Basic String Manipulation

36-600

What is a String?

- A string is a sequence of characters that are bound together, where a character is a symbol is a written language
    - in `R`, a string is of class `character` and is bounded by quotes (either single or double)…double quotes are preferable, because then one can use single quotes as apostrophes in strings
- In this set of notes, we will focus on common functions that one might use to analyze string-based data

Data

- Below, we load in the text of Hillary Clinton's acceptance of the Democratic Party nomination in 2016
  - we use the `readLines()` function, which reads in data from an ASCII text file line-by-line

```r
lines <- readLines("http://www.stat.cmu.edu/~pfreeman/clinton.txt")
lines[1]
```

```
## [1] "Thank you! Thank you all very much! Thank you for that amazing welcome. "
```

Basic String Functions

- To concatenate strings, use the `paste()` function

```r
paste(lines[1],lines[3])
```

```
## [1] "Thank you! Thank you all very much! Thank you for that amazing welcome.  Thank you all for the great convention
```

- Note that by default, the strings are concatenated with a space separating them; this is controlled by the `sep` argument

```r
paste(lines[1],lines[3],sep=" 1 2 3 ")
```

```
## [1] "Thank you! Thank you all very much! Thank you for that amazing welcome.  1 2 3 Thank you all for the great conve
```

Basic String Functions

- Note that above, we pasted two separate arguments together; we can also pass in a vector of strings, in which case the separation is controlled by the `collapse` argument

```
paste(lines[1:3],collapse=" ")
```

```
## [1] "Thank you! Thank you all very much! Thank you for that amazing welcome.   Thank you all for the great conventio
```

- `sep` and `collapse` can be combined

```
paste(c("a","b"),c("1","2"),sep="+") ; paste(c("a","b"),c("1","2"),sep="+",collapse=",")
```

```
## [1] "a+1" "b+2"
```

```
## [1] "a+1,b+2"
```

Basic String Functions

- To count the number of characters in a string, use the function `nchar()` (as opposed to `length()`, which counts the number of elements in a vector)

```
length(lines)         # number of lines input via readLines()
```

```
## [1] 301
```

```
nchar(lines)[1:3]    # number of characters in each of the first three lines
```

```
## [1] 72  0 54
```

- If you, for instance, want to extract the first and last ten characters in one of the lines, you would use `substr()`

```
substr(lines[53],1,10)                                 # characters 1 through 10 of line 53
```

```
## [1] "Too many p"
```

```
substr(lines[53],nchar(lines[53])-9,nchar(lines[53])) # ...and the last ten characters
```

```
## [1] "the crash."
```

String Splitting

- Suppose you want to split a string on a particular character
  - the most common example is splitting on spaces, to get all the words in a string

```
strsplit(lines[c(1,3)],split=" ")  # split the first and third lines on spaces
```

```
## [[1]]
##  [1] "Thank"    "you!"     "Thank"    "you"      "all"      "very"
##  [7] "much!"    "Thank"    "you"      "for"      "that"     "amazing"
## [13] "welcome."
##
## [[2]]
##  [1] "Thank"       "you"         "all"         "for"         "the"
##  [6] "great"       "convention"  "that"        "we've"       "had."
```

- We immediately note the following:
  - `strsplit()` returns a list, with each element of the list mapping back to each element of the string vector that was input (here, we input two lines, and got back a list with two elements)
  - each list element contains a vector of split-up characters
  - not every output string is actually a word

String Splitting and Regular Expressions

- *Regular expressions*, or *regexes*, are specially constructed strings that allow for flexible pattern matching
    - the rules for constructing regexes are independent of `R`; you may already know them
- We will focus on the use of square brackets and metacharacters to define a regex
- Square brackets: we want to match any one character that appears inside
    - "[abcde]" means "look for any string that contains a, b, c, d, or e" (case sensitive!)
    - "[a-e]" means the same thing; the dash denotes a range
    - "[^a-e]" means "look for any string that contain characters other than a, b, c, d, or e"
    - " [1-4][2-6] " matches strings that contain the numbers 12-16, 22-26, 32-36, or 42-46

String Splitting and Regular Expressions

- Let's split the first two lines from the speech on spaces *and* exclamation points

```
strsplit(lines[c(1,3)],split="[ !]") # not space, then exclamation point, but space or exclamation point
```

```
## [[1]]
##  [1] "Thank"    "you"      ""         "Thank"    "you"      "all"
##  [7] "very"     "much"     ""         "Thank"    "you"      "for"
## [13] "that"     "amazing"  "welcome."
##
## [[2]]
##  [1] "Thank"      "you"        "all"        "for"        "the"
##  [6] "great"      "convention" "that"       "we’ve"      "had."
```

String Splitting and Regular Expressions

- Commonly used metacharacters include
  - "[[:alnum:]]" is the same as "[a-zA-Z0-9]"
  - "[[:punct:]]" means "match any string that contains a punctuation mark"
  - "[[:space:]]" means "match any string that contains a space, a tab, or a new line"

```
strsplit(lines[c(1,3)],split="( |[[:punct:]])") # split on space or a punctuation mark
```

```
## [[1]]
##  [1] "Thank"   "you"     ""        "Thank"   "you"     "all"     "very"   
##  [8] "much"    ""        "Thank"   "you"     "for"     "that"    "amazing"
## [15] "welcome" ""       
## 
## [[2]]
##  [1] "Thank"      "you"        "all"        "for"        "the"       
##  [6] "great"      "convention" "that"       "we"         "ve"        
## [11] "had"
```

- But now we've split on the apostrophe (and we have empty strings)
  - we'll stop here because we've (basically) made the point of how to turn text input into words...

String Splitting and Regular Expressions

- ...well, except for one last thing
- In R, the following are special characters

```
. $ ^ * + ? \ | { } [ ] ( ) \
```

- To find occurrences of these symbols in strings, we use an *escape sequence*: we place a backslash in front of the symbol
  - but given that the backslash is a special character, it itself needs to be escaped

```
strsplit(lines[c(1,3)],split="[ !\\.]") # split on spaces, exclamation points, and escaped periods
```

```
## [[1]]
##  [1] "Thank"   "you"     ""        "Thank"   "you"     "all"     "very"
##  [8] "much"    ""        "Thank"   "you"     "for"     "that"    "amazing"
## [15] "welcome" ""
##
## [[2]]
##  [1] "Thank"      "you"        "all"        "for"        "the"
##  [6] "great"      "convention" "that"       "we've"      "had"
```

Word Tables

- What are the 20 most common words in Clinton's speech?

```
sort(table(unlist(strsplit(lines,split="[ !\\.]"))),decreasing=TRUE)[1:20]
```

```
##
##        the   to  and    a   of   in  you  our   we  And that    I  for    -   is
##  253  199  171  157   99   97   72   70   69   69   59   58   56   53   43   42
##  are   it will with
##   36   36   36   36
```

- We note a few more things here:
  - `unlist()` is a way to concatenate all the elements of a list together into a single vector
  - we need to do some additional processing to remove the empty strings
  - case sensitivity impacts the count (e.g., "and" and "And" were treated separately)...this can be mitigated by applying the `tolower()` function after `unlist()`, which converts all letters to lower case

String Searching

- If you want to see if a particular word appears in a line, you can use the `grep` family of functions
  - e.g., if you want to determine if "And" occurs on a line, use `grepl()`, which turns `TRUE` or `FALSE`

```
grepl("And",lines[1:5])
```

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

- String searches allow for regexes

```
grepl("(and|And)",lines[1:5])
```

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

String Searching

- `grep()` itself either returns the number of the line in which the string is observed (here, line 5)

```
grep("and",lines[1:5])
```

```
## [1] 5
```

- If you pass in the argument `value=TRUE`, you get the lines themselves

```
grep("and",lines[1:5],value=TRUE)
```

```
## [1] "And Chelsea, thank you. I'm so proud to be your mother and so proud of the woman you've become. Thank you for b
```

Dynamic String Extraction

- We saw above that we can use `substr()` to extract a substring
  - however, we need to specify where the substring starts and where it ends
- A more dynamic extractor involves combining `gregexpr()` and `regmatches()`
  - let's extract every occurrence of "and" and "And" in the first five lines

```
out     <- gregexpr("(a|A)nd",lines[1:5])
matches <- regmatches(lines[1:5],out)     # outputs a list, one for each line
unlist(matches)
```

```
## [1] "And" "and" "and" "and"
```

Removing/Replacing Characters

- Let's say that instead of splitting on punctuation, as we (eventually) did above, we want to remove or replace them
  - one way to do that is to use `gsub()`

```r
gsub("[[:punct:]]","-",lines[1])
```

```
## [1] "Thank you- Thank you all very much- Thank you for that amazing welcome- "
```

Stopwords

- One last thing to look at is the removal of uninformative *stopwords* from a document
- It is often the case that we'd want to remove words like "a", "and", and "the," etc., before doing any real statistical analysis

```
library(stopwords)
head(stopwords("en"),10)
```

```
##  [1] "i"         "me"        "my"        "myself"    "we"        "our"
##  [7] "ours"      "ourselves" "you"       "your"
```

- Here's what Clinton's speech looks like after simplistic processing and stopword removal

```
speech          <- tolower(unlist(strsplit(lines,split="[ ,!\\.]")))
w               <- which(nchar(speech)==0)
speech          <- speech[-w]  # could do speech <- speech[speech!=""] also, or dplyr...
stopword.logical <- speech %in% stopwords("en")      # is element of left "in" vector at right? [T/F]
paste(speech[stopword.logical==FALSE],collapse=" ")
```

```
## [1] "thank thank much thank amazing welcome thank great convention we’ve chelsea thank proud mother proud woman becon
```