Text Manipulation

Statistical Computing

Last class: Indexing and iteration

- Three ways to index vectors, matrices, data frames, lists: integers, Booleans, names
- Boolean on-the-fly indexing can be very useful
- Named indexing will be especially useful for data frames
- Indexing lists can be a bit tricky (beware of the difference between `[ ]` and `[[ ]]`)
- `if()`, `elseif()`, `else`: standard conditionals
- `ifelse()`: shortcut for using `if()` and `else` in combination
- `switch()`: shortcut for using `if()`, `elseif()`, and `else` in combination
- `for()`, `while()`: standard loop constructs
- Don't overuse explicit `for()` loops, vectorization is your friend!
- `apply()` and `**ply()`: can also be very useful (we'll see them later)

Part I

String basics

What are strings?

The simplest distinction:

- **Character:** a symbol in a written language, like letters, numerals, punctuation, space, etc.
- **String:** a sequence of characters bound together

```
class("r")
```

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

```
class("New Mexico")
```

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

Why do we care about strings?

- A lot of interesting data out there is in text format!
- Webpages, emails, surveys, logs, search queries, etc.
- Even if you just care about numbers eventually, you'll need to understand how to get numbers from text

Whitespaces

Whitespaces count as characters and can be included in strings:

- `" "` for space
- `"\n"` for newline
- `"\t"` for tab

```
str = "Dear friend,\n\n Hope you are doing well!\n\nSincerely, Li"
str
```

```
## [1] "Dear friend,\n\n Hope you are doing well!\n\nSincerely, Li"
```

Use `cat()` to print strings to the console, displaying whitespaces properly

```
cat(str)
```

```
## Dear friend,
##
##  Hope you are doing well!
##
## Sincerely, Li
```

Vectors/matrices of strings

The character is a basic data type in R (like numeric, or logical), so we can make vectors or matrices of out them. Just like we would with numbers

```
str.vec = c("Statistical", "Computing", "isn't that bad") # Collect 3 strings
str.vec # All elements of the vector
```

```
## [1] "Statistical"    "Computing"      "isn't that bad"
```

```
str.vec[3] # The 3rd element
```

```
## [1] "isn't that bad"
```

```
str.vec[-(1:2)] # All but the 1st and 2nd
```

```
## [1] "isn't that bad"
```

```
str.mat = matrix("", 2, 3) # Build an empty 2 x 3 matrix
str.mat[1,] = str.vec # Fill the 1st row with str.vec
str.mat[2,1:2] = str.vec[1:2] # Fill the 2nd row, only entries 1 and 2, with
                              # those of str.vec
str.mat[2,3] = "isn't a fad" # Fill the 2nd row, 3rd entry, with a new string
str.mat # All elements of the matrix
```

```
##      [,1]          [,2]        [,3]
## [1,] "Statistical" "Computing" "isn't that bad"
## [2,] "Statistical" "Computing" "isn't a fad"
```

```
t(str.mat) # Transpose of the matrix
```

```
##      [,1]             [,2]
## [1,] "Statistical"    "Statistical"
## [2,] "Computing"      "Computing"
## [3,] "isn't that bad" "isn't a fad"
```

Converting other data types to strings

Easy! Make things into strings with `as.character()`

```r
as.character(0.8)
```

```
## [1] "0.8"
```

```r
as.character(0.8e+10)
```

```
## [1] "8e+09"
```

```r
as.character(1:5)
```

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

```r
as.character(TRUE)
```

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

Converting strings to other data types

Not as easy! Depends on the given string, of course

```r
as.numeric("0.5")
```

```
## [1] 0.5
```

```r
as.numeric("0.5 ")
```

```
## [1] 0.5
```

```r
as.numeric("0.5e-10")
```

```
## [1] 5e-11
```

```r
as.numeric("Hi!")
```

```
## Warning: NAs introduced by coercion
## [1] NA
```

```r
as.logical("True")
```

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

```r
as.logical("TRU")
```

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

Number of characters

Use `nchar()` to count the number of characters in a string

```r
nchar("coffee")
```

```
## [1] 6
```

```r
nchar("code monkey")
```

```
## [1] 11
```

```r
length("code monkey")
```

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

```
length(c("coffee", "code monkey"))
```

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

```
nchar(c("coffee", "code monkey")) # Vectorization!
```

```
## [1]  6 11
```

Part II

Substrings, splitting and combining strings

Getting a substring

Use `substr()` to grab a subseqence of characters from a string, called a **substring**

```
phrase = "Give me a try"
substr(phrase, 1, 4)
```

```
## [1] "Give"
```

```
substr(phrase, nchar(phrase)-4, nchar(phrase))
```

```
## [1] "a try"
```

```
substr(phrase, nchar(phrase)+1, nchar(phrase)+10)
```

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

`substr()` vectorizes

Just like `nchar()`, and many other string functions

```
presidents = c("Clinton", "Bush", "Reagan", "Carter", "Ford")
substr(presidents, 1, 2) # Grab the first 2 letters from each
```

```
## [1] "Cl" "Bu" "Re" "Ca" "Fo"
```

```
substr(presidents, 1:5, 1:5) # Grab the first, 2nd, 3rd, etc.
```

```
## [1] "C" "u" "a" "t" ""
```

```
substr(presidents, 1, 1:5) # Grab the first, first 2, first 3, etc.
```

```
## [1] "C"    "Bu"   "Rea"  "Cart" "Ford"
```

```
substr(presidents, nchar(presidents)-1, nchar(presidents)) # Grab the last 2
```

```
## [1] "on" "sh" "an" "er" "rd"
```

```
                                                           # letters from each
```

Replace a substring

Can also use `substr()` to replace a character, or a substring

```
phrase
```

```
## [1] "Give me a try"
```

```
substr(phrase, 1, 1) = "L"
phrase # "G" changed to "L"
```

```
## [1] "Live me a try"
```

```
substr(phrase, 1000, 1001) = "R"
phrase # Nothing happened
```

```
## [1] "Live me a try"
```

```
substr(phrase, 1, 4) = "Show"
phrase # "Live" changed to "Show"
```

```
## [1] "Show me a try"
```

Splitting a string

Use the `strsplit()` function to split based on a keyword

```
ingredients = "chickpeas, tahini, olive oil, garlic, salt"
split.obj = strsplit(ingredients, split=",")
split.obj
```

```
## [[1]]
## [1] "chickpeas"  " tahini"    " olive oil" " garlic"    " salt"
```

```
class(split.obj)
```

```
## [1] "list"
```

```
length(split.obj)
```

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

Note that the output is actually a list! (With just one element, which is a vector of strings)

`strsplit()` vectorizes

Just like `nchar()`, `substr()`, and the many others

```
great.profs = "Nugent, Genovese, Greenhouse, Seltman, Shalizi, Ventura"
favorite.cats = "tiger, leopard, jaguar, lion"
split.list = strsplit(c(ingredients, great.profs, favorite.cats), split=",")
split.list
```

```
## [[1]]
## [1] "chickpeas"  " tahini"    " olive oil" " garlic"    " salt"
##
## [[2]]
```

```
## [1] "Nugent"      " Genovese"   " Greenhouse" " Seltman"    " Shalizi"
## [6] " Ventura"
##
## [[3]]
## [1] "tiger"    " leopard" " jaguar"  " lion"
```

- Returned object is a list with 3 elements
- Each one a vector of strings, having lengths 5, 6, and 4
- Do you see why `strsplit()` needs to return a list now?

Splitting character-by-character

Finest splitting you can do is character-by-character: use `strsplit()` with `split=""`

```
split.chars = strsplit(ingredients, split="")[[1]]
split.chars
```

```
##  [1] "c" "h" "i" "c" "k" "p" "e" "a" "s" "," " " "t" "a" "h" "i" "n" "i"
## [18] "," " " "o" "l" "i" "v" "e" " " "o" "i" "l" "," " " "g" "a" "r" "l"
## [35] "i" "c" "," " " "s" "a" "l" "t"
```

```
length(split.chars)
```

```
## [1] 42
```

```
nchar(ingredients) # Matches the previous count
```

```
## [1] 42
```

Combining strings

Use the `paste()` function to join two (or more) strings into one, separated by a keyword

```
paste("Spider", "Man") # Default is to separate by " "
```

```
## [1] "Spider Man"
```

```
paste("Spider", "Man", sep="-")
```

```
## [1] "Spider-Man"
```

```
paste("Spider", "Man", "does whatever", sep=", ")
```

```
## [1] "Spider, Man, does whatever"
```

`paste()` vectorizes

Just like `nchar()`, `substr()`, `strsplit()`, etc. Seeing a theme yet?

```
presidents
```

```
## [1] "Clinton" "Bush"    "Reagan"  "Carter"  "Ford"
```

```
paste(presidents, c("D", "R", "R", "D", "R"))
```

```
## [1] "Clinton D" "Bush R"    "Reagan R"  "Carter D"  "Ford R"
```

```
paste(presidents, c("D", "R")) # Notice the recycling (not historically accurate!)
```

```
## [1] "Clinton D" "Bush R"    "Reagan D"  "Carter R"  "Ford D"
```

```
paste(presidents, " (", 42:38, ")", sep="")
```

```
## [1] "Clinton (42)" "Bush (41)"    "Reagan (40)"  "Carter (39)"
## [5] "Ford (38)"
```

Condensing a vector of strings

Can condense a vector of strings into one big string by using `paste()` with the `collapse` argument

```
presidents
```

```
## [1] "Clinton" "Bush"    "Reagan"  "Carter"  "Ford"
```

```
paste(presidents, collapse="; ")
```

```
## [1] "Clinton; Bush; Reagan; Carter; Ford"
```

```
paste(presidents, " (", 42:38, ")", sep="", collapse="; ")
```

```
## [1] "Clinton (42); Bush (41); Reagan (40); Carter (39); Ford (38)"
```

```
paste(presidents, " (", c("D", "R", "R", "D", "R"), 42:38, ")", sep="", collapse="; ")
```

```
## [1] "Clinton (D42); Bush (R41); Reagan (R40); Carter (D39); Ford (R38)"
```

```
paste(presidents, collapse=NULL) # No condensing, the default
```

```
## [1] "Clinton" "Bush"    "Reagan"  "Carter"  "Ford"
```

Part III

Reading in text, summarizing text

Text from the outside

How to get text, from an external source, into R? Use the `readLines()` function

```
trump.lines = readLines("http://www.stat.cmu.edu/~ryantibs/statcomp/data/trump.txt")
class(trump.lines) # We have a character vector
```

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

```
length(trump.lines) # Many lines (elements)!
```

```
## [1] 113
```

```
trump.lines[1:3] # First 3 lines
```

```
## [1] "Friends, delegates and fellow Americans: I humbly and gratefully accept your nomination for the
## [2] "Story Continued Below"
## [3] ""
```

(This was Trump's acceptance speech at the 2016 Republican National Convention)

Reading from a local file

We don't need to use the web; `readLines()` can be used on a local file. The following code would read in a text file from Professor Tibs' computer:

```
trump.lines.2 = readLines("~/Dropbox/xx/trump.txt")
```

```
## Warning in file(con, "r"): cannot open file '/Users/lili/Dropbox/xx/
## trump.txt': No such file or directory
```

```
## Error in file(con, "r"): cannot open the connection
```

This will cause an error for you, unless your folder is set up exactly like Professor Tibs' laptop! So using web links is more robust

Reconstitution

Fancy word, but all it means: make one long string, then split the words

```
trump.text = paste(trump.lines, collapse=" ")
trump.words = strsplit(trump.text, split=" ")[[1]]

# Sanity check
substr(trump.text, 1, 200)
```

```
## [1] "Friends, delegates and fellow Americans: I humbly and gratefully accept your nomination for the
```

```
trump.words[1:20]
```

```
##  [1] "Friends,"   "delegates"  "and"        "fellow"     "Americans:"
##  [6] "I"          "humbly"     "and"        "gratefully" "accept"
## [11] "your"       "nomination" "for"        "the"        "presidency"
## [16] "of"         "the"        "United"     "States."    "Story"
```

Counting words

Our most basic tool for summarizing text: **word counts**, retrieved using `table()`

```
trump.wordtab = table(trump.words)
class(trump.wordtab)
```

```
## [1] "table"
```

```
length(trump.wordtab)
```

```
## [1] 1604
```

```
trump.wordtab[1:10]
```

```
## trump.words
##                          -          'I   "extremely         "I'm
##            1           34           1            1            1
##         "I'M "negligent,"        $150          $19           $2
```

```
##             1             1             1             1             1
```

What did we get? Alphabetically sorted unique words, and their counts = number of appearances

The names are words, the entries are counts

Note: this is actually a vector of numbers, and the words are the names of the vector

```
trump.wordtab[1:5]
```

```
## trump.words
##                       -          'I "extremely       "I'm
##           1          34           1           1           1
```

```
trump.wordtab[2] == 34
```

```
##    -
## TRUE
```

```
names(trump.wordtab)[2] == "-"
```

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

So with named indexing, we can now use this to look up whatever words we want

```
trump.wordtab["America"]
```

```
## America
##      19
```

```
trump.wordtab["great"]
```

```
## great
##     7
```

```
trump.wordtab["wall"]
```

```
## wall
##    1
```

```
trump.wordtab["Canada"] # NA means Trump never mentioned Canada
```

```
## <NA>
##   NA
```

Most frequent words

Let's sort in decreasing order, to get the most frequent words

```
trump.wordtab.sorted = sort(trump.wordtab, decreasing=TRUE)
length(trump.wordtab.sorted)
```

```
## [1] 1604
```

```
head(trump.wordtab.sorted, 20) # First 20
```

```
## trump.words
##   the   and    of    to   our  will    in     I  have     a  that   for
```

```
##   189   145   127   126    90    82    69    64    57    51    48    46
##    is   are    we     - their    be    on   was
##    40    39    35    34    28    26    26    26
```

```
tail(trump.wordtab.sorted, 20) # Last 20
```

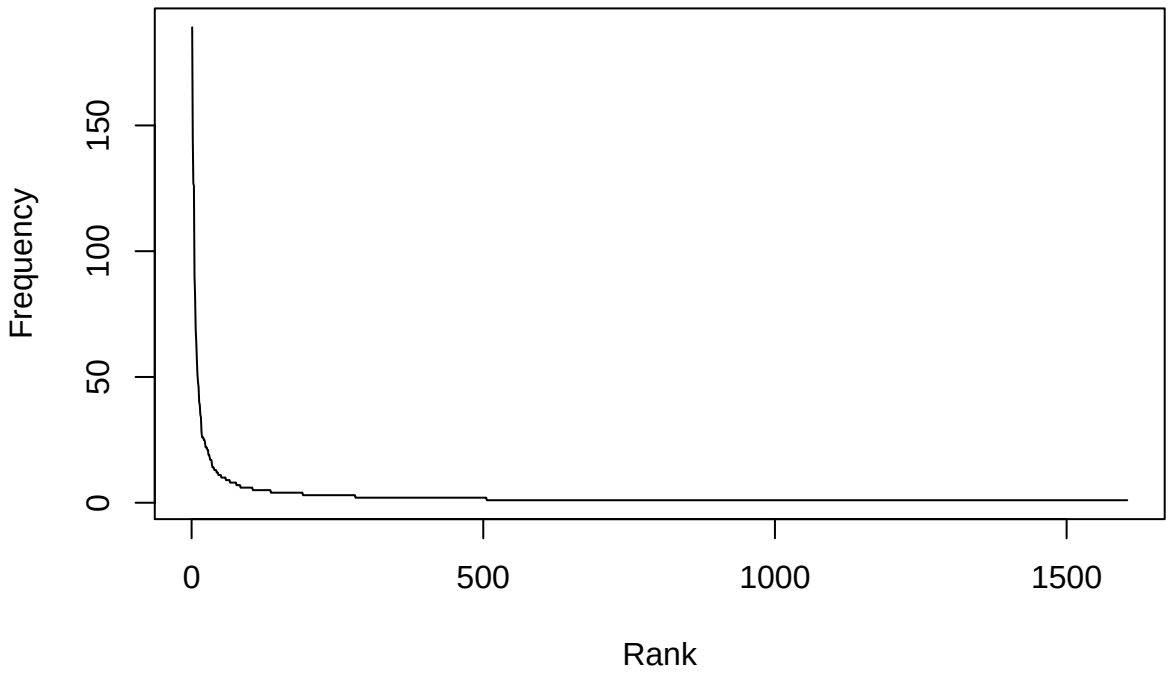
```
## trump.words
##     wonder    workers  workforce     works,      worth   wouldn't
##          1          1          1          1          1          1
##    wounded years-old,     years.        yet        Yet       Yet,
##          1          1          1          1          1          1
##        YOU       you,       You,       you:       YOU.   youngest
##          1          1          1          1          1          1
##       YOUR      youth
##          1          1
```

Notice that punctuation matters, e.g., "Yet" and "Yet," are treated as separate words, not ideal—we'll learn just a little bit about how to fix this on lab/homework, using **regular expressions**

Visualizing frequencies

Let's use a plot to visualize frequencies

```
nw = length(trump.wordtab.sorted)
plot(1:nw, as.numeric(trump.wordtab.sorted), type="l",
     xlab="Rank", ylab="Frequency")
```



A pretty drastic looking trend! It looks as if Frequency $\propto (1/\text{Rank})^a$ for some $a > 0$

Zipf's law

This phenomenon, that frequency tends to be inversely proportional to a power of rank, is called **Zipf's law**

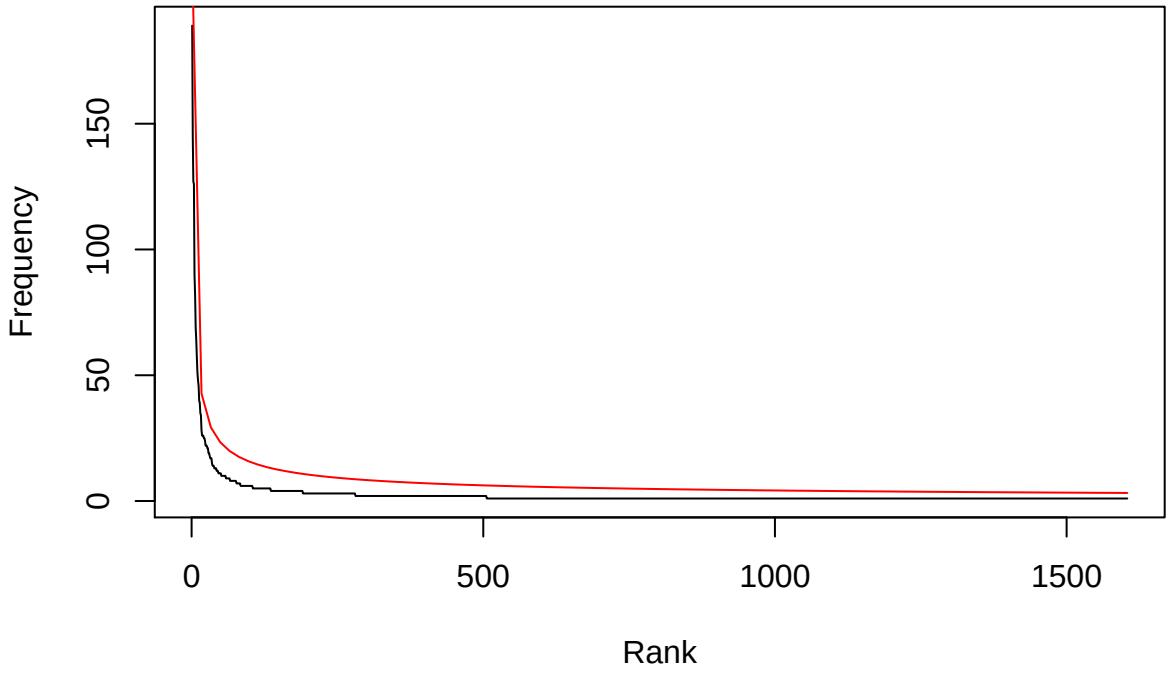
For our data, Zipf's law approximately holds, with Frequency $\approx C(1/\text{Rank})^a$ for $C = 215$ and $a = 0.57$

```
C = 215; a = 0.57
trump.wordtab.zipf = C*(1/1:nw)^a
cbind(trump.wordtab.sorted[1:8], trump.wordtab.zipf[1:8])
```

```
##      [,1]      [,2]
## the   189 215.00000
## and   145 144.82761
## of    127 114.94216
## to    126  97.55831
## our    90  85.90641
## will   82  77.42697
## in     69  70.91410
## I      64  65.71691
```

Not perfect, but not bad. We can also plot the original sorted word counts, and those estimated by our formula law on top

```
plot(1:nw, as.numeric(trump.wordtab.sorted), type="l",
     xlab="Rank", ylab="Frequency")
curve(C*(1/x)^a, from=1, to=nw, col="red", add=TRUE)
```



We'll learn about plotting tools in detail a bit later

Summary

- Strings are, simply put, sequences of characters bound together
- Text data occurs frequently "in the wild", so you should learn how to deal with it!

- `nchar()`, `substr()`: functions for substring extractions and replacements
- `strsplit()`, `paste()`: functions for splitting and combining strings
- Reconstitution: take lines of text, combine into one long string, then split to get the words
- `table()`: function to get word counts, useful way of summarizing text data
- Zipf's law: word frequency tends to be inversely proportional to (a power of) rank