Working with Strings and Stringr

Introduction

This Thanksgiving, I decided to do a little traveling with a close friend of mine. After doing some research, we decided that we wanted to go to *Seattle*! As per all my previous trips, the first thing we did after booking our plane tickets was to look at things to do in Seattle. My friend and I are camping enthusiasts, and we wanted to buy a tent that would be able to stand tall in the Seattle weather that is sometimes cold, windy, and rainy.

The tools that you will need

For today's tutorial, you will need several packages that will help you in your data analysis. Let's start off by adding some packages to our library. These packages should not be unfamiliar to most of you:

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.2.5
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(magrittr)
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 3.2.5
```

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 3.2.5
```

```
library(xtable)
```

These tools will come in handy as we clean and manipulate the data set.

The dataset that we will be working with is available in our stat133 github repository (*how very convenient!*) - it is a camping dataset that has information on brand, weight, height, function, and number of people the tent fits. Let's download the data by using these lines of code, and save it with the file name "tents.csv":

```
github <- "https://raw.githubusercontent.com/ucb-stat133/stat133-fall-2017/master/"
csv <- "data/camping-tents.csv"
download.file(url = paste0(github, csv), destfile = 'tents.csv')
```

Let's create an object *tents* that captures the tents.csv dataset:

```
tents <- read.csv("tents.csv")

tents2 <- tents
head(tents)
```

```
##              name     brand  price weight height     bestuse  seasons
## 1   fly-creek-ul2 big-agnes 349.95    960     96 Backpacking 3-season
## 2   fly-creek-ul3 big-agnes 449.95   1450    107 Backpacking 3-season
## 3        salida-2     kelty 159.95   1700    102 Backpacking 3-season
## 4 jack-rabbit-sl3 big-agnes 359.95   2160    107 Backpacking 3-season
## 5       passage-2       rei 149.00   2210    107 Backpacking 3-season
## 6 copper-spur-ul2 big-agnes 399.95   1530    107 Backpacking 3-season
##   capacity
## 1 2-person
## 2 3-person
## 3 2-person
## 4 3-person
## 5 2-person
## 6 2-person
```

Things we have learned in class

A first look at the raw data file reveals that for some of the variables like seasons and capacity, there is a mix of numerics and characters. If the dataset were any larger, it might be a little cluttered to work through it.

substr()

The substr() function can help us look at just the numeric part, which is what is important in this data set.

```
substr(tents$seasons, start = 1, stop = 1)
```

```
##  [1] "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3"
## [18] "3" "3" "3" "3" "3" "3" "4" "3" "3" "3" "3" "3" "3" "3" "4" "4" "3"
## [35] "3" "3" "4" "3" "3" "3" "3" "3" "3" "3" "4" "3" "3" "3" "3" "3" "3"
## [52] "3" "3" "3" "3" "4" "3" "3" "3" "3" "4" "3" "3" "3" "3" "3" "3" "3"
## [69] "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "4" "3" "3" "3" "3"
## [86] "3" "3" "3" "3" "3"
```

If you wanted to extract another part of the string, all you have to do is change the 'start' and 'stop' portions of the function.

```
substr(tents$seasons, start = 2, stop = 3)
```

```
##  [1] "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s"
## [15] "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s"
## [29] "-s" "-s" "-s" "-s" "-s" "-s" "-4" "-4" "-s" "-4" "-s" "-s" "-s" "-s"
## [43] "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s"
## [57] "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s"
## [71] "-s" "-s" "-s" "-4" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s" "-s"
## [85] "-s" "-s" "-s" "-s" "-s" "-s"
```

Notice that substr() counts by characters, not words! Notice also that the output of substr() is characters. Hence, in order to modify the column, we need to add in a few additional functions:

```
tents$seasons <- as.numeric(substr(tents$seasons, start = 1, stop = 1))
```

str_split()

Another function that we've learned that we can do the same thing with is str_split(). Let's perform this function on the capacity column - you see that we obtain characters of just the number and the 'person' word.

```
cap_split <- str_split(tents$capacity, "-")
```

Similarly, because the output is a character type, we need to do some extra work to modify the capacity column. We start by creating a function that performs the string splitting.

```
cap_number <- c()

for (j in 1:90) {
  cap_number <- c(cap_number, cap_split[[j]][1])
}
cap_number
```

```
##  [1] "2" "3" "2" "3" "2" "2" "3" "4" "4" "1" "2" "2" "2" "2" "1" "2" "3"
## [18] "4" "3" "3" "3" "1" "2" "3" "3" "2" "2" "1" "4" "1" "2" "2" "3" "2"
## [35] "2" "3" "2" "2" "2" "3" "1" "2" "2" "1" "2" "3" "2" "1" "3" "2" "1"
## [52] "2" "3" "4" "3" "2" "2" "2" "2" "2" "2" "2" "3" "3" "2" "6" "4" "6"
## [69] "6" "6" "6" "4" "4" "6" "4" "6" "6" "4" "4" "6" "4" "4" "6" "4" "6"
## [86] "4" "6" "6" "6" "4"
```

```
typeof(cap_number)
```

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

Now that we have just the numbers in character form, we can perform a similar function as we just previously to mutate the current capacity column.

```
tents$capacity <- as.numeric(cap_number)
```

Perfect! Let's take a peek at how the data set has changed.

```
head(tents)
```

```
##              name     brand  price weight height     bestuse seasons
## 1   fly-creek-ul2 big-agnes 349.95    960     96 Backpacking       3
## 2   fly-creek-ul3 big-agnes 449.95   1450    107 Backpacking       3
## 3        salida-2     kelty 159.95   1700    102 Backpacking       3
## 4 jack-rabbit-sl3 big-agnes 359.95   2160    107 Backpacking       3
## 5       passage-2       rei 149.00   2210    107 Backpacking       3
## 6 copper-spur-ul2 big-agnes 399.95   1530    107 Backpacking       3
##   capacity
## 1        2
## 2        3
## 3        2
## 4        3
## 5        2
## 6        2
```

Functions to Introduce - Modifiers

Within Stringr, there exists functions known as 'modifiers' that are used to control the package's matching functions.

boundary()

The first cool one that I wanted to introduce to you is one called "boundary()". This function is often used with *str_split*, which we had just learned how to use! The awesome thing is that with boundary, you can call the variables you want to keep without specifying the things you want to drop. For example, instead of specifying that the "-" is to be dropped, all you have to do is call boundary, and specify that you want to keep the "words".

```
head(str_split(tents2$season, boundary("word")))
```

```
## [[1]]
## [1] "3"      "season"
##
## [[2]]
## [1] "3"      "season"
##
## [[3]]
## [1] "3"      "season"
##
## [[4]]
## [1] "3"      "season"
##
## [[5]]
## [1] "3"      "season"
##
## [[6]]
## [1] "3"      "season"
```

You can actually split not just by word, but also by character, line, and sentence boundaries. This is documented here. This function is especially helpful if the dataset that you are working with has many different separators. Let's create an example and test this function!

```
dat <- data.frame(c("hello?friend", "hello-friend", "hello!friend", "hello, friend"))

str_split(dat[,1], boundary("word"))
```

```
## [[1]]
## [1] "hello"  "friend"
##
## [[2]]
## [1] "hello"  "friend"
##
## [[3]]
## [1] "hello"  "friend"
##
## [[4]]
## [1] "hello"  "friend"
```

See? Same result, but so much simpler. Let's apply it to the names of tents in the original dataset. My friend and I believe that we can recognize products based on just the first word of each tent name, so we want to just keep the first word.

```
name_split <- str_split(tents$name, boundary("word"))

name_vec <- c()

for (j in 1:90) {
  name_vec <- c(name_vec, name_split[[j]][1])
}
name_vec
```

```
##  [1] "fly"         "fly"         "salida"      "jack"        "passage"
##  [6] "copper"      "limelight"   "fly"         "half"        "fly"
## [11] "half"        "quarter"     "camp"        "half"        "lynx"
## [16] "limelight"   "quarter"     "copper"      "copper"      "quarter"
## [21] "zia"         "hubba"       "jack"        "ve"          "losi"
## [26] "seedhouse"   "hubba"       "seedhouse"   "limelight"   "quarter"
## [31] "obi"         "string"      "mountain"    "tadpole"     "cirque"
## [36] "arete"       "mountain"    "arete"       "phoenix"     "carbon"
## [41] "eos"         "lynx"        "carbon"      "morpho"      "mountain"
## [46] "espri"       "espri"       "obi"         "holler"      "losi"
## [51] "carbon"      "meta"        "mutha"       "soda"        "gunnison"
## [56] "fury"        "big"         "gunnison"    "morpho"      "aura"
## [61] "alpinist"    "burn"        "lightning"   "widi"        "fast"
## [66] "king"        "hobitat"     "kingdom"     "limestone"   "base"
## [71] "limestone"   "kingdom"     "big"         "flying"      "base"
## [76] "camp"        "big"         "camp"        "king"        "hobitat"
## [81] "flying"      "asashi"      "trail"       "backcountry" "mountain"
## [86] "trail"       "hula"        "docking"     "hacienda"    "hula"
```

```
tents$name <- name_vec
```

Let's briefly explore two more modifiers that I think are pretty helpful if you are exploring a very text-heavy dataset. One such example is a google form that encourages people to respond in short and long paragraphs. What happens if people have typos in crucial portions that you are trying to scrape, as seen here? Let's try an example with different capitalizations

```
dessert <- c("Ice-cream", "ice-cream", "pIe", "snickerdoodles")
dessert
```

```
## [1] "Ice-cream"      "ice-cream"      "pIe"            "snickerdoodles"
```

```
str_detect(dessert, "i")
```

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

coll()

Even though all the above variables are desserts that have 'i's in them, using str_detect with the small 'i' returns just two of four desserts. The coll() function helps you look past these minor differences in capitalization and usage of accents in words.

```
str_detect(dessert, coll("i", ignore_case = TRUE))
```

```
## [1] TRUE TRUE TRUE TRUE
```

regex()

Another way that you can achieve this ignore case effect is through the modifier regex(). We briefly talked about regex (regular expressions) in class, but here's a super intuitive cheatsheet that you can use.

```
str_match(dessert, regex("[a-z]+", ignore_case = TRUE))
```

```
##      [,1]
## [1,] "Ice"
## [2,] "ice"
## [3,] "pIe"
## [4,] "snickerdoodles"
```

An Additional Function - str_subset()

str_subset() allows us to keep all the values that match a pattern. If you recall, we tried to do the same previously with str_split() and substr(), but we had to convert the output to numerics / vectors, before mutating the original dataset with the new column. str_subset() helps us do away with that as exemplified in this link. Let's try out this function with a simple example:

```
x <- c("aPPLE", "BANANA", "CaRAMEL", "DIPS")
str_subset(x, regex("[aBC]", ignore_case = TRUE))
```

```
## [1] "aPPLE"   "BANANA"  "CaRAMEL"
```

Returning to our tents dataset, suppose that my friend and I only like brands that have the alphabet 'i'. Let's use the same function to create a more selective data set.

```
shortlist <-
  filter(tents,
       tents$brand == str_subset(tents$brand, regex("[i]", ignore_case = TRUE)))
```

```
## Warning: package 'bindrcpp' was built under R version 3.2.5
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
## Warning in `==.default`(tents$brand, str_subset(tents$brand, regex("[i]", :
## longer object length is not a multiple of shorter object length
```

Putting It Altogether

Perfect! Let's put all the things we learned together. Today, we learned how to apply a bunch of functions we had gotten from class, and saw how we could supplement those functions with modifiers.

1. str_split() helps us split our input, and gives us an output of character form
2. substr() helps us look at particular parts of a string, and also returns characters
3. The boundary() modifier makes it easier for us to split up our input, when the separators in a single dataframe are not consistent
4. The coll() modifier helps us look past cases and language inconsistencies when we are trying to match patterns
5. The regex() modifier allows us to ignore cases effortlessly, and specify with regex syntax what we would like to keep in the output
6. str_subset() does a similar job to substr(), but is a smarter tool that matches throughout the string, and you don't have to specify that you would like the subset to start and end at particular points

Our current output data set is 'shortlist'. To wrap up our quest to pick a tent, let's create a table with key variables that my friend and I actually care about. We use subsets and filter functions. We also want to do some sorting by price:

```
final <-
  shortlist %>%
  subset(select = -c(weight, height, bestuse)) %>%
  filter(capacity == 2 & seasons == 3) %>%
  arrange(-desc(price))

head(final)
```

```
##      name     brand  price seasons capacity
## 1    half       rei 179.00       3        2
## 2    burn big-agnes 189.95       3        2
## 3    half       rei 199.00       3        2
## 4    lynx big-agnes 199.95       3        2
## 5    jack big-agnes 279.95       3        2
## 6 quarter       rei 299.00       3        2
```

From the output of our analysis today, my friend and I would pick the first tent from REI that is the most affordable (*$179 only*), because it fits us both (capacity = 2), and can weather 3 seasons, including the rainy one (we don't need it for the snow, we hope!) Hopefully from this post, you've managed to catch a glimpse of how powerful stringr and its various embedded tools and modifiers can be! :)

References

- https://github.com/ucb-stat133/stat133-fall-2017/blob/master/data/camping-tents.csv
- https://www.rdocumentation.org/packages/stringr/versions/1.1.0/topics/modifiers

- http://www.fabianheld.com/stringr/
- https://blog.rstudio.com/2015/05/05/stringr-1-0-0/
- https://www.rstudio.com/wp-content/uploads/2016/09/RegExCheatsheet.pdf
- https://blog.rstudio.com/2015/05/05/stringr-1-0-0/
- https://stackoverflow.com/questions/4605206/drop-data-frame-columns-by-name
- https://www.rdocumentation.org/packages/dplyr/versions/0.7.3/topics/filter
- https://stackoverflow.com/questions/1296646/how-to-sort-a-dataframe-by-columns