Introduction to Strings

Daniel Anderson

Agenda

- Basics of string data
- String operators
    - Cover both *stringr* and *base functions*
- Special characters
- Pattern matching

Before we get started

- In many cases, I've opted to show both base and *stringr* functions
- *stringr* is part of the tidyverse, and has some nice functionality, but many of the base functions are so common I think I'd be doing you a disservice if I didn't also introduce them.
- There are some *stringr* functions that are far easier than the base alternative, so I'll skip base on them (e.g., `str_extract()`).

Some properties of strings

- Strings can be anything wrapped in quotes
- All of the below are strings

```
"TRUE"
"1"
"a"
"4.78"
"purple"
```

- Strings are the most flexible data type. Can be coerced to other types if it makes sense (the below spits out warnings)

```
as.double(c("TRUE", "1", "a", "4.78", "purple"))
```

```
## [1]   NA 1.00   NA 4.78   NA
```

```
as.logical(c("TRUE", "1", "a", "4.78", "purple"))
```

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

Vectors must be of the same type

- This implies that if you have a character element in an atomic vector, all will be coerced to character (because it's the most flexible)

```
c("string", 1.45, TRUE, 5L)
```

```
## [1] "string" "1.45"   "TRUE"   "5"
```

Factors

- Factors may not behave as you'd expect

```
c(factor("a"), "b", 1, 4.59)
```

```
## [1] "1"    "b"    "1"    "4.59"
```

- Why do you think this is happening?
- How could we get this to do what we intend? (i.e., return "a")

Overriding factors

```
c(as.character(factor("a")), "b", 1, 4.59)
```

```
## [1] "a"    "b"    "1"    "4.59"
```

String data for today

Strings to process

```
library(stringr) # loaded with the tidyverse as of version 1.2.0
fruit
sentences
words
```

fruit

```
head(fruit, n = 10)
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"
```

sentences

```
head(sentences, n = 10)
```

```
##  [1] "The birch canoe slid on the smooth planks."
##  [2] "Glue the sheet to the dark blue background."
##  [3] "It's easy to tell the depth of a well."
##  [4] "These days a chicken leg is a rare dish."
##  [5] "Rice is often served in round bowls."
##  [6] "The juice of lemons makes fine punch."
##  [7] "The box was thrown beside the parked truck."
##  [8] "The hogs were fed chopped corn and garbage."
##  [9] "Four hours of steady work faced us."
## [10] "Large size in stockings is hard to sell."
```

words

```
head(words, n = 10)
```

```
##  [1] "a"        "able"     "about"    "absolute" "accept"   "account"
##  [7] "achieve"  "across"   "act"      "active"
```

String operators

Make everything upper case

stringr

```
str_to_upper(fruit[1:10])
```

```
##  [1] "APPLE"        "APRICOT"      "AVOCADO
##  [5] "BELL PEPPER"  "BILBERRY"     "BLACKBE
##  [9] "BLOOD ORANGE" "BLUEBERRY"
```

base

```
toupper(fruit[1:10])
```

```
##  [1] "APPLE"        "APRICOT"      "AVOCADO"
##  [5] "BELL PEPPER"  "BILBERRY"     "BLACKBER
##  [9] "BLOOD ORANGE" "BLUEBERRY"
```

Make everything lower case

stringr

```
str_to_lower(sentences[1:10])
```

```
##  [1] "the birch canoe slid on the smooth pl
##  [2] "glue the sheet to the dark blue backg
##  [3] "it's easy to tell the depth of a well
##  [4] "these days a chicken leg is a rare di
##  [5] "rice is often served in round bowls."
##  [6] "the juice of lemons makes fine punch
##  [7] "the box was thrown beside the parked
##  [8] "the hogs were fed chopped corn and ga
##  [9] "four hours of steady work faced us."
## [10] "large size in stockings is hard to se
```

base

```
tolower(sentences[1:10])
```

```
##  [1] "the birch canoe slid on the smooth pla
##  [2] "glue the sheet to the dark blue backgr
##  [3] "it's easy to tell the depth of a well.
##  [4] "these days a chicken leg is a rare dis
##  [5] "rice is often served in round bowls."
##  [6] "the juice of lemons makes fine punch."
##  [7] "the box was thrown beside the parked t
##  [8] "the hogs were fed chopped corn and garl
##  [9] "four hours of steady work faced us."
## [10] "large size in stockings is hard to sel
```

Make title case

Notice these are slightly different

stringr

```
str_to_title("big movie that is really amazing")
```

```
## [1] "Big Movie That Is Really Amazing"
```

base (tools package - comes pre-installed)

```
tools::toTitleCase("big movie that is really am
```

```
## [1] "Big Movie that is Really Amazing"
```

Other options?

Look at `?toupper`

```
.simpleCap <- function(x) {
    s <- strsplit(x, " ")[[1]]
    paste(toupper(substring(s, 1, 1)), substring(s, 2),
          sep = "", collapse = " ")
}
.simpleCap("the quick brown fox jumps over the lazy brown dog")
```

```
## [1] "The Quick Brown Fox Jumps Over The Lazy Brown Dog"
```

Which mimics `str_to_title` rather than `tools::toTitleCase`.

```
tools::toTitleCase("the quick brown fox jumps over the lazy brown dog")
```

```
## [1] "The Quick Brown Fox Jumps over the Lazy Brown Dog"
```

Join strings together

stringr

```
str_c("green", "apple")
```

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

```
str_c("green", "apple", sep = " ")
```

```
## [1] "green apple"
```

```
str_c("green", "apple", sep = " : ")
```

```
## [1] "green : apple"
```

base

```
paste0("green", "apple")
```

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

```
paste("green", "apple")
```

```
## [1] "green apple"
```

```
paste("green", "apple", sep = " : ")
```

```
## [1] "green : apple"
```

Joining strings w/vectors

```
str_c("a", c("b", "c", "d"), 1:3)
```

```
## [1] "ab1" "ac2" "ad3"
```

```
str_c("a", c("b", "c", "d"), c(1, 1, 1, 2, 2, 2, 3, 3, 3))
```

```
## [1] "ab1" "ac1" "ad1" "ab2" "ac2" "ad2" "ab3" "ac3" "ad3"
```

- Note, the last vector could be created with `rep(1:3, each = 3)`
- Base version is the same but with `paste0`

Collapsing strings

```
str_c("a", c("b", "c", "d"), 1:3, collapse = "|")
```

```
## [1] "ab1|ac2|ad3"
```

```
str_c("a", c("b", "c", "d"), c(1, 1, 1, 2, 2, 2, 3, 3, 3), collapse = ":")
```

```
## [1] "ab1:ac1:ad1:ab2:ac2:ad2:ab3:ac3:ad3"
```

Calculate string length

```
words[1:3]
```

```
## [1] "a"     "able"  "about"
```

stringr

```
str_length(words[1:3])
```

```
## [1] 1 4 5
```

base

```
nchar(words[1:3])
```

```
## [1] 1 4 5
```

substrings: stringr

```
words[10:13]
```

```
## [1] "active"  "actual"  "add"     "address"
```

```
str_sub(words[10:13], 3)
```

```
## [1] "tive"  "tual"  "d"     "dress"
```

```
str_sub(words[10:13], 3, 5)
```

```
## [1] "tiv" "tua" "d"   "dre"
```

```
str_sub(words[10:13], -3)
```

```
## [1] "ive" "ual" "add" "ess"
```

substrings: base

```
substr(words[10:13], 3, nchar(words[10:13]))
```

```
## [1] "tive"  "tual"  "d"     "dress"
```

```
substr(words[10:13], 3, 5)
```

```
## [1] "tiv" "tua" "d"   "dre"
```

```
substr(words[10:13], nchar(words[10:13]) - 2, nchar(words[10:13]))
```

```
## [1] "ive" "ual" "add" "ess"
```

A few more substrings with stringr

```
words[10:13]
```

```
## [1] "active"  "actual"  "add"     "address"
```

Extract the second to second to last characters

```
str_sub(words[10:13], 2, -2)
```

```
## [1] "ctiv"  "ctua"  "d"     "ddres"
```

Use to modify

```
str_sub(words[10:13], 2, 4) <- "XX"
words[10:13]
```

```
## [1] "aXXve"  "aXXal"  "aXX"    "aXXess"
```

Locate where strings occur

```
fruit[c(1, 62, 2:5)]
```

```
## [1] "apple"       "pineapple"   "apricot"     "avocado"     "banana"
## [6] "bell pepper"
```

```
str_locate(fruit[c(1, 62, 2:5)], "ap")
```

```
##      start end
## [1,]     1   2
## [2,]     5   6
## [3,]     1   2
## [4,]    NA  NA
## [5,]    NA  NA
## [6,]    NA  NA
```

Trim white space

```
white_space <- c(" before", "after ", " both ")
```

stringr

```
str_trim(white_space)
```

```
## [1] "before" "after"  "both"
```

```
str_trim(white_space, side = "left")
```

```
## [1] "before" "after " "both "
```

```
str_trim(white_space, side = "right")
```

```
## [1] " before" "after"   " both"
```

base

```
trimws(white_space)
```

```
## [1] "before" "after"  "both"
```

```
trimws(white_space, which = "left")
```

```
## [1] "before" "after " "both "
```

```
trimws(white_space, which = "right")
```

```
## [1] " before" "after"   " both"
```

Pad white space

stringr

(we won't talk about base, but it's `sprintf` if you're interested)

```
strings <- c("abc", "abcdefg")
```

```
str_pad(strings, 10)
```

```
## [1] "       abc" "   abcdefg"
```

```
str_pad(strings, 10, side = "right")
```

```
## [1] "abc       " "abcdefg   "
```

```
str_pad(strings, 10, side = "both")
```

```
## [1] "   abc    " " abcdefg  "
```

Pad w/something else

```
string_nums <- as.character(1:15)
string_nums
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15"
```

```
str_pad(string_nums, 3, pad = "0")
```

```
##  [1] "001" "002" "003" "004" "005" "006" "007" "008" "009" "010" "011"
## [12] "012" "013" "014" "015"
```

Special characters

What you see isn't always what R sees

```
fox <- "the quick \nbrown fox \n \t jumps over the \t lazy dog"
fox
```

```
## [1] "the quick \nbrown fox \n \t jumps over the \t lazy dog"
```

- The above code has special characters telling R to break for a new line `\n` and to tab over `\t`.
- You can see how R "sees" the data using the `base::writeLines()` function

```
writeLines(fox)
```

```
## the quick
## brown fox
##       jumps over the      lazy dog
```

- `\n` and `\t` are probably the two most common.
- Use `?"'"` to see others

Special symbols

```
symbols <- c("\u03B1", "\u03B2", "\u03B3", "\u03B4", "\u03B5", "\u03B6")
symbols
```

```
## [1] "α" "β" "γ" "δ" "ε" "ζ"
```

These are called unicode characters and are consistent across programming languages. You can do plenty of other symbols outside of greek too. For example `"\u0807"` turns into ◻.

See http://graphemica.com/unicode/characters/ to find specific numbers

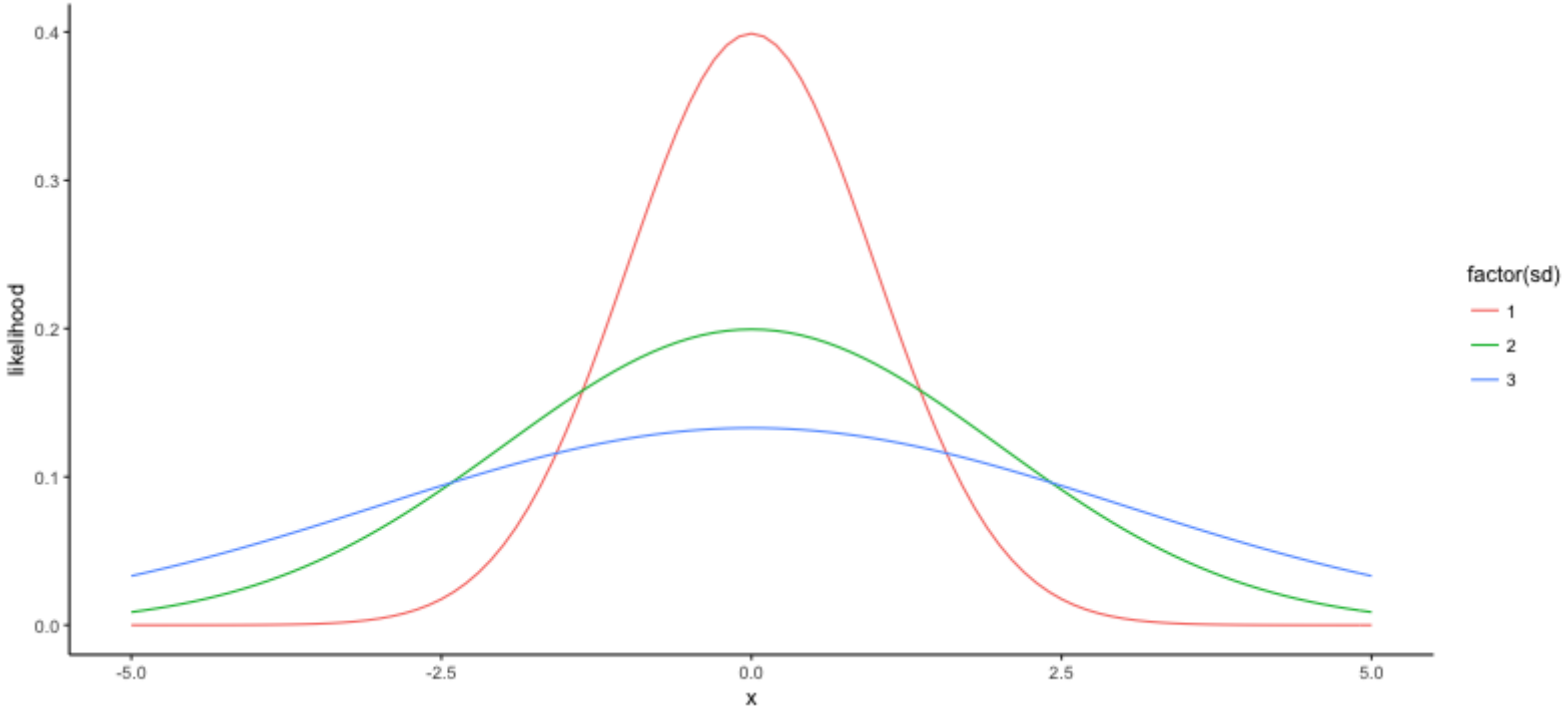
You might be thinking... Plots!

- Better ways to do that
- Let's create some normal distributions and annotate the plot

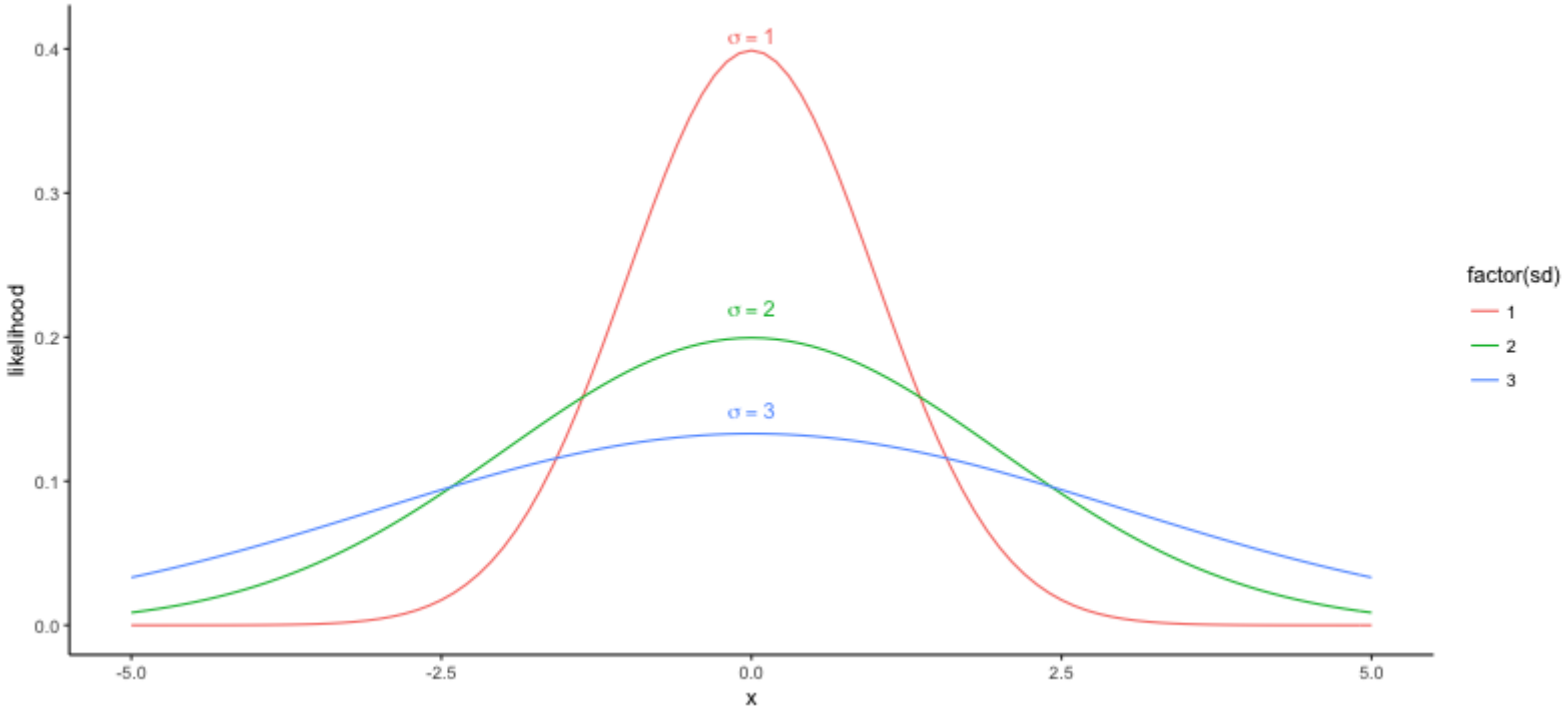
```
x <- seq(-5, 5, .1)
l1 <- dnorm(x, 0, 1)
l2 <- dnorm(x, 0, 2)
l3 <- dnorm(x, 0, 3)

d <- data.frame(x = x,
                likelihood = c(l1, l2, l3),
                sd = rep(1:3, each = length(x)))
```

Let's annotate this plot

```
library(tidyverse)
theme_set(theme_classic())
ggplot(d, aes(x, likelihood, color = factor(sd))) + geom_line()
```



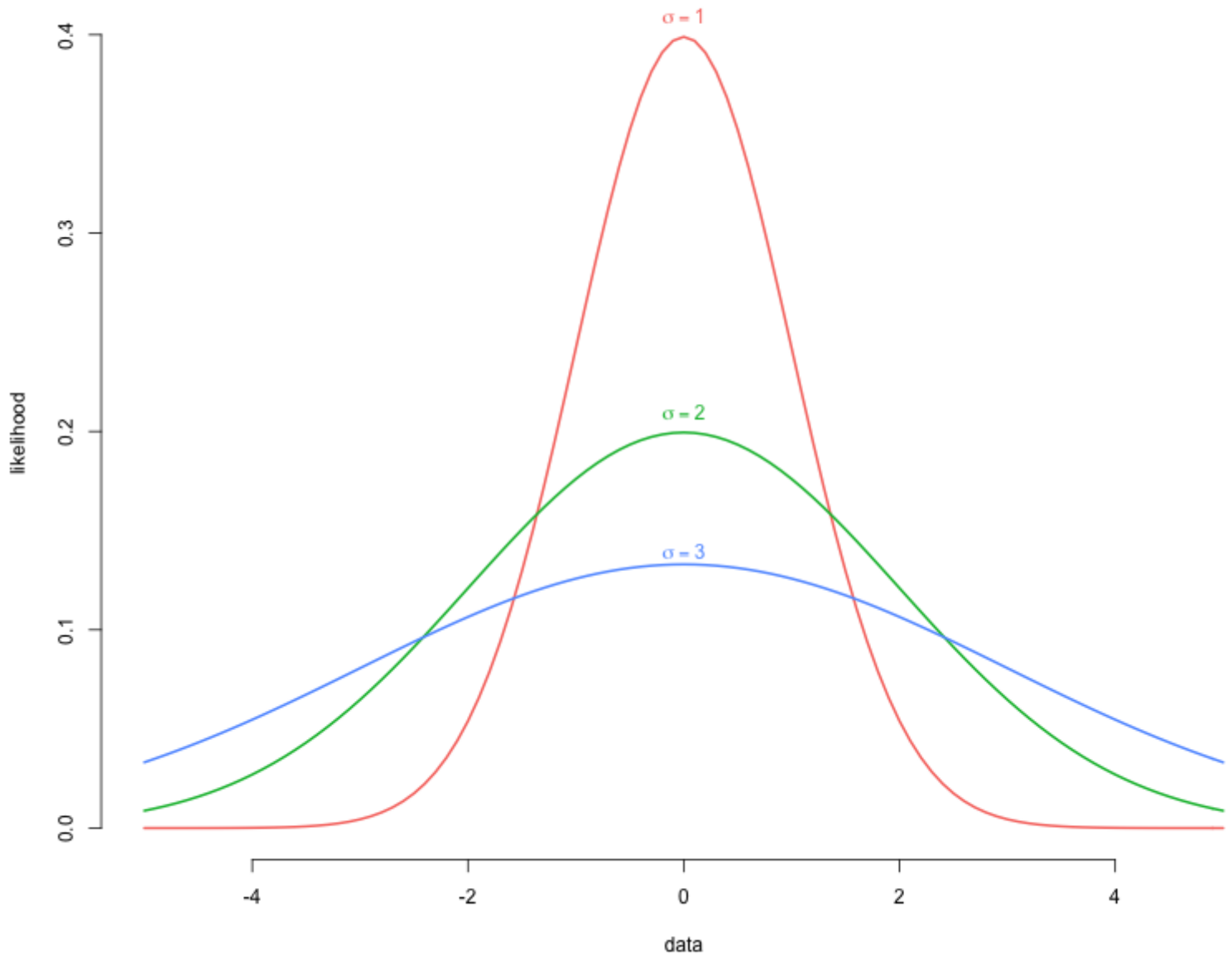
```
ggplot(d, aes(x, likelihood, color = factor(sd))) +
    geom_line() +
    annotate("text", x = 0, y = 0.41, label = "sigma == 1", parse = TRUE,
        color = "#F8766D") +
    annotate("text", x = 0, y = 0.22, label = "sigma == 2", parse = TRUE,
        color = "#00BA38") +
    annotate("text", x = 0, y = 0.15, label = "sigma == 3", parse = TRUE,
        color = "#619CFF")
```



Base plot annotations

- Use `expression` with no quotes

```
splt <- split(d, d$sd)

plot(splt[[1]]$x, splt[[1]]$likelihood,
    type = "l",
    col = "#F8766D",
    bty = "n",
    xlab = "data",
    ylab = "likelihood",
    lwd = 2)
lines(splt[[2]]$x, splt[[2]]$likelihood, col = "#00BA38", lwd = 2)
lines(splt[[3]]$x, splt[[3]]$likelihood, col = "#619CFF", lwd = 2)

text(0, 0.41, expression(sigma == 1), col = "#F8766D")
text(0, 0.21, expression(sigma == 2), col = "#00BA38")
text(0, 0.14, expression(sigma == 3), col = "#619CFF")
```


σ = 1
σ = 2
σ = 3
likelihood
0.0
0.1
0.2
0.3
0.4
-4
-2
0
2
4
data

Escaping special characters

- If you want the literal text to show up, instead of the symbol, you have to escape it it `\`.
- Because `\` itself is a special character, that means you need two: `\\`.

```
show_symbols <- c("\\u03B1", "\\u03B2", "\\u03B3", "\\u03B4", "\\u03B5", "\\u03B6")
show_symbols
```

```
## [1] "\\u03B1" "\\u03B2" "\\u03B3" "\\u03B4" "\\u03B5" "\\u03B6"
```

- There are also a host of regular expressions that have special characters that you'll need to escape if you want them to print too. We'll talk more about these later.

Pattern Matching

Locate pattern

- Really helpful function when learning/trying to figure out what R is doing: `str_view()`
    - Requires the *htmlwidgets* package

```
str_view(c("apple", "banana", "balloon"), "an")
```

Where is the pattern in a vector?

```
str_which(sentences, "red")
```

```
##  [1]  28  44  82 116 146 149 160 175 177 178 184 215 217 220 247 255 256
## [18] 274 277 279 293 311 345 368 372 387 388 485 494 512 539 551 576 582
## [35] 611 642 644 674 688 705
```

Or with `base::grep`

```
grep("red", sentences)
```

```
##  [1]  28  44  82 116 146 149 160 175 177 178 184 215 217 220 247 255 256
## [18] 274 277 279 293 311 345 368 372 387 388 485 494 512 539 551 576 582
## [35] 611 642 644 674 688 705
```

Extract the sentence

```
str_subset(sentences, "red")
```

```
##  [1] "The colt reared and threw the tall rider."
##  [2] "The wide road shimmered in the hot sun."
##  [3] "See the cat glaring at the scared mouse."
##  [4] "He ordered peach pie with ice cream."
##  [5] "Pure bred poodles have curls."
##  [6] "Mud was spattered on the front of his white shirt."
##  [7] "The sofa cushion is red and of light weight."
##  [8] "Torn scraps littered the stone floor."
##  [9] "The doctor cured him with these pills."
## [10] "The new girl was fired today at noon."
## [11] "The third act was dull and tired the players."
## [12] "Lire wires should be kept covered."
## [13] "It is hard to erase blue or red ink."
## [14] "The wreck occurred by the bank on Main Street."
## [15] "The box is held by a bright red snapper."
## [16] "The prince ordered his head chopped off."
## [17] "The houses are built of red clay bricks."
## [18] "The red tape bound the smuggled food."
## [19] "Nine men were hired to dig the ruins."
```

Or with `grep`

```
grep("red", sentences, value = TRUE)
```

```
##  [1] "The colt reared and threw the tall rider."
##  [2] "The wide road shimmered in the hot sun."
##  [3] "See the cat glaring at the scared mouse."
##  [4] "He ordered peach pie with ice cream."
##  [5] "Pure bred poodles have curls."
##  [6] "Mud was spattered on the front of his white shirt."
##  [7] "The sofa cushion is red and of light weight."
##  [8] "Torn scraps littered the stone floor."
##  [9] "The doctor cured him with these pills."
## [10] "The new girl was fired today at noon."
## [11] "The third act was dull and tired the players."
## [12] "Lire wires should be kept covered."
## [13] "It is hard to erase blue or red ink."
## [14] "The wreck occurred by the bank on Main Street."
## [15] "The box is held by a bright red snapper."
## [16] "The prince ordered his head chopped off."
## [17] "The houses are built of red clay bricks."
## [18] "The red tape bound the smuggled food."
## [19] "Nine men were hired to dig the ruins."
```

Related - count occurrences

```
str_count(sentences, "the")
```

```
##   [1] 1 2 1 0 0 0 1 0 0 0 2 0 2 1 1 1 0 1 1 1 1 1 2 0 2 1 0 1 1 0 1 2 2 1 1
##  [36] 0 0 0 1 1 2 1 1 1 1 2 1 1 1 0 1 1 0 1 0 0 2 0 1 2 2 1 1 1 1 0 1 1 2 2
##  [71] 2 1 1 0 0 0 2 0 0 0 1 2 0 0 0 1 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 2 0 2 3
## [106] 0 0 1 1 2 1 1 1 1 1 0 2 1 0 1 0 1 1 0 1 0 0 2 2 0 0 1 1 0 2 1 1 1 0 0
## [141] 1 2 0 1 0 0 0 1 1 1 1 0 0 1 0 0 2 1 1 0 2 1 2 0 1 0 0 2 0 1 1 1 0 0 1
## [176] 2 1 0 1 2 0 0 0 1 0 2 0 1 1 1 0 0 2 0 0 1 1 1 1 1 1 0 2 0 0 2 2 1 2 1
## [211] 0 1 1 0 0 0 0 0 0 1 0 0 0 0 3 0 0 0 2 1 2 1 0 2 0 0 0 2 0 1 2 0 1 1 0
## [246] 0 0 0 0 2 1 2 2 1 0 0 0 0 1 0 1 3 1 0 0 1 0 1 0 1 1 2 2 1 2 1 1 0 0 2
## [281] 0 1 1 0 1 1 0 0 2 0 2 0 0 2 1 0 0 0 0 0 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1
## [316] 1 1 0 0 0 1 0 0 1 2 1 0 1 0 2 0 0 0 2 0 2 1 1 1 0 1 2 2 0 1 0 0 1 2 0
## [351] 3 0 2 1 1 0 0 0 1 0 1 1 2 1 2 0 0 1 1 2 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0
## [386] 1 0 2 1 1 2 1 0 2 0 1 1 0 2 0 0 2 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 2 1 0
## [421] 1 1 0 1 0 1 3 0 0 1 1 1 1 0 1 1 1 0 0 0 1 1 1 0 0 1 2 2 0 1 0 0 1 0 1
## [456] 1 1 2 0 0 1 0 0 2 1 2 0 1 1 0 1 0 0 0 1 1 0 0 0 1 0 2 0 1 1 1 1 0 0 2
## [491] 1 0 0 0 1 3 1 1 1 0 1 0 1 0 2 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 1 1 2
## [526] 0 1 0 0 0 2 0 0 2 0 1 0 1 0 0 0 1 0 2 1 0 0 1 1 1 0 0 0 1 2 1 1 0 0 1
## [561] 2 1 0 2 1 1 2 0 1 1 0 2 0 2 1 1 1 0 3 2 0 1 2 2 2 2 0 2 2 0 1 2 2 0 0
## [596] 1 0 0 2 2 0 1 0 1 1 1 0 1 0 1 1 1 0 3 0 0 2 0 3 1 1 1 1 0 1 2 3 2 0 0
## [631] 1 0 1 1 1 2 1 0 0 0 0 0 1 1 0 1 1 2 0 0 1 0 0 1 1 0 0 2 0 1 0 1 1 2 1
```

Logical tests

- Sometimes, particularly with filtering, a logical test is best.

```
str_detect(sentences[1:100], "red")
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE
```

Or with `grepl`

```
grepl("red", sentences[1:100])
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE
```

Filtering example w/Scorecard data

```
library(rio)
scorecard <- import("../../data/Most-Recent-Cohorts-Scorecard-Elements.csv",
                    setclass = "tbl_df",
                    na = c("NULL", "PrivacySuppressed"))
head(scorecard)
```

```
## # A tibble: 6 x 122
##   UNITID   OPEID OPEID6                                INSTNM       CITY
##    <int>   <int>  <int>                                 <chr>      <chr>
## 1 100654  100200   1002            Alabama A & M University     Normal
## 2 100663  105200   1052 University of Alabama at Birmingham Birmingham
## 3 100690 2503400  25034                  Amridge University Montgomery
## 4 100706  105500   1055 University of Alabama in Huntsville Huntsville
## 5 100724  100500   1005            Alabama State University Montgomery
## 6 100751  105100   1051          The University of Alabama Tuscaloosa
## # ... with 117 more variables: STABBR <chr>, INSTURL <chr>, NPCURL <chr>,
## #   HCM2 <int>, PREDDEG <int>, CONTROL <int>, LOCALE <int>, HBCU <int>,
## #   PBI <int>, ANNHI <int>, TRIBAL <int>, AANAPII <int>, HSI <int>,
## #   NANTI <int>, MENONLY <int>, WOMENONLY <int>, RELAFFIL <int>,
## #   SATVR25 <int>, SATVR75 <int>, SATMT25 <int>, SATMT75 <int>,
## #   SATWR25 <int>, SATWR75 <int>, SATVRMID <int>, SATMTMID <int>,
```

Type of institution

- These data contain website address information.
- We can use the domain suffix to filter for different types of schools.

Let's filter for schools with a website ending in .com.

First, check our search

Is this what we want?

```
str_view(na.omit(scorecard$INSTURL[1:30]), "\\.com")
```

www.aamu.edu/
www.uab.edu
www.amridgeuniversity.edu
www.uah.edu
www.alasu.edu
www.ua.edu/
www.cacc.edu
www.athens.edu
www.aum.edu
www.auburn.edu
www.bsc.edu/
www.cv.edu
www.ccal.edu/
southuniversity.edu
www.escc.edu
www.faulknerstate.edu
www.faulkner.edu
www.gadsdenstate.edu
www.nbccosmetology.com
www.wallace.edu
www.wallacestate.edu

Next, Filter

```
coms <- filter(scorecard, str_detect(INSTURL, "\\.com"))
coms
```

```
## # A tibble: 1,276 x 122
##    UNITID   OPEID OPEID6                                INSTNM        CITY
##     <int>   <int>  <int>                                 <chr>       <chr>
##  1 101277 4187200  41872 New Beginning College of Cosmetology Albertville
##  2 101505  102200   1022    Jefferson State Community College  Birmingham
##  3 103811 2317800  23178     American Institute of Technology     Phoenix
##  4 103954  886400   8864       Arizona Academy of Beauty-East      Tucson
##  5 104504 2582700  25827             Cortiva Institute-Tucson      Tucson
##  6 104911 1218400  12184 International Academy of Hair Design       Tempe
##  7 105482 2623800  26238         Cortiva Institute-Scottsdale  Scottsdale
##  8 105659 1168900  11689             Refrigeration School Inc     Phoenix
##  9 105677 2113800  21138      Roberto-Venn School of Luthiery     Phoenix
## 10 105701 3005000  30050              Hair Academy of Safford     Safford
## # ... with 1,266 more rows, and 117 more variables: STABBR <chr>,
## #   INSTURL <chr>, NPCURL <chr>, HCM2 <int>, PREDDEG <int>, CONTROL <int>,
## #   LOCALE <int>, HBCU <int>, PBI <int>, ANNHI <int>, TRIBAL <int>,
## #   AANAPII <int>, HSI <int>, NANTI <int>, MENONLY <int>, WOMENONLY <int>,
## #   RELAFFIL <int>, SATVR25 <int>, SATVR75 <int>, SATMT25 <int>,
```

What if we wanted a *domain* variable

- Multiple ways to do this. Discuss with your neighbor how you would go about it.

Did you think of `separate`?

- What do you think; would the following code work?

```
scorecard %>%
    separate(INSTURL, c("prefix", "site", "domain"), sep = ".")
```

Try it

Why does this fail?

```
scorecard %>%
    separate(INSTURL, c("prefix", "site", "domain"), sep = ".") %>%
    select(1, prefix, site, domain)
```

```
## Warning: Too many values at 7626 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
```

```
## # A tibble: 7,703 x 4
##    UNITID prefix  site domain
##  *  <int>  <chr> <chr>  <chr>
##  1 100654
##  2 100663
##  3 100690
##  4 100706
##  5 100724
##  6 100751
##  7 100760
##  8 100812
##  9 100830
```

. is a regex special character

- Escape . and you get the expected result. Why are there still some warnings?

```
scorecard %>%
    separate(INSTURL, c("prefix", "site", "domain"), sep = "\\.") %>%
    select(1, prefix, site, domain)
```

```
## Warning: Too many values at 304 locations: 61, 64, 118, 151, 206, 265, 296,
## 379, 421, 482, 522, 639, 649, 650, 655, 661, 666, 667, 764, 793, ...
```

```
## Warning: Too few values at 731 locations: 14, 44, 48, 94, 108, 114, 115,
## 117, 120, 129, 133, 144, 160, 168, 169, 185, 192, 197, 204, 221, ...
```

```
## # A tibble: 7,703 x 4
##    UNITID prefix              site domain
##  *  <int>  <chr>              <chr>  <chr>
##  1 100654    www              aamu   edu/
##  2 100663    www               uab    edu
##  3 100690    www amridgeuniversity    edu
##  4 100706    www               uah    edu
##  5 100724    www             alasu    edu
```

str_sub?

```
scorecard %>%
    mutate(domain = str_sub(INSTURL, -3)) %>%
    select(1, domain)
```

```
## # A tibble: 7,703 x 2
##    UNITID domain
##     <int>  <chr>
##  1 100654    du/
##  2 100663    edu
##  3 100690    edu
##  4 100706    edu
##  5 100724    edu
##  6 100751    du/
##  7 100760    edu
##  8 100812    edu
##  9 100830    edu
## 10 100858    edu
## # ... with 7,693 more rows
```

So what do we do?

- Need to get rid of `/` if it's the last character, then use `str_sub`
- Replace `"/"` with `""`, using `str_replace`, **BUT**, only if "/" is the last character
- Regular expressions come in handy here - this is a very basic one
    - Special character `$` is an anchor for the end of the string

```
str_view(na.omit(scorecard$INSTURL[1:10]), "/$")
```

Replace with nothing

- `stringr::str_replace` or `base::sub`

```
scorecard %>%
    mutate(INSTURL = str_replace(INSTURL, "/$", "")) %>%
  select(INSTNM, INSTURL)
```

```
## # A tibble: 7,703 x 2
##                                    INSTNM                    INSTURL
##                                     <chr>                      <chr>
##  1            Alabama A & M University                www.aamu.edu
##  2 University of Alabama at Birmingham                 www.uab.edu
##  3                  Amridge University www.amridgeuniversity.edu
##  4 University of Alabama in Huntsville                 www.uah.edu
##  5            Alabama State University                www.alasu.edu
##  6          The University of Alabama                   www.ua.edu
##  7   Central Alabama Community College                 www.cacc.edu
##  8             Athens State University               www.athens.edu
##  9     Auburn University at Montgomery                  www.aum.edu
## 10                  Auburn University               www.auburn.edu
## # ... with 7,693 more rows
```

Create domain variable

(more complicated than I intended)

```
scorecard <- scorecard %>%
  mutate(decimals = str_count(INSTURL, "\\."))

scorecard %>%
  select(INSTNM, INSTURL, decimals)
```

```
## # A tibble: 7,703 x 3
##                                  INSTNM                    INSTURL decimals
##                                   <chr>                      <chr>    <int>
##  1            Alabama A & M University              www.aamu.edu/        2
##  2 University of Alabama at Birmingham                www.uab.edu        2
##  3                 Amridge University www.amridgeuniversity.edu        2
##  4 University of Alabama in Huntsville                www.uah.edu        2
##  5            Alabama State University              www.alasu.edu        2
##  6          The University of Alabama                www.ua.edu/        2
##  7   Central Alabama Community College               www.cacc.edu        2
##  8             Athens State University             www.athens.edu        2
##  9     Auburn University at Montgomery                www.aum.edu        2
```

```
scorecard %>%
  count(decimals)
```

```
## # A tibble: 6 x 2
##   decimals     n
##      <int> <int>
## 1        1   731
## 2        2  6591
## 3        3   284
## 4        4    19
## 5        5     1
## 6       NA    77
```

Finally, create domain

```
library(purrr)
scorecard$domain <- map2_chr(strsplit(scorecard$INSTURL, "\\."),
                             scorecard$decimals,
                             ~.x[.y + 1])
scorecard %>%
  select(INSTNM, INSTURL, domain)
```

Something to be aware of...

- Both `stringr::str_replace` and `base::sub` only replace the first match.

```
str_replace(sentences[1], " ", "_")
```

```
## [1] "The_birch canoe slid on the smooth planks."
```

```
sub(" ", "_", sentences[1])
```

```
## [1] "The_birch canoe slid on the smooth planks."
```

- To replace all instances, use `stringr::str_replace_all` or `base::gsub`

```
str_replace_all(sentences[1], " ", "_")
```

```
## [1] "The_birch_canoe_slid_on_the_smooth_planks."
```

```
gsub(" ", "_", sentences[1])
```

```
## [1] "The_birch_canoe_slid_on_the_smooth_planks."
```

Final note on pattern matching

- You can get a lot done with just basic pattern matching and knowing when to escape characters, particularly when you combine the basic pattern matching with other functions.
- Regular expressions make your pattern searching abilities much more powerful, and your code much less verbose

String splitting

- Say we want to split the sentences into words
- `stringr::str_split` or `base::strsplit`
- The output will generally be trickier than you might imagine, because we're starting with an atomic vector - not a data frame.

Split to words

What's tricky about the below?

```
str_split(sentences[1:10], " ")
```

```
## [[1]]
## [1] "The"     "birch"   "canoe"   "slid"    "on"      "the"     "smooth"
## [8] "planks."
##
## [[2]]
## [1] "Glue"        "the"         "sheet"       "to"          "the"
## [6] "dark"        "blue"        "background."
##
## [[3]]
## [1] "It's"  "easy"  "to"    "tell"  "the"   "depth" "of"    "a"     "well."
##
## [[4]]
## [1] "These"   "days"    "a"       "chicken" "leg"     "is"      "a"
## [8] "rare"    "dish."
##
## [[5]]
## [1] "Rice"   "is"     "often"  "served" "in"     "round"  "bowls."
```

Alternative format

- start with a data frame

```
sentences_df <- tibble(sentence_num = seq_along(sentences),
                       sentence = sentences)
sentences_df
```

```
## # A tibble: 720 x 2
##    sentence_num                                      sentence
##           <int>                                         <chr>
##  1            1  The birch canoe slid on the smooth planks.
##  2            2 Glue the sheet to the dark blue background.
##  3            3      It's easy to tell the depth of a well.
##  4            4    These days a chicken leg is a rare dish.
##  5            5        Rice is often served in round bowls.
##  6            6       The juice of lemons makes fine punch.
##  7            7 The box was thrown beside the parked truck.
##  8            8 The hogs were fed chopped corn and garbage.
##  9            9         Four hours of steady work faced us.
## 10           10    Large size in stockings is hard to sell.
## # ... with 710 more rows
```

Split to words

The *tidytext* library provides a means of taking data from a format like the previous slide, and splitting it by word but keeping the same structure.

```
# install.packages("tidytext")
library(tidytext)

word_df <- sentences_df %>%
    unnest_tokens(word, sentence)
word_df
```

```
## # A tibble: 5,748 x 2
##    sentence_num   word
##           <int>  <chr>
##  1            1    the
##  2            1  birch
##  3            1  canoe
##  4            1   slid
##  5            1     on
##  6            1    the
##  7            1 smooth
##  8            1 planks
```

Manipulations, etc.

- Because it's a data frame, we can now work with it much the way we have all other data frames.
- What's the most frequent words

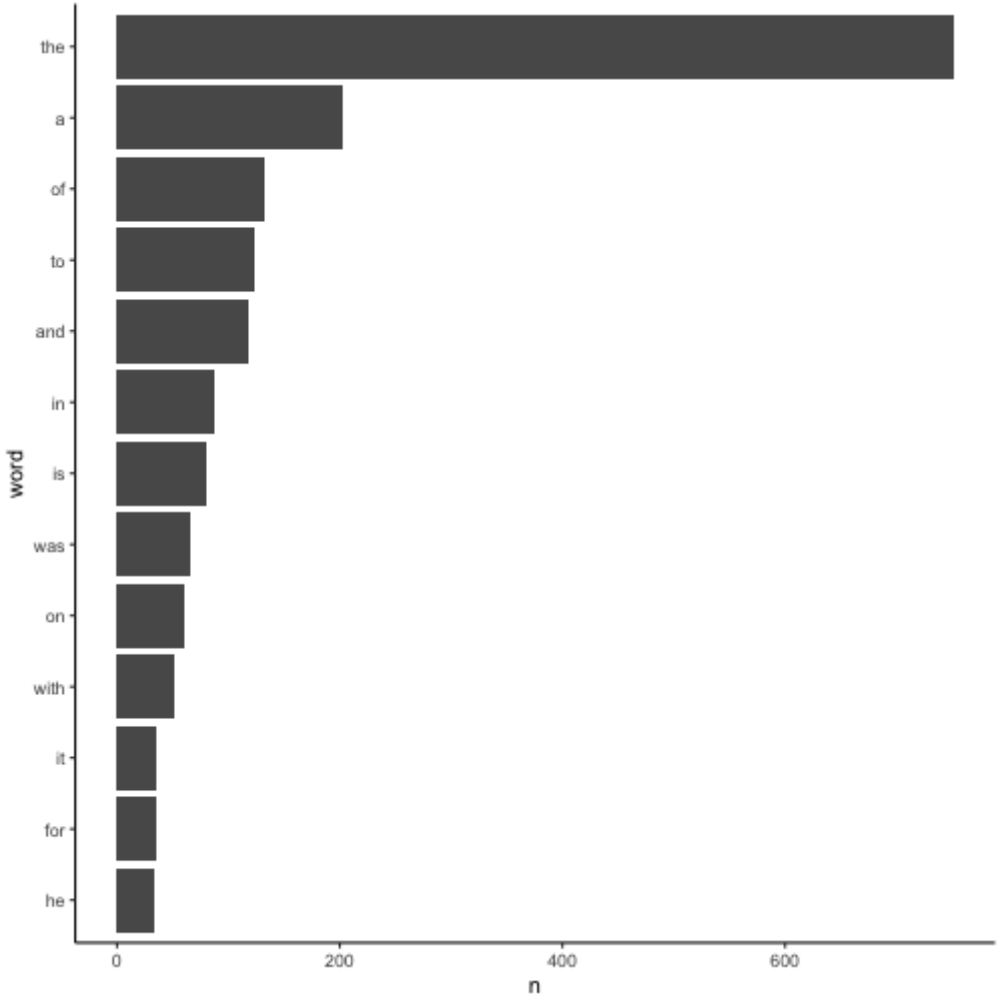
```
word_df %>%
    count(word) %>%
    arrange(desc(n))
```

```
## # A tibble: 1,904 x 2
##     word     n
##    <chr> <int>
##  1   the   751
##  2     a   202
##  3    of   132
##  4    to   123
##  5   and   118
##  6    in    87
##  7    is    81
##  8   was    66
##  9    on    60
```

Plot high frequency words

- Below, most should be familiar with the exception of `reorder`. In this case, it's defining word as an ordered factor, ordered according to *n*. This gets the plotting order correct

```
word_df %>%
    count(word) %>%
    filter(n > 30) %>%
    mutate(word = reorder(word, n)) %>%
    ggplot(aes(word, n)) +
    geom_col() +
    coord_flip()
```



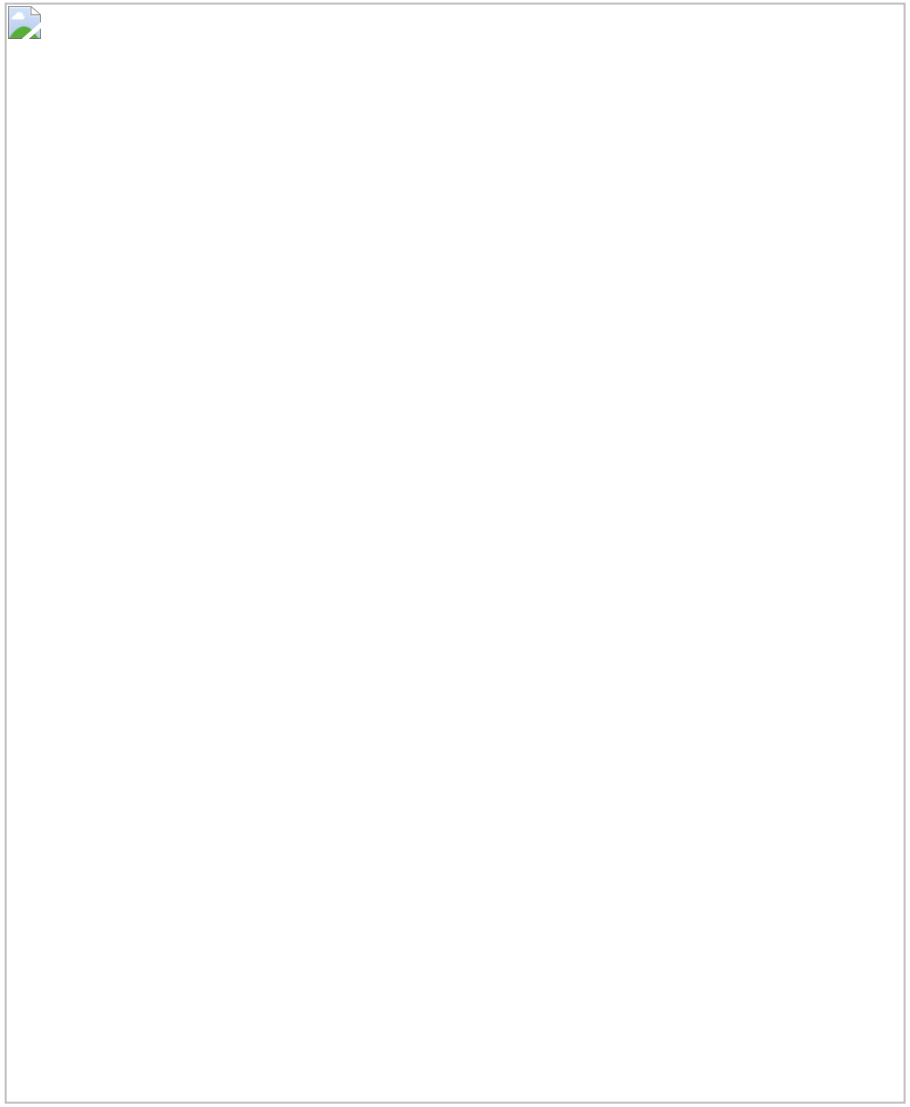
simplifying

- Simplify it to a matrix, instead of a list, using `simplify = TRUE`.

```
str_split(sentences[1:10], " ", simplify = TRUE)
```

```
##       [,1]    [,2]    [,3]    [,4]        [,5]      [,6]    [,7]
##  [1,] "The"   "birch" "canoe" "slid"      "on"      "the"   "smooth"
##  [2,] "Glue"  "the"   "sheet" "to"        "the"     "dark"  "blue"
##  [3,] "It's"  "easy"  "to"    "tell"      "the"     "depth" "of"
##  [4,] "These" "days"  "a"     "chicken"   "leg"     "is"    "a"
##  [5,] "Rice"  "is"    "often" "served"    "in"      "round" "bowls."
##  [6,] "The"   "juice" "of"    "lemons"    "makes"   "fine"  "punch."
##  [7,] "The"   "box"   "was"   "thrown"    "beside"  "the"   "parked"
##  [8,] "The"   "hogs"  "were"  "fed"       "chopped" "corn"  "and"
##  [9,] "Four"  "hours" "of"    "steady"    "work"    "faced" "us."
## [10,] "Large" "size"  "in"    "stockings" "is"      "hard"  "to"
##       [,8]          [,9]
##  [1,] "planks."     ""
##  [2,] "background." ""
##  [3,] "a"           "well."
##  [4,] "rare"        "dish."
##  [5,] ""            ""
```

From there, tidy, if you want to



```
words <- str_split(sentences, " ", simplify = TRUE)
tidy_words <- as.data.frame(as.table(words), responseName = "word")
head(tidy_words)
```

```
##   Var1 Var2  word
## 1    A    A   The
## 2    B    A  Glue
## 3    C    A  It's
## 4    D    A These
## 5    E    A  Rice
## 6    F    A   The
```

What about `separate()`

- Why would separate not be ideal for this sort of task?

- Requires data be in a data frame
- More importantly - requires you define the new variables that the current variable will be separated into

Extracting patterns

- `str_extract` is most useful when using regular expressions.
- We'll use another simple one, `"|"`, which means "or" (just like in logical tests)
- First subset to sentences with colors in them
- Extract colors

```
color_search <- c("red|orange|green|yellow|green|blue|purple")
subset_sentences <- str_subset(sentences, color_search)
subset_sentences[1:10]
```

```
##  [1] "Glue the sheet to the dark blue background."
##  [2] "Two blue fish swam in the tank."
##  [3] "The colt reared and threw the tall rider."
##  [4] "The wide road shimmered in the hot sun."
##  [5] "See the cat glaring at the scared mouse."
##  [6] "A wisp of cloud hung in the blue air."
##  [7] "Leaves turn brown and yellow in the fall."
##  [8] "He ordered peach pie with ice cream."
##  [9] "Pure bred poodles have curls."
## [10] "The spot on the blotter was made by green ink."
```

```
str_extract(subset_sentences, color_search)
```

```
##  [1] "blue"   "blue"   "red"    "red"    "red"    "blue"   "yellow"
##  [8] "red"    "red"    "green"  "red"    "red"    "blue"   "red"
## [15] "red"    "red"    "red"    "blue"   "red"    "blue"   "red"
## [22] "green"  "red"    "red"    "red"    "red"    "red"    "red"
## [29] "green"  "red"    "green"  "red"    "purple" "green"  "red"
## [36] "red"    "red"    "red"    "red"    "blue"   "red"    "blue"
## [43] "red"    "red"    "red"    "red"    "green"  "green"  "green"
## [50] "red"    "red"    "yellow" "red"    "orange" "red"    "red"
## [57] "red"
```

Back to scorecard

- Use `str_extract` to create a dummy variable indicating whether or not the school has a .com domain

```
scorecard %>%
    mutate(com_domain = str_extract(INSTURL, "\\.com"),
           com_domain = ifelse(is.na(com_domain), 0, 1)) %>%
    select(1, com_domain)
```

```
## # A tibble: 7,703 x 2
##    UNITID com_domain
##     <int>      <dbl>
##  1 100654          0
##  2 100663          0
##  3 100690          0
##  4 100706          0
##  5 100724          0
##  6 100751          0
##  7 100760          0
##  8 100812          0
##  9 100830          0
```

Probably out of time, but...

Practice

- Select the first letter from every word. What's the most common letter?
- Calculate the number of digits in `unitid` from the scorecard data.
  - Are they all the same?
  - How many instances of each length?
  - Can you make them all the same length?
- Use the institution names (INSTNM) to select for colleges with "Community College" in their name. Do the same for "University". How does the number of community colleges compare to the number of universities?

regular expressoins

Agenda

- Basics of regular expressions
  - You're not expected to be an expert from one brief lecture
- Provide resources for learning more
- Purpose: Generally to make you aware of them

Disclaimer

- Before we get started, I think it's worth mentioning this is NOT a specialty area for me.
- Still working hard myself to understand all of this
- Today is mostly about exposure and awareness

Resources

(Some of which I'll reference and others I won't)

- Sanchez: Handling and Processing Strings in R (available here)
- Li and Bryan: Regular Expression in R (available here)
- Espenoza slides: (available here)
- Google (lots of other stuff out there)

One of the nice things about regular expressions, in particular, is that they are pretty language agnostic, so you can read something about regular expressions in Java or PHP or something else it will probably mostly transfer to R.

What is a regular expression?

- "an 'instruction' given to a function on what and how to match or replace strings" (Eden, 2007)
- *Metacharacters* are special characters that define specific operations
  - can be interpreted as standard characters, provided the appropriate syntax is used.
  - include the following:

```
##  [1] $  *  +  .  ?  [  ^  {  |  (  \\
```

- *Sequences* define sequences of characters to match
- *Character classes* define ranges to match or not match

Why does this matter?

Many of the built-in string functions in R take regular expressions as their arguments. If you're unaware of how regular expressions work, you could end up with unexpected behavior. For example, the following seems like it should work, but it will not.

```
library(stringr)
string <- "School is fun. Especially recess. That's the best part. I love recess."
str_split(string, ".")
```

```
## [[1]]
##  [1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [24] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [47] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [70] "" ""
```

We can get the behavior we intend by overriding the metacharacter

```
str_split(string, "\\.")
```

```
## [[1]]
## [1] "School is fun"          " Especially recess"    " That's the best part"
## [4] " I love recess"         ""
```

Quick overview of metacharacters

(we'll talk about each in more detail)

METACHARACTER	OPERATION
$	Matches the end of a string
*	Matches preceding character at least 0 times
+	Matches preceding character at least 1 time
.	Matches any single character (i.e., skip operator)
?	Matches preceding character 0 or 1 time
[	Define character list or character classes
^	Matches the start of a string
{	Define n to m matches of preceding character
|	Or operator
(	Define groupings for backreferencing
\	Suppress metacharacters and define anchor sequences

Quantifiers

METACHARACTER	OPERATION
*	Matches preceding character at least 0 times
+	Matches preceding character at least 1 time
?	Matches preceding character 0 or 1 time
{	Define n to m matches of preceding character

Examples taken from http://stat545.com/block022_regular-expression.html

```
letterSet <- c("a", "ab", "acb", "accb", "acccb", "accccb")
str_subset(letterSet, "ac*b")
```

```
## [1] "ab"    "acb"   "accb"  "acccb"  "accccb"
```

```
str_subset(letterSet, "ac+b")
```

```
## [1] "acb"   "accb"  "acccb"  "accccb"
```

METACHARACTER	OPERATION
?	Matches preceding character 0 or 1 time
{	Define n to m matches of preceding character

```
## [1] "a"      "ab"     "acb"    "accb"   "acccb"  "accccb"
```

```
str_subset(letterSet, "ac?b")
```

```
## [1] "ab"  "acb"
```

```
str_subset(letterSet, "ac{2}b")
```

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

```
str_subset(letterSet, "ac{2,}b")
```

```
## [1] "accb"   "acccb"  "accccb"
```

```
## [1] "a"      "ab"     "acb"    "accb"   "acccb"  "accccb"
```

```
str_subset(letterSet, "ac{1,3}b")
```

```
## [1] "acb"   "accb"  "acccb"
```

```
str_subset(letterSet, "ac{0,3}b")
```

```
## [1] "ab"    "acb"   "accb"  "acccb"
```

Position matching

METACHARACTER	OPERATION
$	Matches the end of a string
^	Matches the start of a string
\	Suppress metacharacters and define anchor sequences

Anchor sequences are provided on the right (from Sanchez, p. 61). These should not be confused with R escape characters, such as `\n` and `\t`, for new line and tab, respectively.

Anchor Sequences in R

Anchor	Description
\\d	match a digit character
\\D	match a non-digit character
\\s	match a space character
\\S	match a non-space character
\\w	match a word character
\\W	match a non-word character
\\b	match a word boundary
\\B	match a non-(word boundary)
\\h	match a horizontal space
\\H	match a non-horizontal space
\\v	match a vertical space
\\V	match a non-vertical space

Match the end of a string

```
itemIDs <- c("RF3L02E03", "RF3M08E05", "RF3H10E08", "RL1L03E03", "RL1M05E05",
             "RL1H10E04", "RI2L03E07", "RI2M06E05", "RI2HSAMPLEE06",
             "WR4L02E03", "WR4M06E06", "WR4H09E03", "WR9L03E04", "WR9M08E04",
             "WR9H12E04", "LA1L01E11", "LA1M06E04", "LA1H09E04", "WR2L03E05",
             "WR2M06E05", "WR2H10E05", "LA2LSAMPLEE03", "LA2M06E04",
             "LA2H09E08", "RF4L02E03", "RF4M08E06", "RF4H09E07", "RL7L02E07",
             "RL7M06E06", "RL7H10E06", "RI1L02E07", "RI1M07E08", "RI1H11E07",
             "WR1L02E07", "WR1M07E07", "WR1H11E08")

# Select fourth grade items
str_subset(itemIDs, "4$")
```

```
## [1] "RL1H10E04" "WR9L03E04" "WR9M08E04" "WR9H12E04" "LA1M06E04" "LA1H09E04"
## [7] "LA2M06E04"
```

Match the start of a string

```
# Select RF items
str_subset(itemIDs, "^RF")
```

```
## [1] "RF3L02E03" "RF3M08E05" "RF3H10E08" "RF4L02E03" "RF4M08E06" "RF4H09E07"
```

```
# Select WR items
str_subset(itemIDs, "^WR")
```

```
##  [1] "WR4L02E03" "WR4M06E06" "WR4H09E03" "WR9L03E04" "WR9M08E04"
##  [6] "WR9H12E04" "WR2L03E05" "WR2M06E05" "WR2H10E05" "WR1L02E07"
## [11] "WR1M07E07" "WR1H11E08"
```

Note that in this case, the result would be the same without using the `^` metacharacter, but it's safer to go ahead and use it anyway. In other cases, there may be matches with the same pattern that are not at the beginning.

Other examples

Examples taken from http://stat545.com/block022_regular-expression.html

```
string2 <- c("abcd", "cdab", "cabd", "c abd")
str_subset(string2, "ab")
```

```
## [1] "abcd"  "cdab"  "cabd"  "c abd"
```

```
str_subset(string2, "^ab")
```

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

```
str_subset(string2, "ab$")
```

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

Search start of words, rather than strings

```
## [1] "abcd"  "cdab"  "cabd"  "c abd"
```

```
str_subset(string2, "\\ba")
```

```
## [1] "abcd"  "c abd"
```

```
str_subset(string2, "\\b a")
```

```
## [1] "c abd"
```

```r
ids <- c("123225-5417", "132123-132975", "321579-123569")
str_subset(ids, "123")
```

```
## [1] "123225-5417"   "132123-132975" "321579-123569"
```

```r
str_subset(ids, "^123")
```

```
## [1] "123225-5417"
```

```r
str_subset(ids, "\\b123") # New word starts with 123
```

```
## [1] "123225-5417"   "321579-123569"
```

```r
str_subset(ids, "\\B123") # New word does not start with 123
```

```
## [1] "132123-132975"
```

Digits

```
digits <- c("Charlie", "Charlie2", "Mary", "Marianne", "Mary2", "15")
str_subset(digits, "\\d")
```

```
## [1] "Charlie2" "Mary2"    "15"
```

```
str_subset(digits, "\\D")
```

```
## [1] "Charlie"  "Charlie2" "Mary"     "Marianne" "Mary2"
```

Spaces

```
string
```

```
## [1] "School is fun. Especially recess. That's the best part. I love recess."
```

```
str_replace_all(string, "\\s", "_")
```

```
## [1] "School_is_fun._Especially_recess._That's_the_best_part._I_love_recess."
```

```
str_replace_all(string, "\\S", "_")
```

```
## [1] "______ __ ____ _________ _______ ______ ___ ____ _____ _ ____ _______"
```

Words

```
str_replace_all(string, "\\w", "_")
```

```
## [1] "______ __ ___. _________ ______. ____'_ ___ ____ ____. _ ____ ______."
```

```
str_replace_all(string, "\\W", "_")
```

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

Operators

METACHARACTER	OPERATION
.	Matches any single character (i.e., skip operator)
[	Define character list or character classes
|	Or operator
(	Define groupings for backreferencing
\	Suppress metacharacters and define anchor sequences

Skip characters

```
string2
```

```
## [1] "abcd"  "cdab"  "cabd"  "c abd"
```

```
str_subset(string2, ".ab")
```

```
## [1] "cdab"  "cabd"  "c abd"
```

```
str_subset(string2, "ab.")
```

```
## [1] "abcd"  "cabd"  "c abd"
```

Select high items (easy)

```
head(itemIDs)
```

```
## [1] "RF3L02E03" "RF3M08E05" "RF3H10E08" "RL1L03E03" "RL1M05E05" "RL1H10E04"
```

```
str_subset(itemIDs, "H")
```

```
##  [1] "RF3H10E08"     "RL1H10E04"     "RI2HSAMPLEE06" "WR4H09E03"
##  [5] "WR9H12E04"     "LA1H09E04"     "WR2H10E05"     "LA2H09E08"
##  [9] "RF4H09E07"     "RL7H10E06"     "RI1H11E07"     "WR1H11E08"
```

Select low items (more difficult)

```
head( str_subset(itemIDs, "L") ) # Fails
```

```
## [1] "RF3L02E03"     "RL1L03E03"     "RL1M05E05"     "RL1H10E04"
## [5] "RI2L03E07"     "RI2HSAMPLEE06"
```

```
str_subset(itemIDs, "...L") # Fails
```

```
##  [1] "RF3L02E03"     "RL1L03E03"     "RI2L03E07"     "RI2HSAMPLEE06"
##  [5] "WR4L02E03"     "WR9L03E04"     "LA1L01E11"     "WR2L03E05"
##  [9] "LA2LSAMPLEE03" "RF4L02E03"     "RL7L02E07"     "RI1L02E07"
## [13] "WR1L02E07"
```

```
str_subset(itemIDs, "^.{3}L") # Success!
```

```
##  [1] "RF3L02E03"     "RL1L03E03"     "RI2L03E07"     "WR4L02E03"
##  [5] "WR9L03E04"     "LA1L01E11"     "WR2L03E05"     "LA2LSAMPLEE03"
##  [9] "RF4L02E03"     "RL7L02E07"     "RI1L02E07"     "WR1L02E07"
```

Or

Select RL or WR Items

```
str_subset(itemIDs, "RL|WR")
```

```
##  [1] "RL1L03E03" "RL1M05E05" "RL1H10E04" "WR4L02E03" "WR4M06E06"
##  [6] "WR4H09E03" "WR9L03E04" "WR9M08E04" "WR9H12E04" "WR2L03E05"
## [11] "WR2M06E05" "WR2H10E05" "RL7L02E07" "RL7M06E06" "RL7H10E06"
## [16] "WR1L02E07" "WR1M07E07" "WR1H11E08"
```

```
string3 <- c("^ab", "ab", "abc", "abd", "abe", "ab 12")
str_subset(string3, "bc|be")
```

```
## [1] "abc" "abe"
```

```
str_subset(string3, "1|c")
```

```
## [1] "abc"   "ab 12"
```

Backreferences

Capture a pattern

```
string3 <- c("^ab", "ab", "abc", "abd", "abe", "ab 12")
str_replace_all(string3, "(ab)", "hello")
```

```
## [1] "^hello"   "hello"    "helloc"   "hellod"   "helloe"   "hello 12"
```

```
str_replace_all(string3, "(ab)", "\\1 oy!")
```

```
## [1] "^ab oy!"   "ab oy!"    "ab oy!c"   "ab oy!d"   "ab oy!e"   "ab oy! 12"
```

Two backreferences

```
string3 <- c("^ab", "ab", "abc", "abd", "abe", "ab 12")
str_replace_all(string3, "(ab)(c)", "\\2! \\1")
```

```
## [1] "^ab"   "ab"    "c! ab" "abd"   "abe"   "ab 12"
```

```
str_replace_all(string3, "(ab)( )", "\\1_")
```

```
## [1] "^ab"   "ab"    "abc"   "abd"   "abe"   "ab_12"
```

```
str_replace_all(string3, "(ab)( )(12)", "\\3_\\1_867")
```

```
## [1] "^ab"       "ab"        "abc"       "abd"       "abe"       "12_ab_867"
```

Back to scorecard example

```
scorecard %>%
  select(INSTNM, INSTURL, domain) %>%
  slice(90:100)
```

```
## # A tibble: 11 x 3
##                                              INSTNM
##                                               <chr>
##  1     Cochise County Community College District
##  2                Empire Beauty School-Flagstaff
##  3                 Empire Beauty School-Chandler
##  4                      Cortiva Institute-Tucson
##  5             Avalon School of Cosmetology-Mesa
##  6                       Eastern Arizona College
##  7 Embry-Riddle Aeronautical University-Prescott
##  8     Frank Lloyd Wright School of Architecture
##  9                   Glendale Community College
## 10                      Grand Canyon University
## 11          International Academy of Hair Design
## # ... with 2 more variables: INSTURL <chr>, domain <chr>
```

Back to scorecard example

```
scorecard %>%
  mutate(domain = str_replace_all(domain, "(.)(/).*+", "\\1")) %>%
  select(INSTNM, INSTURL, domain) %>%
  slice(90:100)
```

```
## # A tibble: 11 x 3
##                                                INSTNM
##                                                 <chr>
##  1     Cochise County Community College District
##  2                Empire Beauty School-Flagstaff
##  3                 Empire Beauty School-Chandler
##  4                      Cortiva Institute-Tucson
##  5             Avalon School of Cosmetology-Mesa
##  6                       Eastern Arizona College
##  7 Embry-Riddle Aeronautical University-Prescott
##  8     Frank Lloyd Wright School of Architecture
##  9                   Glendale Community College
## 10                      Grand Canyon University
## 11          International Academy of Hair Design
## # ... with 2 more variables: INSTURL <chr>, domain <chr>
```

Suppress metacharacters

```
string3
```

```
## [1] "^ab"   "ab"    "abc"   "abd"   "abe"   "ab 12"
```

```
str_subset(string3, "^ab")
```

```
## [1] "ab"    "abc"   "abd"   "abe"   "ab 12"
```

```
str_subset(string3, "\\^ab")
```

```
## [1] "^ab"
```

```
string
```

```
## [1] "School is fun. Especially recess. That's the best part. I love recess."
```

```
str_replace_all(".", "___", string)
```

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

```
str_replace_all("\\.", "____", string)
```

```
## [1] "\\."
```

More examples

```
set.seed(300)
d <- data.frame(rnorm(3), rbinom(3, 1, 0.5), rpois(3, 2))
d
```

```
##    rnorm.3. rbinom.3..1..0.5. rpois.3..2.
## 1 1.3737909                 1           4
## 2 0.8621069                 0           4
## 3 0.4734891                 0           1
```

```
names(d) <- str_replace_all(names(d), "\\.+", "_")
```

How could we clean this up even further by removing the underscore at the ends?

```
d
```

```
##    rnorm_3_ rbinom_3_1_0_5_ rpois_3_2_
## 1 1.3737909               1          4
## 2 0.8621069               0          4
## 3 0.4734891               0          1
```

```
names(d) <- str_replace(names(d), "_$", "")
d
```

```
##     rnorm_3 rbinom_3_1_0_5 rpois_3_2
## 1 1.3737909              1         4
## 2 0.8621069              0         4
## 3 0.4734891              0         1
```

Character classes

Search/specify entire classes of characters

```
string3 <- c("^ab", "ab", "abc", "abd", "abe", "ab 12")
str_subset(string3, "ab[cd]")
```

```
## [1] "abc" "abd"
```

```
str_subset(string3, "ab[c-e]")
```

```
## [1] "abc" "abd" "abe"
```

```r
text_num <- paste0(rep(letters[1:14], each = 7), 1:7)
text_num
```

```
##  [1] "a1" "a2" "a3" "a4" "a5" "a6" "a7" "b1" "b2" "b3" "b4" "b5" "b6" "b7"
## [15] "c1" "c2" "c3" "c4" "c5" "c6" "c7" "d1" "d2" "d3" "d4" "d5" "d6" "d7"
## [29] "e1" "e2" "e3" "e4" "e5" "e6" "e7" "f1" "f2" "f3" "f4" "f5" "f6" "f7"
## [43] "g1" "g2" "g3" "g4" "g5" "g6" "g7" "h1" "h2" "h3" "h4" "h5" "h6" "h7"
## [57] "i1" "i2" "i3" "i4" "i5" "i6" "i7" "j1" "j2" "j3" "j4" "j5" "j6" "j7"
## [71] "k1" "k2" "k3" "k4" "k5" "k6" "k7" "l1" "l2" "l3" "l4" "l5" "l6" "l7"
## [85] "m1" "m2" "m3" "m4" "m5" "m6" "m7" "n1" "n2" "n3" "n4" "n5" "n6" "n7"
```

```r
str_subset(text_num, "[a-e][3-5]")
```

```
##  [1] "a3" "a4" "a5" "b3" "b4" "b5" "c3" "c4" "c5" "d3" "d4" "d5" "e3" "e4"
## [15] "e5"
```

Invert the character list with `^`.

```
str_subset(text_num, "[^a-e][3-5]")
```

```
##  [1] "f3" "f4" "f5" "g3" "g4" "g5" "h3" "h4" "h5" "i3" "i4" "i5" "j3" "j4"
## [15] "j5" "k3" "k4" "k5" "l3" "l4" "l5" "m3" "m4" "m5" "n3" "n4" "n5"
```

Build up compound expressions

- California license plates: Example from Espenoza
    - start with a number, followed by three letters, followed by three numbers

```
## [1] "^[0-9][A-Z]{3}[0-9]{3}$"
```

POSIX character classes

CODE	DESCRIPTION	EQUIVALENT CODE
[:digit:]	Digits, 0-9	[0-9]
[:lower:]	Lower case letters	[a-z]
[:upper:]	Upper case letters	[A-Z]
[:alpha:]	Alphabetic characters	[[:lower:][:upper:]] or [A-z]
[:alnum:]	Alphanumeric characters	[[:alpha:][:digit:]] or [A-z0-9]
[:xdigit:]	Base 16 hexadecimal	[0-9A-Fa-f]
[:blank:]	Blank characters (space and tab)	
[:space:]	Space characters (all)	
[:punct:]	Punctuation characters	
[:graph:]	Human readable characters	[[:alnum:][:punct:]]
[:print:]	Printable characters	[[:alnum:][:punct:]\\s]
[:cntrl:]	Control characters (line breaks, etc.)	

```r
string4 <- c("ab", "ab12", "Dc a", "BB", "Here's some text.", "Here's a \n new line.")
writeLines(string4)
```

```
## ab
## ab12
## Dc a
## BB
## Here's some text.
## Here's a
##  new line.
```

```r
str_subset(string4, "[[:digit:]]")
```

```
## [1] "ab12"
```

```
str_subset(string4, "[[:lower:]]")
```

```
## [1] "ab"                    "ab12"                  "Dc a"
## [4] "Here's some text."     "Here's a \n new line."
```

```
str_subset(string4, "[[:upper:]]")
```

```
## [1] "Dc a"                  "BB"                    "Here's some text."
## [4] "Here's a \n new line."
```

```
str_subset(string4, "[[:space:]]")
```

```
## [1] "Dc a"                  "Here's some text."     "Here's a \n new line."
```

```
str_subset(string4, "[[:punct:]]")
```

```
## [1] "Here's some text."     "Here's a \n new line."
```

```
str_split(sentences, "[[:space:]]")
```

```
## [[1]]
## [1] "The"     "birch"   "canoe"   "slid"    "on"      "the"     "smooth"
## [8] "planks."
##
## [[2]]
## [1] "Glue"        "the"         "sheet"       "to"          "the"
## [6] "dark"        "blue"        "background."
##
## [[3]]
## [1] "It's"  "easy"  "to"    "tell"  "the"   "depth" "of"    "a"     "well."
##
## [[4]]
```

```
str_split(sentences, "[[:space:][:punct:]]")
```

```
## [[1]]
## [1] "The"    "birch"  "canoe"  "slid"   "on"     "the"    "smooth" "planks"
## [9] ""
##
## [[2]]
## [1] "Glue"       "the"        "sheet"      "to"         "the"
## [6] "dark"       "blue"       "background" ""
##
## [[3]]
##  [1] "It"    "s"     "easy"  "to"    "tell"  "the"   "depth" "of"
##  [9] "a"     "well"  ""
##
## [[4]]
##  [1] "These"   "days"    "a"       "chicken" "leg"     "is"      "a"
##  [8] "rare"    "dish"    ""
##
## [[5]]
## [1] "Rice"   "is"     "often"  "served" "in"     "round"  "bowls"  ""
##
## [[6]]
## [1] "The"    "juice"  "of"     "lemons" "makes"  "fine"   "punch"  ""
```

Things we still haven't covered

- A bunch... Many other options, etc. to control text searching
- *stringr* has a `boundary` function that can be very helpful (separating word/sentencen/paragraph boundries)
- Text mining is a world unto its own.

Examples taken from https://rstudio-pubs-static.s3.amazonaws.com/74603_76cd14d5983f47408fdf0b323550b846.html

Example with real data

Gapminder data

```
install.packages("gapminder")
library(gapminder)
data(gapminder)
head(gapminder)
```

```
## # A tibble: 6 x 6
##       country continent  year lifeExp      pop gdpPercap
##        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
## 2 Afghanistan      Asia  1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia  1962  31.997 10267083  853.1007
## 4 Afghanistan      Asia  1967  34.020 11537966  836.1971
## 5 Afghanistan      Asia  1972  36.088 13079460  739.9811
## 6 Afghanistan      Asia  1977  38.438 14880372  786.1134
```

Searching countries

Find all countries that end with *land*, and have an "i" or "t" in them.

```
str_subset(gapminder$country, "(.*[it].*)land$")
```

```
##  [1] "Finland"     "Finland"     "Finland"     "Finland"     "Finland"
##  [6] "Finland"     "Finland"     "Finland"     "Finland"     "Finland"
## [11] "Finland"     "Finland"     "Swaziland"   "Swaziland"   "Swaziland"
## [16] "Swaziland"   "Swaziland"   "Swaziland"   "Swaziland"   "Swaziland"
## [21] "Swaziland"   "Swaziland"   "Swaziland"   "Swaziland"   "Switzerland"
## [26] "Switzerland" "Switzerland" "Switzerland" "Switzerland" "Switzerland"
## [31] "Switzerland" "Switzerland" "Switzerland" "Switzerland" "Switzerland"
## [36] "Switzerland" "Thailand"    "Thailand"    "Thailand"    "Thailand"
## [41] "Thailand"    "Thailand"    "Thailand"    "Thailand"    "Thailand"
## [46] "Thailand"    "Thailand"    "Thailand"
```

Backreferencing

Replace *land* for *LAND* for countries with an "i" or "t" in them

```
gapminder <- gapminder %>%
    mutate(country = str_replace_all(country, "(.*[it].*)land$", "\\1LAND"))

str_subset(gapminder$country, "LAND")
```

```
##  [1] "FinLAND"     "FinLAND"     "FinLAND"     "FinLAND"     "FinLAND"
##  [6] "FinLAND"     "FinLAND"     "FinLAND"     "FinLAND"     "FinLAND"
## [11] "FinLAND"     "FinLAND"     "SwaziLAND"   "SwaziLAND"   "SwaziLAND"
## [16] "SwaziLAND"   "SwaziLAND"   "SwaziLAND"   "SwaziLAND"   "SwaziLAND"
## [21] "SwaziLAND"   "SwaziLAND"   "SwaziLAND"   "SwaziLAND"   "SwitzerLAND"
## [26] "SwitzerLAND" "SwitzerLAND" "SwitzerLAND" "SwitzerLAND" "SwitzerLAND"
## [31] "SwitzerLAND" "SwitzerLAND" "SwitzerLAND" "SwitzerLAND" "SwitzerLAND"
## [36] "SwitzerLAND" "ThaiLAND"    "ThaiLAND"    "ThaiLAND"    "ThaiLAND"
## [41] "ThaiLAND"    "ThaiLAND"    "ThaiLAND"    "ThaiLAND"    "ThaiLAND"
## [46] "ThaiLAND"    "ThaiLAND"    "ThaiLAND"
```