# HW: Week 3

36-350 – Statistical Computing

Week 3 – Fall 2020

Name: Kyle Wagner

Andrew ID: kowagner

You must submit **your own** lab as a PDF file on Gradescope.

---

```
trump.lines = readLines("http://www.stat.cmu.edu/~pfreeman/trump.txt")
```

---

## Question 1

*(10 points)*

Display the lines of text in `trump.lines` that contain both of the strings "America" *and* "great" (in any order, separated by any amount of text). Do so only using regexp literals.

```
grep("great +America|America +great",trump.lines, value = TRUE)
```

```
## [1] "To all Americans tonight in all of our cities and in all of our towns,
I make this promise -- we will make America strong again. We will make America
proud again. We will make America safe again. And we will make America great
again."
```

## Question 2

*(10 points)*

Retrieve (but don't display) the lines of text in `trump.lines` that contain "Trump". Then break the retrieved lines into individual words (`strsplit(input," ")` to split the character vector `input` into words separated by spaces), and merge those words into a single character vector (`unlist()`!). How many unique words are there? Display the top five most commonly occurring words and how often they occur (combine `sort()` and `table()`!)

```
trump = grep("Trump",trump.lines, value = TRUE)
res = unlist(strsplit(trump," "))
length(res) #168 unique words
```

```
## [1] 168
```

```
sort(table(res),decreasing = TRUE)[1:5] #top five most commonly occurring words + frequency
```

```
## res
##   of  the  and    a love
##    7    7    6    4    4
```

## Question 3

*(10 points)*

In Q25 of Lab 3, you coded a regex to match all patterns of the following form: any letter (1 or more), then a punctuation mark, then "ve" or "ll" or "t", then a space or a punctuation mark. You called it `my.pattern`. Use `my.pattern`, along with `regexpr()` and `regmatches()`, to extract and display all the occurrences of the pattern in `trump.lines`. Then repeat the exercise using `gregexpr()` instead of `regexpr`; note that here, you'll want to `unlist()` the output from `regmatches()`. Do you get the same vector of character strings? Why or why not?

```
my.pattern = "([a-z]|[A-Z])+[[:punct:]](ve|ll|t)( |[[:punct:]])"
reg.xpr <- regexpr(my.pattern,trump.lines)
regmatches(trump.lines,reg.xpr)
```

```
##  [1] "would've " "I've "     "won't "    "don't "    "won't "    "we'll "
##  [7] "I'll "     "don't "    "can't "    "don't "
```

```
reg.xpr1 <- gregexpr(my.pattern,trump.lines)
unlist(regmatches(trump.lines,reg.xpr1))
```

```
##  [1] "would've " "would've " "would've " "I've "     "wasn't "   "won't "
##  [7] "can't "    "don't "    "won't "    "we'll "    "don't "    "I'll "
## [13] "they've "  "don't "    "can't "    "wouldn't " "doesn't "  "don't "
## [19] "Don't "    "don't "
```

```
Do not get the same vector of character strings. This is because regxpr returns only returns the first m
```

## Question 4

*(10 points)*

Come up with a strategy that splits punctuation marks or spaces, except that it keeps intact words like "I've" or "wasn't", that have a punctuation mark in the middle, in between two letters. (Or when the punctuation mark is at the beginning, as in "'em", or when there is a dollar sign at the beginning.) Apply your strategy to `trump.words` as defined below such that you display only those words with punctuation marks and/or dollar signs. (Note that I end up with 102 [not necessarily unique, but total] words when I implement this strategy. Some include'"', which we can easily remove in a subsequent post-processing step if we so choose. Note also that a dollar sign is *not* a punctuation mark; this will affect how you define your regex. Hint: `[[:alnum:]]` is a good thing to use here.)

```
punct_pattern = "([[:alnum:]]+[[:punct:]]{1}[[:alnum:]]+)|(\\$[[:alnum:]]+)|([[:punct:]]{1}[[:alnum:]]+

trump <- gregexpr(punct_pattern,trump.lines)
unlist(regmatches(trump.lines,trump))
```

```
##   [1] "would’ve"          "would’ve"          "would’ve"
##   [4] "carefully-crafted" "Administration’s"  "America’s"
##   [7] "That’s"            "nation’s"          "President’s"
##  [10] "2,000"             "4,000"             "180,000"
##  [13] "border-crosser"    "years-old"         "4.0"
##  [16] "I’ve"              "Sarah’s"           "wasn’t"
##  [19] "African-American"  "African-American"  "$4"
##  [22] ",000"              "that’s"            "$800"
##  [25] "$800"              "We’re"             "$19"
##  [28] "forty-three"       "$150"              "America’s"
##  [31] "Let’s"             "Let’s"             "pre-Hillary"
##  [34] "Clinton’s"         "America’s"         "nation’s"
##  [37] "it’s"              "Clinton’s"         "laid-off"
##  [40] "they’re"           "won’t"             "33,000"
##  [43] "can’t"             "\"extremely"       "\"negligent"
##  [46] "America’s"         "America’s"         "It’s"
##  [49] "It’s"              "It’s"              "It’s"
##  [52] "we’re"             "don’t"             "there’s"
##  [55] "African-American"  "it’s"              "America’s"
##  [58] "catch-and"         "-release"          "won’t"
##  [61] "It’s"              "I’m"               "nearly-one"
##  [64] "China’s"           "husband’s"         "it’s"
##  [67] "China’s"           "we’ll"             "don’t"
##  [70] "Middle-income"     "highest-taxed"     "job-killers"
##  [73] "$2"                "that’s"            "she’s"
##  [76] "that’s"            "she’s"             "We’re"
##  [79] "ten-point"         "I’m"               "I’ll"
##  [82] "they’ve"           "I’m"               "I’m"
##  [85] "he’d"              "It’s"              "there’s"
##  [88] "’em"               "It’s"              "don’t"
##  [91] "can’t"             "wouldn’t"          "doesn’t"
##  [94] "don’t"             "Don’t"             "don’t"
##  [97] "It’s"              "three-word"        "\"I"
## [100] "’m"                "\"I"               "’m"
```

---

Below, we read in lines of data from the Advanced National Seismic System (ANSS), on earthquakes of magnitude 6+, between 2002 and 2017. (You don’t have to do anything yet.)

```
anss.lines = readLines("http://www.stat.cmu.edu/~pfreeman/anss.htm")
date.pattern = "[0-9]{4}/[0-9]{2}/[0-9]{2}"
date.lines = grep(date.pattern,anss.lines,value=TRUE)
```

---

## Question 5

*(10 points)*

Check that all the lines in `date.lines` actually start with a date, of the form YYYY/MM/DD. rather than contain a date of this form somewhere in the middle of the text. (Hint: it might help to note that you can look for non-matches, as opposed to matches, by changing one of `grep()`'s logical arguments.)

```
date.pattern1 = "^[0-9]{4}/[0-9]{2}/[0-9]{2}" #check beginning
grep(date.pattern1,date.lines,value=TRUE,invert = TRUE) #instead of look for when true, look for when f
```

```
## character(0)
```

## Question 6

*(10 points)*

Which five days witnessed the most earthquakes, and how many were there, these days? Also, what happened on the day with the most earthquakes: can you find any references to this day in the news?

```
most <- gregexpr(date.pattern,date.lines)
res <- unlist(regmatches(date.lines,most))
sort(table(res), decreasing = TRUE)[1:5]
```

```
## res
## 2011/03/11 2010/02/27 2004/12/26 2006/11/15 2013/02/06
##         42         12         11          7          7
```

```
On, 2011/03/11 the day with the most earthquakes, the 2011 Tōhoku earthquake and tsunami
occurred in Japan. The earthquake was the most powerful earthquake
ever recorded in Japan, and the 4th most powerful in the world.
```

## Question 7

*(10 points)*

Go back to the data in `date.lines`. Following steps similar to the ones you used in the lab to extract the latitude and longitude of earthquakes, extract the depth and magnitude of earthquakes. In the end, you should have one numeric vector of depths, and one numeric vector of magnitudes. Show the first three depths and the first three magnitudes. (Hint: if you use `regexpr()` and `regmatches()`, then the output from the latter will be a vector of strings. Look at this vector. The last four characters always represent the magnitudes. Use a combination of `substr()` and `as.numeric()` to create the numeric vector of magnitudes. Then use the fact that everything but the last four characters represents the depths. There are a myriad of ways to do this exercise, but this suggested way is the most concise.)

```
depth_mag_pattern = "[0-9]{3}\\.[0-9]{2} {2}[0-9]{1}\\.[0-9]{2}"
dept_mag <- regexpr(depth_mag_pattern,date.lines)
res <- regmatches(date.lines,dept_mag)
depth = as.numeric(substr(res,1,6)) #numeric vector of depths
mag = as.numeric(substr(res,8,12)) #numeric vector of magnitudes
```

Here we read in text containing the fastest men's 100-meter sprint times. We retain only the lines that correspond to the sprint data, for times 9.99 seconds or better.

```
sprint.lines = readLines("http://www.stat.cmu.edu/~pfreeman/men_100m.html")
data.lines = grep(" +(9|10)\\.",sprint.lines)
sprint.lines = sprint.lines[min(data.lines):max(data.lines)]
```

---

## Question 8

*(10 points)*

Extract the years in which the sprint times were recorded. Display them in table format. Do the same for the months. Be sure to extract the month of the sprint, not the birth month of the sprinter! (Hint: the month of the sprint is followed by a four-digit year; other instances of two digits in any given line are not. So you may have to extract more than you need, then apply `strsplit()`.)

```
split_pattern = "[0-9]{2}\\.[0-9]{2}\\.[0-9]{4}"
reg.expr = regexpr(split_pattern,sprint.lines)
month_year = regmatches(sprint.lines,reg.expr)
res <- strsplit(unlist(month_year),split = "\\.")
res<- t(matrix(unlist(res),nrow = 3))
sort(table(res[,2]),decreasing = TRUE)
```

```
##
##  08  06  07  09  05  04  03  02  10
## 277 244 223 104  99  28   7   3   3
```

```
sort(table(res[,3]),decreasing = TRUE)
```

```
##
## 2015 2012 2016 2013 2011 2018 2008 2019 2017 2009 1997 2010 1998 2004 2014
1999
## 91 61 60 59 57 57 53 50 48 45 38 36 33 33 32 26
## 1996 2006 2007 1994 2001 2002 2005 2000 1991 1988 2003 1993 1992 1995 1984
1983
## 24 24 18 16 16 16 16 14 13 9 9 8 6 5 4 3
## 1989 1990 1968 1977 1985 1987
## 2 2 1 1 1 1
```

## Question 9

*(10 points)*

Extract the countries of origin (for the sprinters). Note that countries of origin are given as a capitalized three-letter abbreviation. Display the table of country origins. Display the proportion of the list that is accounted for by sprinters from the US and Jamaica.

```
split_pattern = "[A-Z]{3}"
reg.expr = regexpr(split_pattern,sprint.lines)
country = regmatches(sprint.lines,reg.expr)
res <- sort(table(country),decreasing = TRUE)
res
```

```
## country
## USA JAM TTO CAN GBR FRA RSA NAM NGR CIV SKN QAT ANT CHN AHO BAR GHA JPN POR
BAH
## 406 267 51 37 35 30 29 27 25 11 10 8 7 7 5 4 4 4 4 3
## TUR ZIM CUB AUS CAY ITA NED NOR OMA
## 3 3 2 1 1 1 1 1 1
```

```
US_J <- res['USA']+res['JAM']
total <- sum(res)
proportion = US_J / total
cat("\nProprtion of runners from Jamaica and United States: ",proportion)
```

```
##
## Proprtion of runners from Jamaica and United States:  0.6811741
```

## Question 10

*(10 points)*

We conclude with a web scraping exercise. I want you to go to this web site. On it, you see there is a set of 12 bold-faced four-digit numbers: this is the number of submitted astrophysics articles for each month of 2019. I want you to extract these numbers and place them into a single vector, with each vector element having a name: Jan for the first vector element, Feb for the second, etc. You would use `readLines()` to read in the page (pass the URL directly to the function!); this creates a vector of strings. You would then use `regexpr()` and `regmatches()` to extract the numbers (plus potentially some other stuff that you may have to pare off using `substr()`). If necessary, use "view source" to look at the html code for the web page itself to determine how best to extract the 12 numbers and nothing else. You don't want to create a table; you simply want to output the vector of four-digit numbers and add the appropriate names. (Hint: see the documentation for `Constants`. `month.abb` might be helpful here.)

```
astro.lines = readLines("https://arxiv.org/year/astro-ph/19")
astro_pattern = "<[a-z]{1}>[0-9]{4}</[a-z]{1}> {1}\\+"
astro = regexpr(astro_pattern,astro.lines)
num_articles = regmatches(astro.lines,astro)
num_articles = as.numeric(substr(num_articles,4,7))
name.pattern = "\\([a-zA-Z]{3} {1}[0-9]{4}"
name = regexpr(name.pattern,astro.lines)
months = substr(regmatches(astro.lines,name),2,4)
names(num_articles) <- months
num_articles
```

```
##  Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
## 1125 1038 1475 1258 1159 1023 1277 1225 1344 1295 1100 1100
```