Text Manipulation

Canhong Wen

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("w")
```

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

```
class("Wen")
```

```
## [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 Mr. Zhang,\n\n\t\tThanks for the great school!\n\nSincerely, Canhong"
str
```

```
## [1] "Dear Mr. Zhang,\n\n\t\tThanks for the great school!\n\nSincerely, Canhong"
```

Print with `cat()`

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

```
cat(str)
```

```
## Dear Mr. Zhang,
## 
##      Thanks for the great school!
## 
## Sincerely, Canhong
```

Print with `sprintf()`

`sprintf()` is a a wrapper for the C function *sprintf*

```
i <- 8
sprintf("the square of %d is %d", i, i^2)
```

```
## [1] "the square of 8 is 64"
```

```
sprintf("the square of %d is %f", i, i^2)
```

```
## [1] "the square of 8 is 64.000000"
```

```
sprintf("the square of %d is %g", i, i^2)
```

```
## [1] "the square of 8 is 64"
```

```
cat("the square of", i,"is", i^2)
```

```
## the square of 8 is 64
```

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

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

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

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

`tolower()` and `toupper()`

The functions tolower() and toupper() convert strings to all lower case characters, and all upper case characters.

```
a <- "Applied Statistical Software"
tolower(a)
```

```
## [1] "applied statistical software"
```

```
toupper(a)
```

```
## [1] "APPLIED STATISTICAL SOFTWARE"
```

Converting other data types to strings

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

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

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

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

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

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

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

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

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

Converting strings to other data types

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

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

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

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

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

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

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

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

```
## Warning: 强制改变过程中产生了NA
```

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

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

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

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

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

Number of characters

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

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

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

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

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

```
nchar("code monkey\t")
```

```
## [1] 12
```

Compare `nchar()` with `length()`

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

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

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

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

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

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

```
nchar("R统计软件")
```

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

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 break"
substr(phrase, 1, 4)
```

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

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

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

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

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

中文字符

```
tmp = "我们一起上实用统计软件课程"
substr(tmp, 1, 4)
```

```
## [1] "我们一起"
```

```
substr(tmp, nchar(tmp)-3, nchar(tmp))
```

```
## [1] "软件课程"
```

```
substr(tmp, nchar(tmp)+1, nchar(tmp)+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"
```

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

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

Replace a substring

Can also use `substr()` to replace a character, or a substring by specifying its location

```
phrase
```

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

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

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

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

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

```
substr(phrase, 1, 2) = "D"
phrase # "L" changed to "D"
```

```
## [1] "Dive me a break"
```

```
substr(phrase, 1, 2) = "Da"
phrase # "Li" changed to "Da"
```

```
## [1] "Dave me a break"
```

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

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

Replace a substring with `gsub()`

Use `gsub()` to replace a character, or a substring by search

```
phrase
```

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

```
gsub("Show", "Give", phrase)
```

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

```
gsub("e", "o", phrase)
```

```
## [1] "Show mo a broak"
```

```
gsub("me", "", phrase)
```

```
## [1] "Show  a break"
```

```
gsub("[ae]", "o", phrase)
```

```
## [1] "Show mo o brook"
```

```
gsub("*,", "", c("yet,", "Yet", "yet"))
```

```
## [1] "yet" "Yet" "yet"
```

```
salary <- c("22万", "30万", "50万", "120万", "11万")
salary0 <- gsub("万","0000", salary)
mean(as.numeric(salary0))
```

```
## [1] 466000
```

```
## alternative approach
salary1 <- gsub("万", "", salary)
mean(as.numeric(salary1))
```

```
## [1] 46.6
```

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"
```

```
split.obj = strsplit(ingredients, split=", ")
split.obj
```

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

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

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

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

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

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

```
unlist(split.obj)
```

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

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

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

`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?

```
split.list[[1]]
```

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

```
split.list[[2]]
```

```
## [1] "Nugent"     "Genovese"   "Greenhouse" "Seltman"    "Shalizi"   
## [6] "Ventura"
```

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
```

Splitting with regular expression

```
strsplit("For really tough problems, you need R.", split = " ")
```

```
## [[1]]
## [1] "For"       "really"    "tough"     "problems," "you"       "need"     
## [7] "R."
```

```
strsplit("For really tough problems, you need R.", split = "[[:space:]]|[[:punct:]]")
```

```
## [[1]]
## [1] "For"      "really"   "tough"    "problems" ""         "you"
## [7] "need"     "R"
```

Splitting Chinese word

Use the `jiebaR` package

```
library(jiebaR)
```

```
## Loading required package: jiebaRD
```

```
cutter = worker()
segment("我们一起上统计软件课程!", cutter)
```

```
## [1] "我们" "一起" "上"   "统计" "软件" "课程"
```

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

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

```
paste0("Spider", "Man") # More efficient way
```

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

`paste()` vectorizes

Just like `nchar()` , `substr()` , `strsplit()` , etc.

```
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, collapse=NULL) # No condensing, the default
```

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

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

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

```
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)"
```

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("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 pr
esidency of the United States."
## [2] "Story Continued Below"
## [3] ""
```

```
head(trump.lines)
```

```
## [1] "Friends, delegates and fellow Americans: I humbly and gratefully accept your nomination for the pr
esidency of the United States."
## [2] "Story Continued Below"
## [3] ""
## [4] "Together, we will lead our party back to the White House, and we will lead our country back to saf
ety, prosperity, and peace. We will be a country of generosity and warmth. But we will also be a country o
f law and order."
## [5] "Our Convention occurs at a moment of crisis for our nation. The attacks on our police, and the ter
rorism in our cities, threaten our very way of life. Any politician who does not grasp this danger is not
fit to lead our country."
## [6] "Americans watching this address tonight have seen the recent images of violence in our streets and
the chaos in our communities. Many have witnessed this violence personally, some have even been its victim
s."
```

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]]
substr(trump.text, 1, 60)
```

```
## [1] "Friends, delegates and fellow Americans: I humbly and gratef"
```

```
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[1:15]
```

```
## trump.words
##                         'I            -   "extremely         "I'm
##            1            1           34            1            1
##         "I'M  "negligent,"         $150          $19           $2
##            1            1            1            1            1
##          $20       $4,000         $800           10          100
##            1            1            1            1            1
```

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

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

The names are words, the entries are counts

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

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

```
trump.wordtab[3] == 34
```

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

```
names(trump.wordtab)[3] == "-"
```

```
## [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["China"]
```

```
## China
##     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.

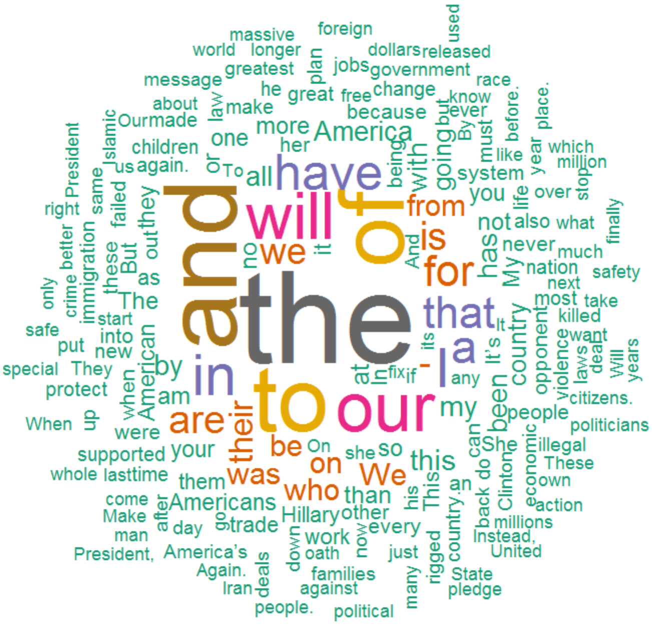
Using **regular expressions** to split the txt file, i.e., `split = "[[:space:]]|[[:punct:]]"` .

```
d <- data.frame(word= names(trump.wordtab.sorted), freq=as.numeric(trump.wordtab.sorted))
library(wordcloud)
```

```
## Warning: package 'wordcloud' was built under R version 3.6.3
```

```
## Loading required package: RColorBrewer
```

```
wordcloud(words = d$word, freq = d$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))
```



Part IV

Dates and Times

Get the current date

```
Sys.Date()
```

```
## [1] "2020-10-20"
```

It returns a `Date` object, and transform to character when output to the Console.

```
class(Sys.Date())
```

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

```
Sys.time()
```

```
## [1] "2020-10-20 16:25:09 CST"
```

Get the current time

```
Sys.time()
```

```
## [1] "2020-10-20 16:25:09 CST"
```

It returns a `POSIXct` object, and transform to character when output to the Console.

```
class(Sys.time())
```

```
## [1] "POSIXct" "POSIXt"
```

`as.Date`

- The `as.Date` function allows a variety of input formats through the `format=` argument.
- The default format is a four digit year, followed by a month, then a day, separated by either dashes or slashes.

```
as.Date("2019-10-23")
```

```
## [1] "2019-10-23"
```

```
as.Date("2019/10/23")
```

```
## [1] "2019-10-23"
```

If your input dates are not in the standard format, a format string can be composed using the elements shown in Table.

Code	Value
`%d`	Day of the month (decimal number)
`%m`	Month (decimal number)
`%b`	Month (abbreviated)

Code	Value
`%B`	Month (full name)
`%y`	Yeat (2 digit)
`%Y`	Yeat (4 digit)

```
as.Date("10/23/2019", format = "%m/%d/%Y")
```

```
## [1] "2019-10-23"
```

```
lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")
```

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

```
as.Date("October 23, 2019", format = "%B %d, %Y")
```

```
## [1] "2019-10-23"
```

```
as.Date("23OCT19", format = "%d%b%y") # %y is system-specific; use with caution
```

```
## [1] "2019-10-23"
```

```
Sys.setlocale("LC_TIME", lct)
```

```
## [1] "Chinese (Simplified)_People's Republic of China.936"
```

Extract the components of the dates

`weekdays`, `months`, `days` or `quarters`

```
bdays <- c(tukey=as.Date('1915-06-16'), fisher=as.Date('1890-02-17'), cramer=as.Date('1893-09-25'), kendall=as.Date('1907-09-06'))
weekdays(bdays)
```

```
##    tukey   fisher   cramer  kendall
## "星期三" "星期一" "星期一" "星期五"
```

```
quarters(bdays)
```

```
## [1] "Q2" "Q1" "Q3" "Q3"
```

```
months(bdays)
```

```
##   tukey  fisher  cramer kendall
##  "六月"  "二月"  "九月"  "九月"
```

`POSIX` format

- Dates stored in the `POSIX` format are date/time values allowing modification of time zones.
- There are two POSIX date/time classes, which differ in the way that the values are stored internally.
    - The `POSIXct` class stores date/time values as the number of seconds since January 1, 1970. It is the usual choice for storing dates in R.
    - The `POSIXlt` class stores date/time values as a list with elements for second, minute, hour, day, month, and year, among others.

Examples:

- 2019/10/23
- 2019-10-23 11:25
- 2019/10/23 12:20:05

```
dts <- c("2019-10-23 18:47:22", "2019-10-23 16:39:58", "2019-10-23 07:30:05 CST")
as.POSIXlt(dts)
```

```
## [1] "2019-10-23 18:47:22 CST" "2019-10-23 16:39:58 CST"
## [3] "2019-10-23 07:30:05 CST"
```

```
as.POSIXct(dts)
```

```
## [1] "2019-10-23 18:47:22 CST" "2019-10-23 16:39:58 CST"
## [3] "2019-10-23 07:30:05 CST"
```

```
as.POSIXct("2019/10/23")
```

```
## [1] "2019-10-23 CST"
```

Extract the components of the dates

```
p <- as.POSIXlt("2019/10/23")
p$mday
```

```
## [1] 23
```

```
p$year + 1900 # The yeas after 1900
```

```
## [1] 2019
```

```
p$wday
```

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

```
p$yday
```

```
## [1] 295
```

Summary on dates and times

Many of the statistical summary functions, like `mean`, `min`, `max`, etc are able to transparently handle date objects.

```
mean(bdays)
```

```
## [1] "1901-10-01"
```

```
range(bdays)
```

```
## [1] "1890-02-17" "1915-06-16"
```

```
bdays[3] - bdays[1]
```

```
## Time difference of -7933 days
```

If an alternative unit of time was desired, the difftime function could be called, using the optional `units=` argument can be used with any of the following values: `auto`, `secs`, `mins`, `hours`, `days`, or `weeks`.

```
difftime(bdays[3], bdays[1], units='weeks')
```

```
## Time difference of -1133.286 weeks
```

Create a date or time sequence

The `by=` argument to the seq function can be specified either as a `difftime` value, or in any units of time that the `difftime` function accepts, making it very easy to generate sequences of dates.

```
seq(as.Date('2019-10-1'),by='days',length=10)
```

```
##  [1] "2019-10-01" "2019-10-02" "2019-10-03" "2019-10-04" "2019-10-05"
##  [6] "2019-10-06" "2019-10-07" "2019-10-08" "2019-10-09" "2019-10-10"
```

```
seq(as.Date('2019-9-4'),to=as.Date('2020-1-1'),by='2 weeks')
```

```
## [1] "2019-09-04" "2019-09-18" "2019-10-02" "2019-10-16" "2019-10-30"
## [6] "2019-11-13" "2019-11-27" "2019-12-11" "2019-12-25"
```

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
- Dates and times in R