "helloworld"
```

So we see that the default separator is also sep="". If NAs are involved, str_c() will refuse to join where the others will treat them as strings. The NA has to be dealt with with a different function, str_replace_na(). This will turn an NA into a string "NA". Once this is done, then all functions behave similar.

```r
paste("Hello",NA)
```

```
## [1] "Hello NA"
```

```r
paste0("Hello",NA)
```

```
## [1] "HelloNA"
```

```r
str_c("Hello",NA)
```

```
## [1] NA
```

```r
paste("Hello",str_replace_na(NA))
```

```
## [1] "Hello NA"
```

```r
paste0("Hello",str_replace_na(NA))
```

```
## [1] "HelloNA"
```

```r
str_c("Hello",str_replace_na(NA))
```

```
## [1] "HelloNA"
```

2. In your own words, describe the difference between the sep and collapse arguments to str_c().

Sep will induce the type of spacing between the joined strings. For example,

```r
str_c(c("This","an"),c("is","example"),sep="+")
```

```
## [1] "This+is"    "an+example"
```

But clearly we would have wanted this to be one string and this is where the collapse arguement will put the separate strings together.

```r
str_c(c("This","an"),c("is","example"),collapse="+")
```

```
## [1] "Thisis+anexample"
```

Together, we can make the correct string,

```r
str_c(c("This","an"),c("is","example"),sep=" ",collapse=" ")
```

```
## [1] "This is an example"
```

3. Use str_length() and str_sub() to extract the middle character from a string. What will you do if the string has an even number of characters?

A: For the even length string, I'd return both values around the middle value.

```r
exampleodd <- "Strings"
exampleeven <- "Cody"

oddlength <- str_length(exampleodd)
oddanswer <- ceiling(oddlength/2)
evenlength <- str_length(exampleeven)
evenanswer <- evenlength/2

str_sub(exampleodd,oddanswer,oddanswer)
```

```
## [1] "i"
```

```
str_sub(exampleeven,evenanswer,evenanswer+1)
```

```
## [1] "od"
```

4. What does str_wrap() do? When might you want to use it?

A: str_wrap is a way to break a string into smaller pieces but by following a special algorithm for keeping words **together** "Knuth-Plass wrapping". This is useful for extracting information out of larger strings like a book or article for example.

5. What does str_trim() do? What's the opposite of str_trim()?

A: This trims whitespace from strings.

```
string <- "           hi               "
trimmed <-str_trim(string)
padded <- str_pad(trimmed,width=10,side="both")

string
```

```
## [1] "           hi               "
```

```
trimmed
```

```
## [1] "hi"
```

```
padded
```

```
## [1] "    hi    "
```

6. Write a function that turns (e.g.) a vector c("a", "b", "c") into the string a, b, and c. Think carefully about what it should do if given a vector of length 0, 1, or 2.

A: This function will return an error for length 0, just the vector for length 1 and will combine following the above rule for any vector of size >=2.

```
my_func <- function(string_vec){
  L <- length(string_vec)
  if(L==0) {
    stop('The length of the string is 0')
  }
  if(L==1){
    return(string_vec)
  }
  firstpart <-
    str_c(string_vec[1:L-1], collapse = ", ")
  secondpart <-
    str_c("and", string_vec[L], sep = " ")
  final <-
    str_c(firstpart, secondpart, sep = ", ")
  return(final)
}
```

To test it:

```
test <- letters[1:10]
my_func(test)
```

```
## [1] "a, b, c, d, e, f, g, h, i, and j"
```

Exercises 2

1. Explain why each of these strings don’t match a : “",”\“,”\".

“" Doesn’t work because a backslash is recognized as the beginning of a special character. This will look for the next character.”\" Doesn’t work because this now escapes from itself and creates a plain backslash, not a regular expression. “\" Now has the issue of forming a single pair to create a backslash but the unpaired backslash now is looking at the next character.

2. How would you match the sequence “’ ?

```
x <- "\"\'\\"
writeLines(x)
```

```
## "'\
```

3. What patterns will the regular expression …… match? How would you represent it as a string?

This will match anything that looks like “.a.a.a”.

```
x <- "\\..\\..\\.."
writeLines(x)
```

```
## \..\..\..
```

Exercises 3

1. How would you match the literal string “$^$”

```
x <- "$^$"
writeLines(x)
```

```
## $^$
```

```
str_match(x,"\\$\\^\\$")
```

```
##      [,1]
## [1,] "$^$"
```

2. Create regular expressions that find:

- Start with “y”
- Ends with “x”
- Are exactly three letters long
- Have seven letters or more

```
test <- c("york","latex","abc","a lot of letters")
str_match(test,"^y")
```

```
##      [,1]
## [1,] "y"
## [2,] NA
## [3,] NA
## [4,] NA
```

```
str_match(test,"x$")
```

```
##      [,1]
## [1,] NA
## [2,] "x"
## [3,] NA
## [4,] NA
```

```
str_match(test,"^...$")
```

```
##      [,1]
## [1,] NA
## [2,] NA
## [3,] "abc"
## [4,] NA
```

```
str_match(test,".......")
```

```
##      [,1]
## [1,] NA
## [2,] NA
## [3,] NA
## [4,] "a lot o"
```

Exercises 4

1. Find all words that contain:

- Start with a vowel
- Only consonants
- End with "ed" but not with eed
- End with "ing" or "ise"

```
test <- c("ao","qwrt","bed","beed","matching","ise")
str_match(test,"^[aeiou]")
```

```
##      [,1]
## [1,] "a"
## [2,] NA
## [3,] NA
## [4,] NA
## [5,] NA
## [6,] "i"
```

```
str_match(test,str_c(rep("[^aeiou]",4),collapse=""))
```

```
##      [,1]
## [1,] NA
## [2,] "qwrt"
## [3,] NA
## [4,] NA
## [5,] NA
## [6,] NA
```

```
str_match(test,"[^e][e][d]$")
```

```
##      [,1]
## [1,] NA
## [2,] NA
## [3,] "bed"
## [4,] NA
## [5,] NA
## [6,] NA
```

```
str_match(test,"([i][n][g])|([i][s][e])$")
```

```
##      [,1]  [,2]  [,3] 
## [1,] NA    NA    NA   
## [2,] NA    NA    NA   
## [3,] NA    NA    NA   
## [4,] NA    NA    NA   
## [5,] "ing" "ing" NA   
## [6,] "ise" NA    "ise"
```

Exercises 5

1. Desribe the equivalences of “?”, “+” and “*" in {m,n} format.

A: ? - {0,1}, + - {1,}, * - {0,}

5. Work with a list

Here I will go through Trump Android Tweets

Loading the data

```
library(purrr)
suppressMessages(library(dplyr))
library(tibble)

load(url("http://varianceexplained.org/files/trump_tweets_df.rda"))
glimpse(trump_tweets_df)
```

```
## Observations: 1,512
## Variables: 16
## $ text          <chr> "My economic policy speech will be carried live ...
## $ favorited     <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,...
## $ favoriteCount <dbl> 9214, 6981, 15724, 19837, 34051, 29831, 19223, 1...
## $ replyToSN     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ created       <dttm> 2016-08-08 15:20:44, 2016-08-08 13:28:20, 2016-...
## $ truncated     <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,...
## $ replyToSID    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ id            <chr> "762669882571980801", "762641595439190016", "762...
## $ replyToUID    <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ statusSource  <chr> "<a href=\"http://twitter.com/download/android\"...
## $ screenName    <chr> "realDonaldTrump", "realDonaldTrump", "realDonal...
## $ retweetCount  <dbl> 3107, 2390, 6691, 6402, 11717, 9892, 5784, 7930,...
## $ isRetweet     <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,...
## $ retweeted     <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,...
## $ longitude     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ latitude      <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
```

```
tweets <- trump_tweets_df$text
tweets %>% head() %>% strtrim(70)
```

```
## [1] "My economic policy speech will be carried live at 12:15 P.M. Enjoy!"
## [2] "Join me in Fayetteville, North Carolina tomorrow evening at 6pm. Ticke"
## [3] "#ICYMI: \"Will Media Apologize to Trump?\" https://t.co/ia7rKBmioA"
## [4] "Michael Morell, the lightweight former Acting Director of C.I.A., and "
## [5] "The media is going crazy. They totally distort so many things on purpo"
## [6] "I see where Mayor Stephanie Rawlings-Blake of Baltimore is pushing Cro"
```

```
regex <- "badly|crazy|weak|spent|strong|dumb|joke|guns|funny|dead"
```

Here we select a few rows with the correct complexity to learn from

```
tweets <- tweets[c(1, 2, 5, 6, 198, 347, 919)]
tweets %>% strtrim(70)
```

```
## [1] "My economic policy speech will be carried live at 12:15 P.M. Enjoy!"
## [2] "Join me in Fayetteville, North Carolina tomorrow evening at 6pm. Ticke"
## [3] "The media is going crazy. They totally distort so many things on purpo"
## [4] "I see where Mayor Stephanie Rawlings-Blake of Baltimore is pushing Cro"
## [5] "Bernie Sanders started off strong, but with the selection of Kaine for"
## [6] "Crooked Hillary Clinton is unfit to serve as President of the U.S. Her"
## [7] "The Cruz-Kasich pact is under great strain. This joke of a deal is fal"
```

In base r

```
matches <- gregexpr(regex, tweets)
str(matches)
```

```
## List of 7
##  $ : atomic [1:1] -1
##   ..- attr(*, "match.length")= int -1
##   ..- attr(*, "useBytes")= logi TRUE
##  $ : atomic [1:1] -1
##   ..- attr(*, "match.length")= int -1
##   ..- attr(*, "useBytes")= logi TRUE
##  $ : atomic [1:1] 20
##   ..- attr(*, "match.length")= int 5
##   ..- attr(*, "useBytes")= logi TRUE
##  $ : atomic [1:1] 134
##   ..- attr(*, "match.length")= int 4
##   ..- attr(*, "useBytes")= logi TRUE
##  $ : atomic [1:2] 28 95
##   ..- attr(*, "match.length")= int [1:2] 6 4
##   ..- attr(*, "useBytes")= logi TRUE
##  $ : atomic [1:2] 87 114
##   ..- attr(*, "match.length")= int [1:2] 4 6
##   ..- attr(*, "useBytes")= logi TRUE
##  $ : atomic [1:3] 50 112 123
##   ..- attr(*, "match.length")= int [1:3] 4 4 4
##   ..- attr(*, "useBytes")= logi TRUE
```

```
matches[[7]]
```

```
## [1]  50 112 123
## attr(,"match.length")
## [1] 4 4 4
## attr(,"useBytes")
## [1] TRUE
```

Base r isn't great. matches is quite ugly and hard to deal with.

```r
lengths(matches) #happens to exist
```

```
## [1] 1 1 1 1 2 2 3
```

```r
sapply(matches,length) #not friendly
```

```
## [1] 1 1 1 1 2 2 3
```

```r
vapply(matches,length,integer(1)) #base approach
```

```
## [1] 1 1 1 1 2 2 3
```

```r
map_int(matches,length) #purrr way
```

```
## [1] 1 1 1 1 2 2 3
```

Exercise: Get a list of the match lengths

```r
match_length <-
  map(matches,~ attr(.x, which = "match.length"))
match_length
```

```
## [[1]]
## [1] -1
##
## [[2]]
## [1] -1
##
## [[3]]
## [1] 5
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] 6 4
##
## [[6]]
## [1] 4 6
##
## [[7]]
## [1] 4 4 4
```

Exercise: Count "Trump words"

Write a function that will count the number of these "Trump words".

```r
f <- function(x) sum(x>0)
map(matches,f)
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 0
```

```
##
## [[3]]
## [1] 1
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 2
##
## [[6]]
## [1] 2
##
## [[7]]
## [1] 3
```

But a more compact way to get a vector of integers, we can use the purrr function:

```
map_int(matches,~sum(.x>0))
```

```
## [1] 0 0 1 1 2 2 3
```

Excersise: Strip the attributes from matches

Here we use the as.vector() function which will throw away non-vectorizable structure

```
match_first <- map(matches,as.vector)
str(match_first)
```

```
## List of 7
##  $ : int -1
##  $ : int -1
##  $ : int 20
##  $ : int 134
##  $ : int [1:2] 28 95
##  $ : int [1:2] 87 114
##  $ : int [1:3] 50 112 123
```

Exercise: Find the final word that fits the "Trump word" criteria

```
match_last <- map2(match_first, match_length, ~ .x+.y-1)
match_last
```

```
## [[1]]
## [1] -3
##
## [[2]]
## [1] -3
##
## [[3]]
## [1] 24
##
## [[4]]
## [1] 137
```

```
##
## [[5]]
## [1] 33 98
##
## [[6]]
## [1]  90 119
##
## [[7]]
## [1]  53 115 126
```

Put it all together using pmap

This will find all the "Trump words" in a tweet and pull them out.

```
pmap(list(text=tweets,first=match_first,last=match_last),.f=substring)
```

```
## [[1]]
## [1] ""
##
## [[2]]
## [1] ""
##
## [[3]]
## [1] "crazy"
##
## [[4]]
## [1] "joke"
##
## [[5]]
## [1] "strong" "weak"
##
## [[6]]
## [1] "weak"   "strong"
##
## [[7]]
## [1] "joke" "dead" "dumb"
```

Just to put it all in one place,

```
trump_frame <-tibble(
  text = tweets,
  first = match_first,
  last = match_last
)
pmap(trump_frame, substring)
```

```
## [[1]]
## [1] ""
##
## [[2]]
## [1] ""
##
## [[3]]
## [1] "crazy"
##
```

```
## [[4]]
## [1] "joke"
##
## [[5]]
## [1] "strong" "weak"
##
## [[6]]
## [1] "weak"   "strong"
##
## [[7]]
## [1] "joke" "dead" "dumb"
```

Reflection

! Regular expressions are rough! I've learned them now two or three times and they're never fun! I didn't work through every single example as there are so many but I tried to do at least one in each section. I still probably need a formal lecture on some of the specifics, like grouping and backreferencing.

As for using purrr functions over strings, I found this to be incredibly easy and I can see why there are better alternatives to base R.