Some data taming

Daniel Anderson
Week 8, Class 1

Agenda

- Discuss variable types
- "Automatic" factor re-leveling
- Maybe some string manipulations (if time allows)
- Lab

Agenda

- Discuss variable types
- "Automatic" factor re-leveling
- Maybe some string manipulations (if time allows)
- Lab

Learning objectives for today

- Get more comfortable with tidy data
- Get more fluent with `gather`

What questions do you have?

Disclaimer

Today we'll be covering the very basics

Creating factors

- Imagine you have a vector of months

```
months <- c("Dec", "Apr", "Jan", "Mar")
```

Creating factors

- Imagine you have a vector of months

```
months <- c("Dec", "Apr", "Jan", "Mar")
```

- We could store this as a string, but there are issues with this.
  - There are only 12 possible months

Creating factors

- Imagine you have a vector of months

```
months <- c("Dec", "Apr", "Jan", "Mar")
```

- We could store this as a string, but there are issues with this.
  - There are only 12 possible months
    - factors will help us weed out values that don't conform to our predefined *levels*, which helps safeguard against typos, etc.

Creating factors

- Imagine you have a vector of months

```
months <- c("Dec", "Apr", "Jan", "Mar")
```

- We could store this as a string, but there are issues with this.
  - There are only 12 possible months
    - factors will help us weed out values that don't conform to our predefined *levels*, which helps safeguard against typos, etc.
  - You can't sort this vector in a meaningful way (it defaults to alphabetic)

Creating factors

- Imagine you have a vector of months

```
months <- c("Dec", "Apr", "Jan", "Mar")
```

- We could store this as a string, but there are issues with this.
  - There are only 12 possible months
    - factors will help us weed out values that don't conform to our predefined *levels*, which helps safeguard against typos, etc.
  - You can't sort this vector in a meaningful way (it defaults to alphabetic)

```
sort(months)
```

```
## [1] "Apr" "Dec" "Jan" "Mar"
```

Define it as a factor

```
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")

months <- factor(months, levels = month_levels)
months
```

```
## [1] Dec Apr Jan Mar
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Define it as a factor

```
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")

months <- factor(months, levels = month_levels)
months
```

```
## [1] Dec Apr Jan Mar
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

- Now, we can sort

```
sort(months)
```

```
## [1] Jan Mar Apr Dec
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Provides a safety net of sorts

```
months[5] <- "Jam"
```

```
## Warning in `[<-.factor`(`*tmp*`, 5, value = "Jam"): invalid factor level,
## NA generated
```

Provides a safety net of sorts

```
months[5] <- "Jam"
```

```
## Warning in `[<-.factor`(`*tmp*`, 5, value = "Jam"): invalid factor level,
## NA generated
```

```
months
```

```
## [1] Dec  Apr  Jan  Mar  <NA>
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

What if we don't specify the levels?

- If you define a factor without specifying the levels, it will assign them alphabetically

What if we don't specify the levels?

- If you define a factor without specifying the levels, it will assign them alphabetically

```
mnths <- c("Dec", "Apr", "Jan", "Mar")
factor(mnths)
```

```
## [1] Dec Apr Jan Mar
## Levels: Apr Dec Jan Mar
```

Accessing and modifying levels

- Use the `levels` function

```
levels(months)
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
## [12] "Dec"
```

Accessing and modifying levels

- Use the `levels` function

```
levels(months)
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
## [12] "Dec"
```

- Modify levels the base way

```
levels(months) <- 1:12
months
```

```
## [1] 12   4    1    3    <NA>
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12
```

If you need to, be specific

```
months <- factor(months, levels = 1:12, labels = month_levels)
months
```

```
## [1] Dec  Apr  Jan  Mar  <NA>
## Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

New package

- When working with factors, we can use the *forcats* package
  - *for* **cat** egorical variables *s*
  - anagram for factors
- Part of the tidyverse, so should be good to go

Change factors

Back to the order they appeared

```
c("Dec", "Apr", "Jan", "Mar") %>%
    factor()
```

```
## [1] Dec Apr Jan Mar
## Levels: Apr Dec Jan Mar
```

```
c("Dec", "Apr", "Jan", "Mar") %>%
    factor(levels = c("Jan", "Mar", "Apr", "Dec"))
```

```
## [1] Dec Apr Jan Mar
## Levels: Jan Mar Apr Dec
```

... see next slide

```r
library(forcats)
c("Dec", "Apr", "Jan", "Mar") %>%
    factor(levels = c("Jan", "Mar", "Apr", "Dec")) %>%
    fct_inorder()
```

```
## [1] Dec Apr Jan Mar
## Levels: Dec Apr Jan Mar
```

Or order by frequency

```
c("b", "b", "c", "a", "a", "a") %>%
    fct_infreq()
```

```
## [1] b b c a a a
## Levels: a b c
```

- This can be particularly useful for plotting

Words example

```
data(sentences, package = "stringr")
sentences <- tibble(sent_num = seq_along(sentences),
                    sentence = sentences)

sentences
```

```
## # A tibble: 720 x 2
##    sent_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
```

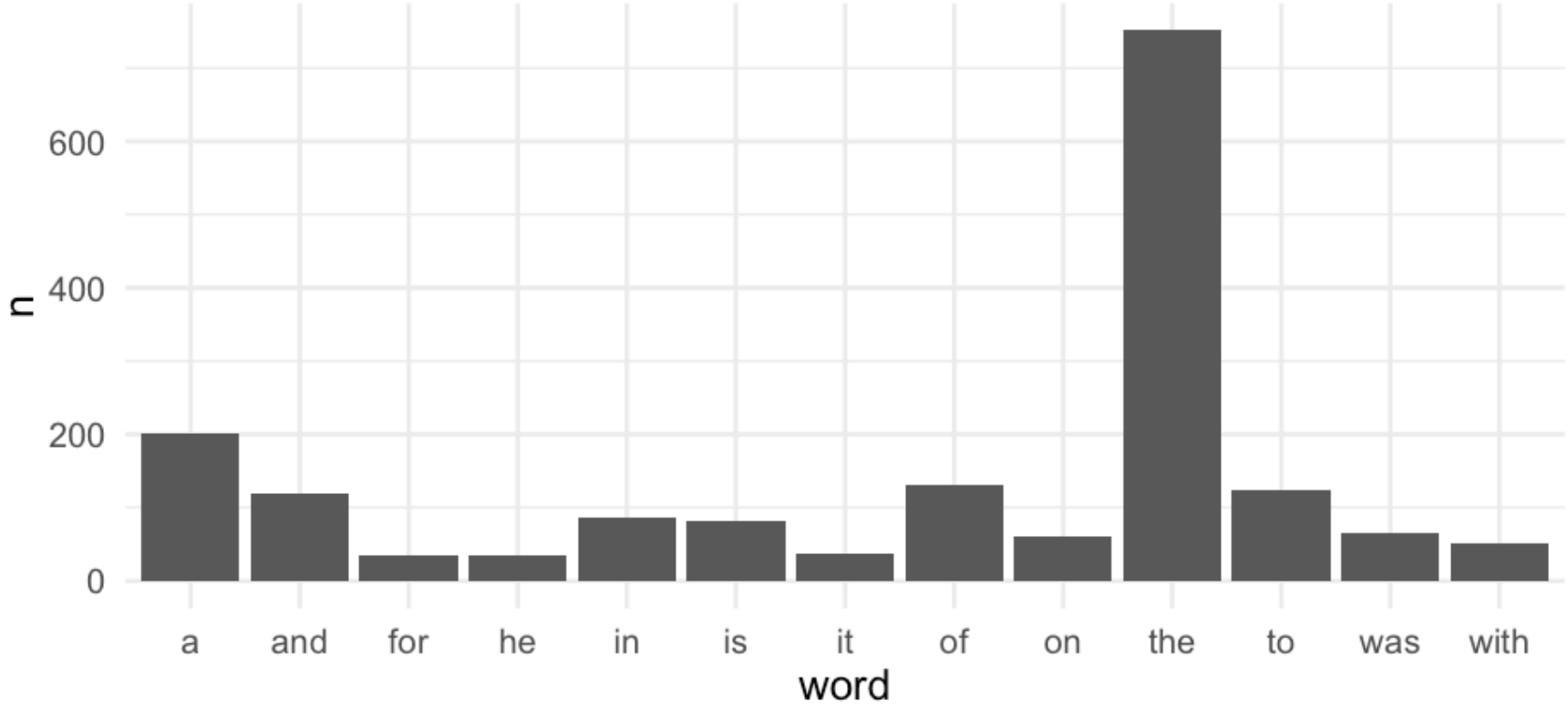
Word frequency

```
library(tidytext)

words_freq <- sentences %>%
  unnest_tokens(word, sentence) %>%
  count(word) %>%
  filter(n > 30)

head(words_freq)
```

```
## # A tibble: 6 x 2
##   word      n
##   <chr> <int>
## 1 a       202
## 2 and     118
## 3 for      35
## 4 he       34
## 5 in       87
## 6 is       81
```

Try to plot frequencies

```
ggplot(words_freq, aes(word, n)) +
    geom_col()
```



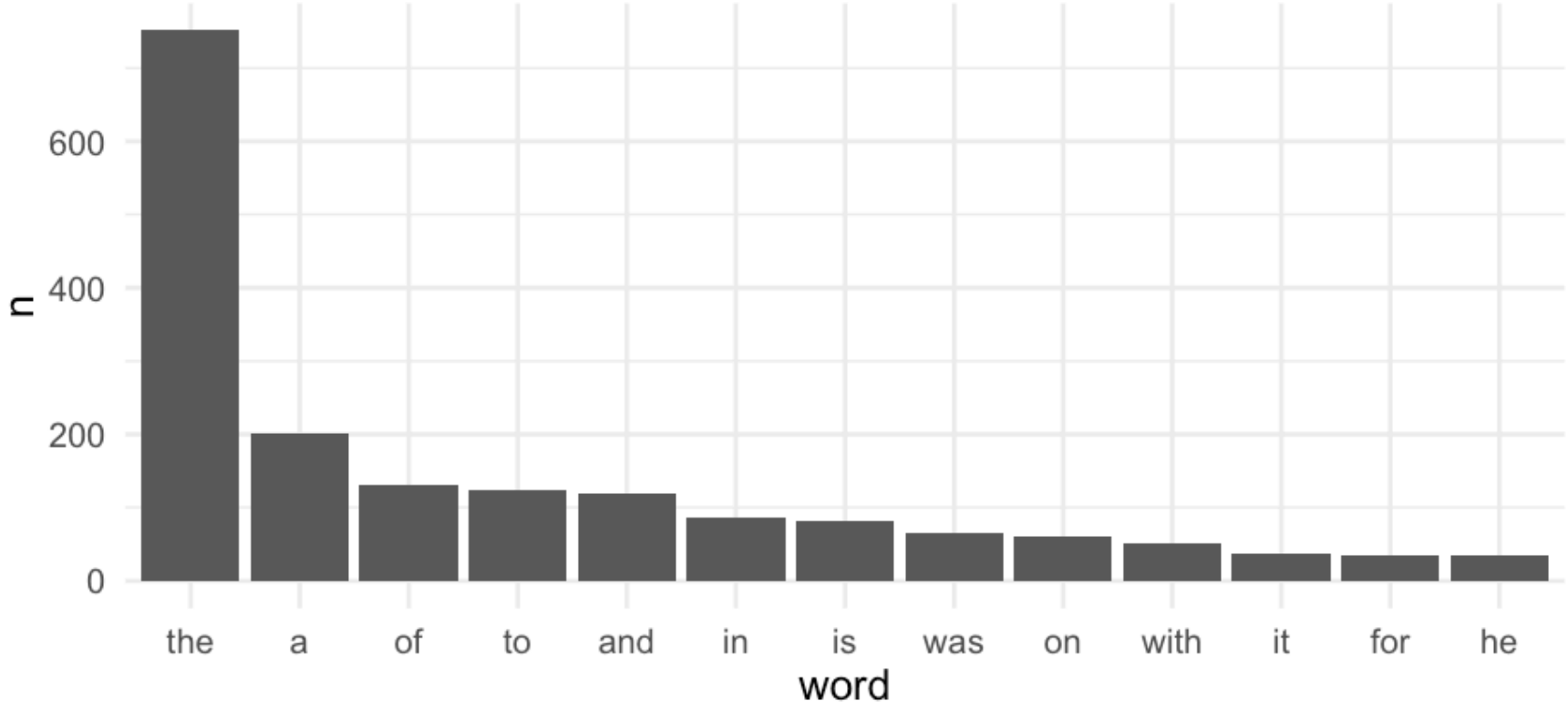
Reorder according to frequency

```
words_freq2 <- sentences %>%
    unnest_tokens(word, sentence) %>%
mutate(word = fct_infreq(word)) %>%
    count(word) %>%
    filter(n > 30)

head(words_freq2)
```

```
## # A tibble: 6 x 2
##   word      n
##   <fct> <int>
## 1 the     751
## 2 a       202
## 3 of      132
## 4 to      123
## 5 and     118
## 6 in       87
```

Reproduce plot

```
ggplot(words_freq2, aes(word, n)) +
    geom_col()
```



Looking at the levels

```
levels(factor(words_freq$word))
```

```
##  [1] "a"    "and"  "for"  "he"   "in"   "is"   "it"   "of"   "on"   "the"
## [11] "to"   "was"  "with"
```

```
levels(words_freq2$word) %>%
  head()
```

```
## [1] "the" "a"   "of"  "to"  "and" "in"
```

When do we really want factors?

Generally two reasons to declare a factor

- Only finite number of categories
  - Treatment/control
  - Income categories
  - Performance levels
  - etc.
- Use in modeling

GSS

General Social Survey

```
gss_cat
```

```
## # A tibble: 21,483 x 9
##     year marital         age race  rincome  partyid  relig  denom  tvhours
##    <int> <fct>         <int> <fct> <fct>    <fct>    <fct>  <fct>    <int>
##  1  2000 Never married    26 White $8000 t… Ind,nea… Prote… South…       12
##  2  2000 Divorced         48 White $8000 t… Not str… Prote… Bapti…       NA
##  3  2000 Widowed          67 White Not app… Indepen… Prote… No de…        2
##  4  2000 Never married    39 White Not app… Ind,nea… Ortho… Not a…        4
##  5  2000 Divorced         25 White Not app… Not str… None   Not a…        1
##  6  2000 Married          25 White $20000 … Strong … Prote… South…       NA
##  7  2000 Never married    36 White $25000 … Not str… Chris… Not a…        3
##  8  2000 Divorced         44 White $7000 t… Ind,nea… Prote… Luthe…       NA
##  9  2000 Married          44 White $25000 … Not str… Prote… Other         0
## 10  2000 Married          47 White $25000 … Strong … Prote… South…        3
## # ... with 21,473 more rows
```

Investigate factors

Tidyverse gives you convenient ways to evaluate factors

- Use `count`
- Use `geom_bar` or `geom_col` with *ggplot*

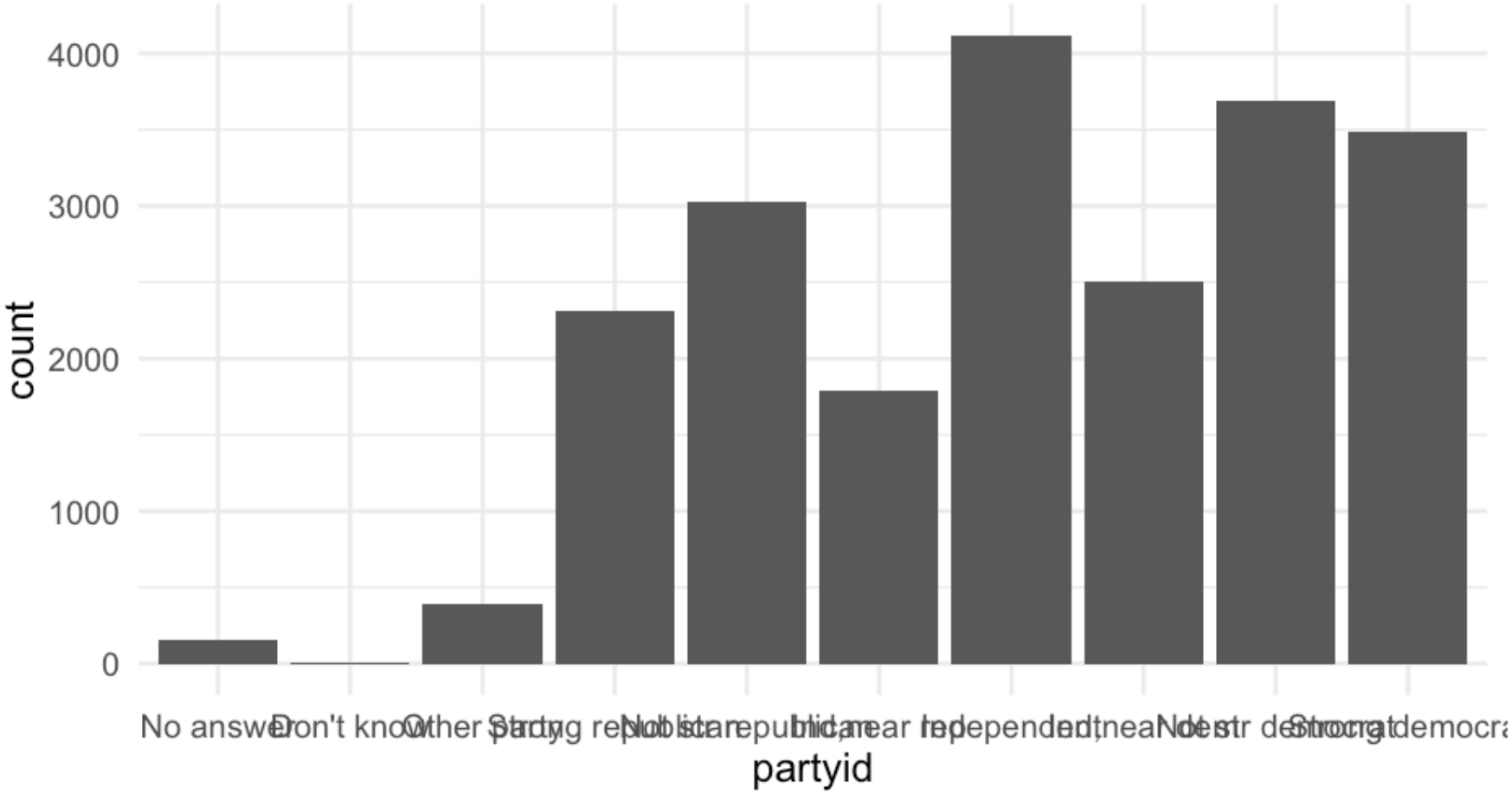
```
gss_cat %>%
    count(partyid)
```

```
## # A tibble: 10 x 2
##    partyid              n
##    <fct>            <int>
##  1 No answer          154
##  2 Don't know           1
##  3 Other party        393
##  4 Strong republican 2314
##  5 Not str republican 3032
##  6 Ind,near rep       1791
##  7 Independent        4119
##  8 Ind,near dem       2499
##  9 Not str democrat   3690
## 10 Strong democrat    3490
```

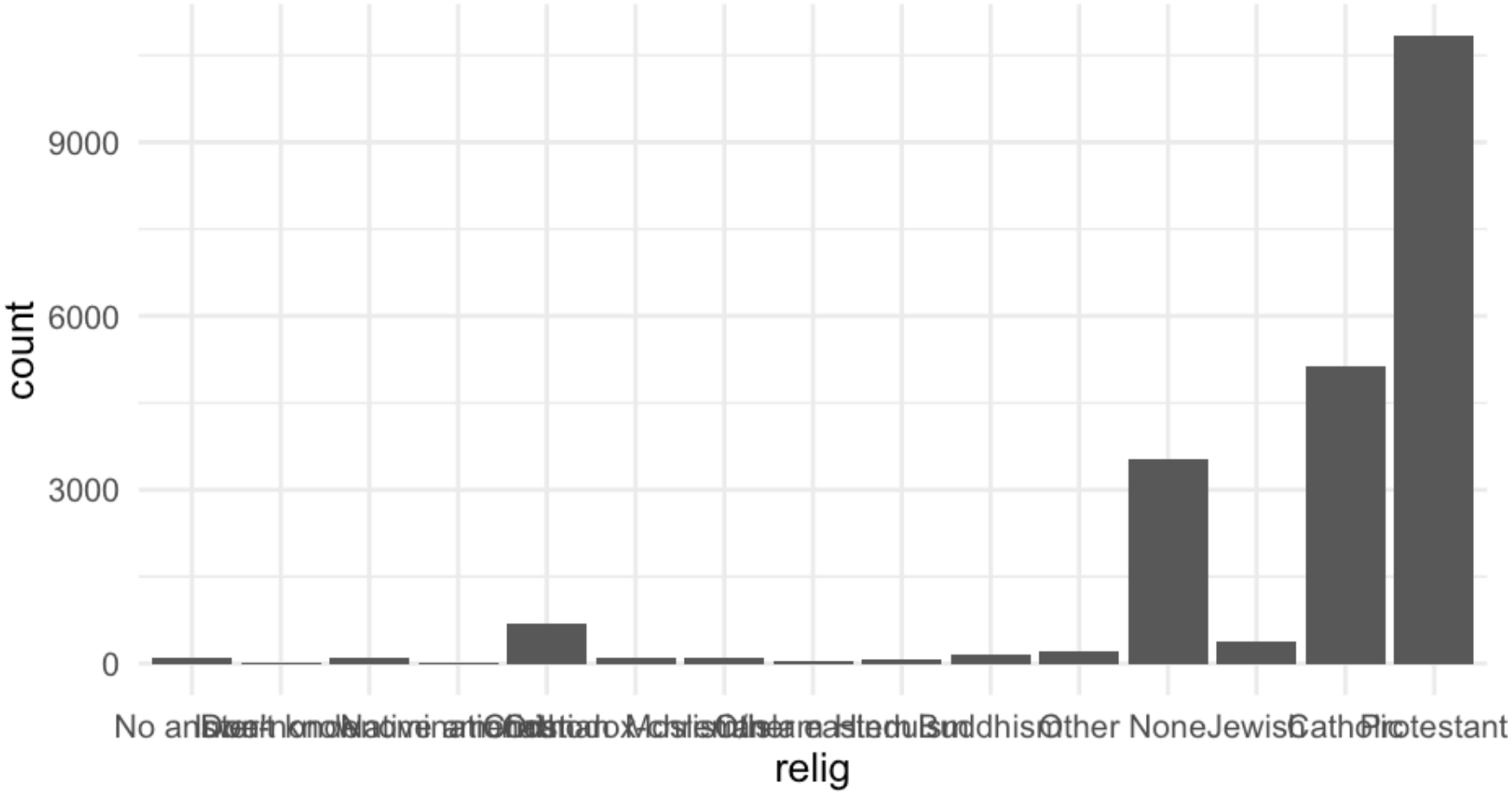
```
gss_cat %>%
    count(relig)
```

```
## # A tibble: 15 x 2
##    relig                      n
##    <fct>                  <int>
##  1 No answer                 93
##  2 Don't know                15
##  3 Inter-nondenominational  109
##  4 Native american           23
##  5 Christian                689
##  6 Orthodox-christian        95
##  7 Moslem/islam             104
##  8 Other eastern             32
##  9 Hinduism                  71
## 10 Buddhism                 147
## 11 Other                    224
## 12 None                    3523
## 13 Jewish                   388
## 14 Catholic                5124
## 15 Protestant             10846
```

```
ggplot(gss_cat, aes(partyid)) +
    geom_bar()
```

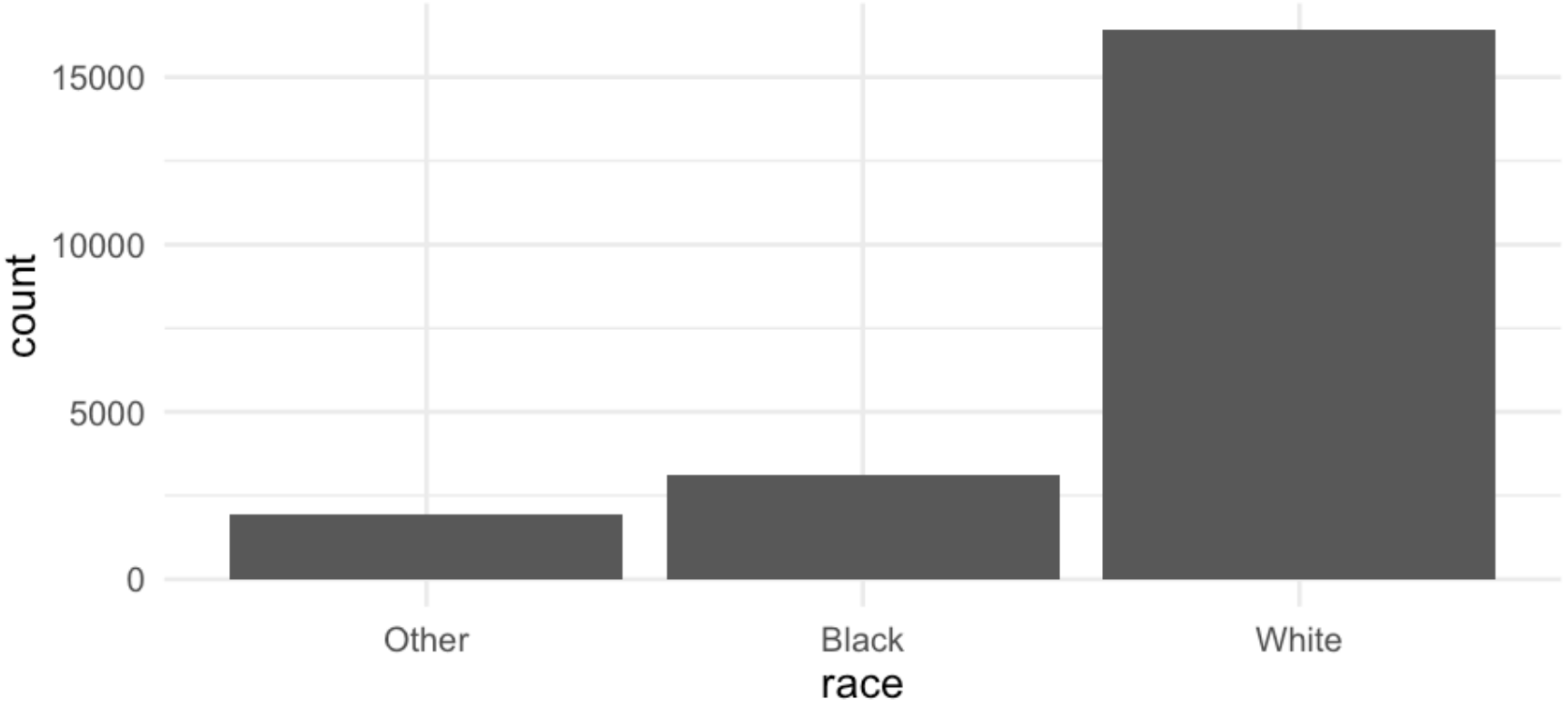


```
ggplot(gss_cat, aes(relig)) +
    geom_bar()
```



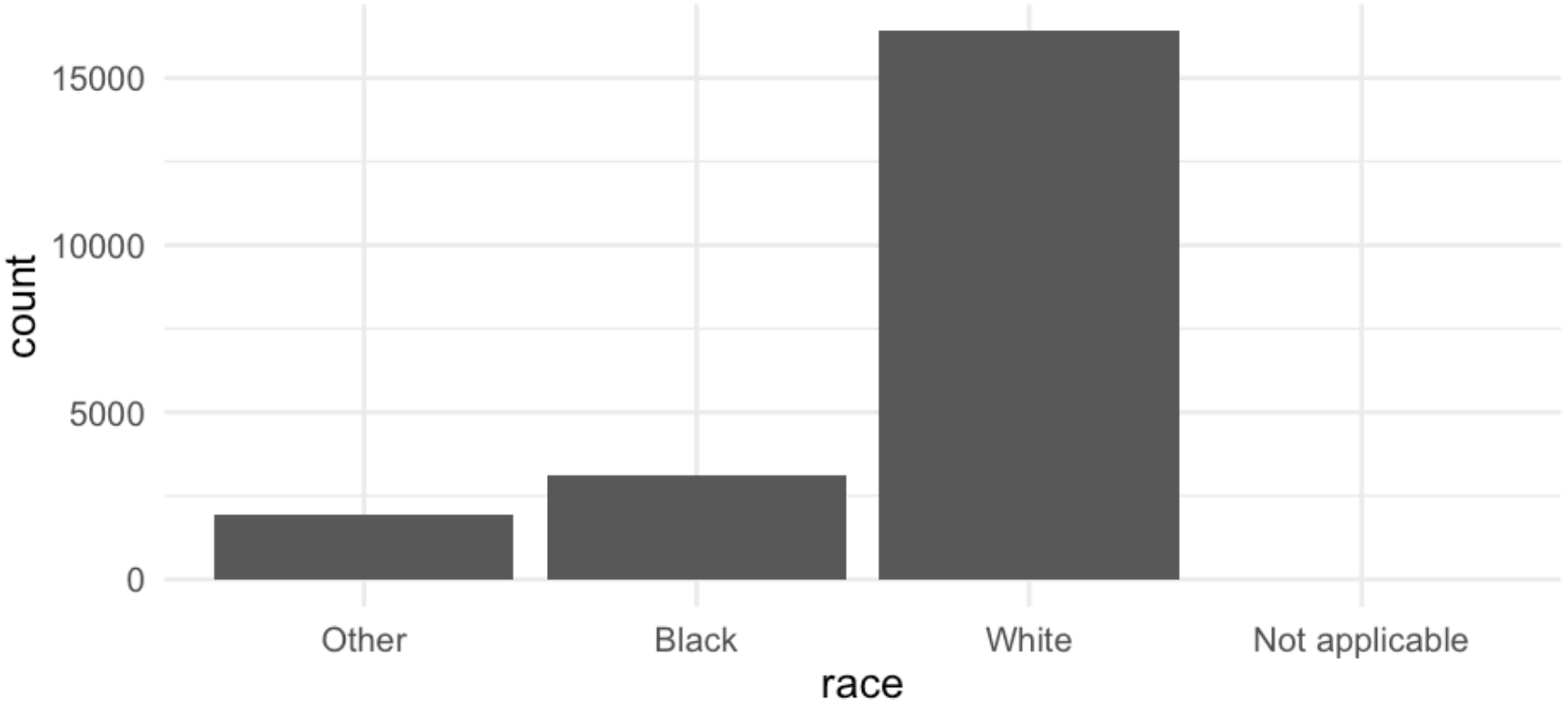
Missing categories

```
ggplot(gss_cat, aes(race)) +
    geom_bar()
```



Include missing

```
ggplot(gss_cat, aes(race)) +
    geom_bar() +
    scale_x_discrete(drop = FALSE)
```



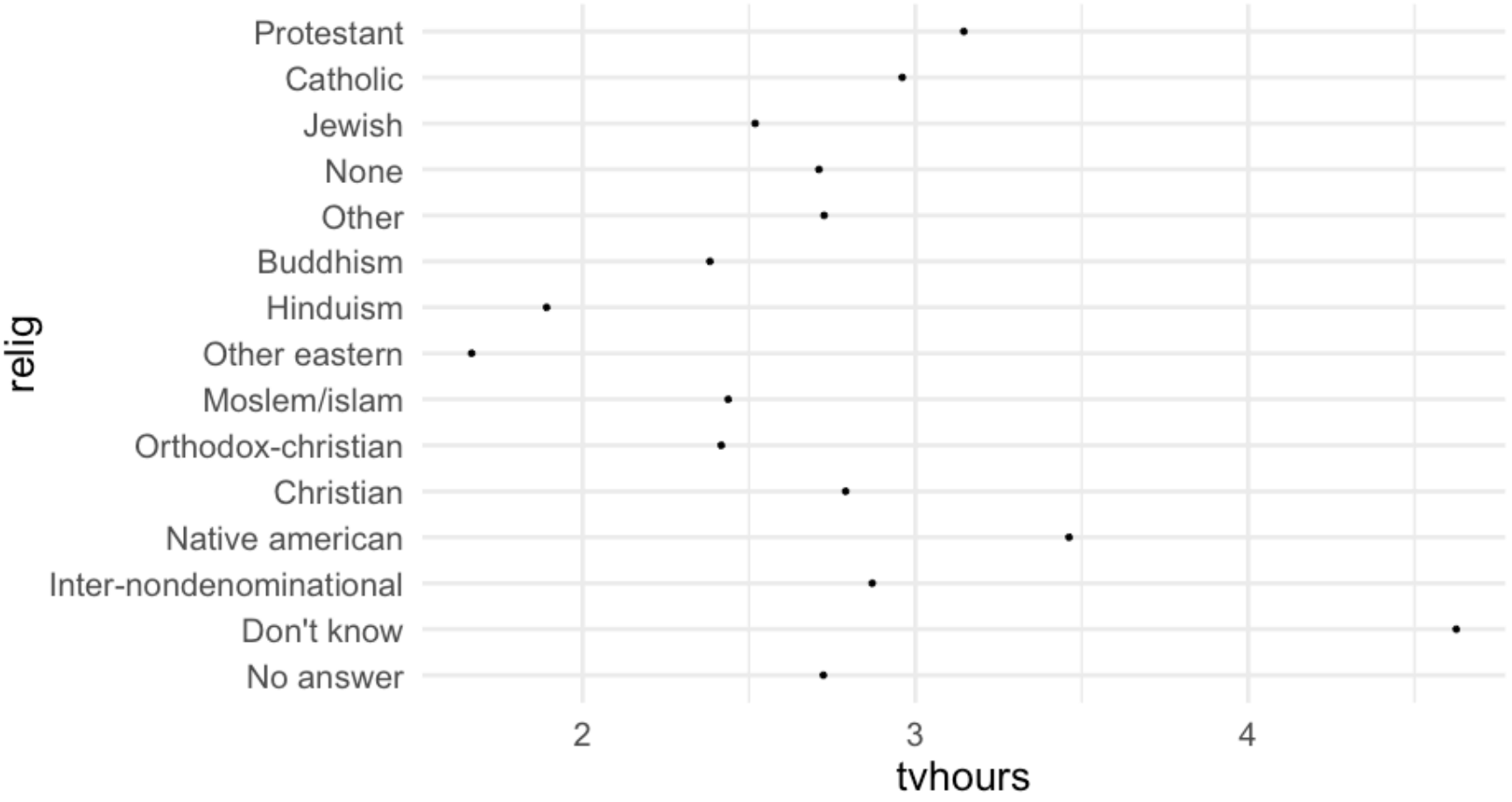
Reorder factors

- Let's bring some (more) automation to the game
- `forcats::fct_reorder` allows you to easily reorder factors according to another variable

```
relig_summary <- gss_cat %>%
  group_by(relig) %>%
  summarise(tvhours = mean(tvhours, na.rm = TRUE),
                n = n())

head(relig_summary)
```

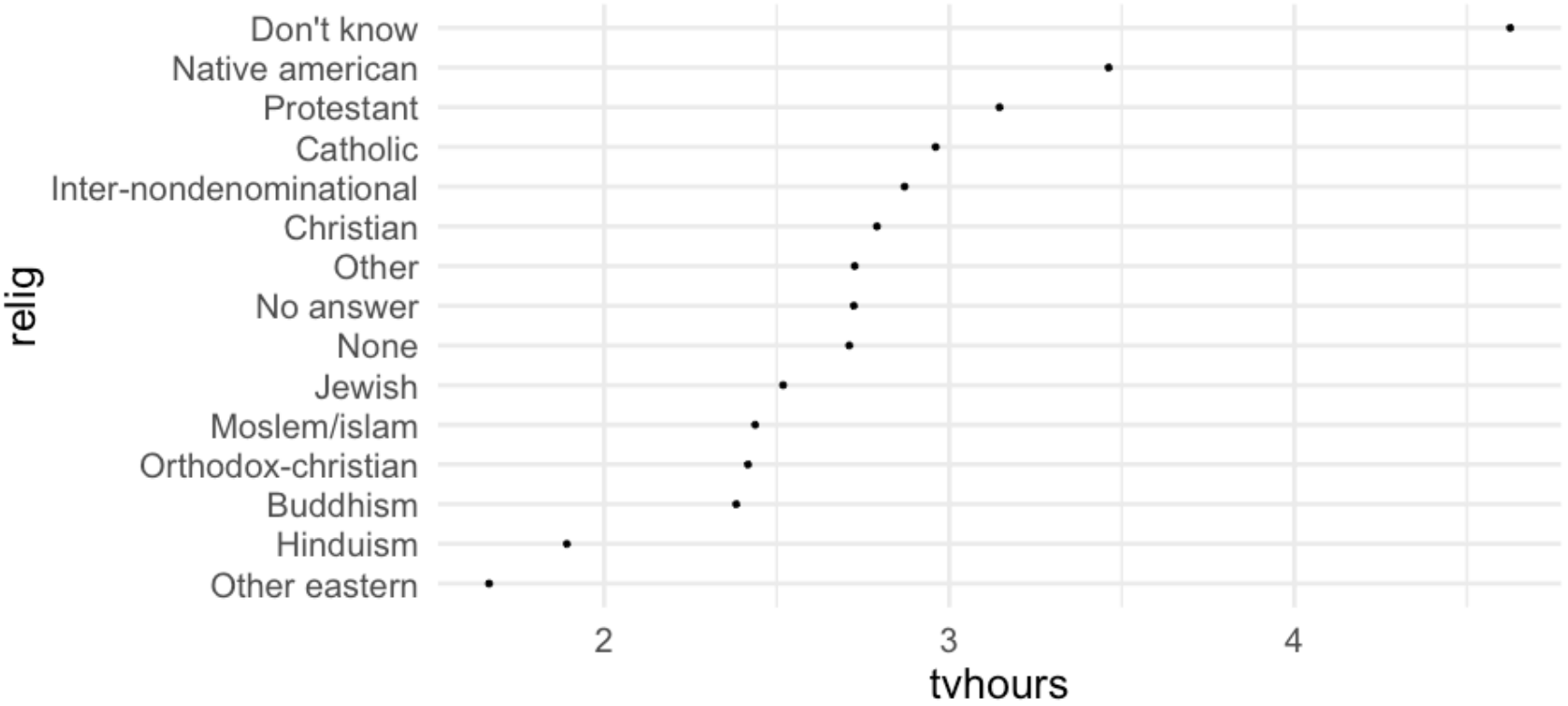
```
## # A tibble: 6 x 3
##   relig                   tvhours     n
##   <fct>                     <dbl> <int>
## 1 No answer                2.72     93
## 2 Don't know               4.62     15
## 3 Inter-nondenominational  2.87    109
## 4 Native american          3.46     23
## 5 Christian                2.79    689
## 6 Orthodox-christian       2.42     95
```

```
ggplot(relig_summary, aes(tvhours, relig)) +
  geom_point()
```



Note - you could actually include the factor reorder right within the `ggplot` call.

```
relig_summary <- relig_summary %>%
    mutate(relig = fct_reorder(relig, tvhours))

ggplot(relig_summary, aes(tvhours, relig)) +
  geom_point()
```



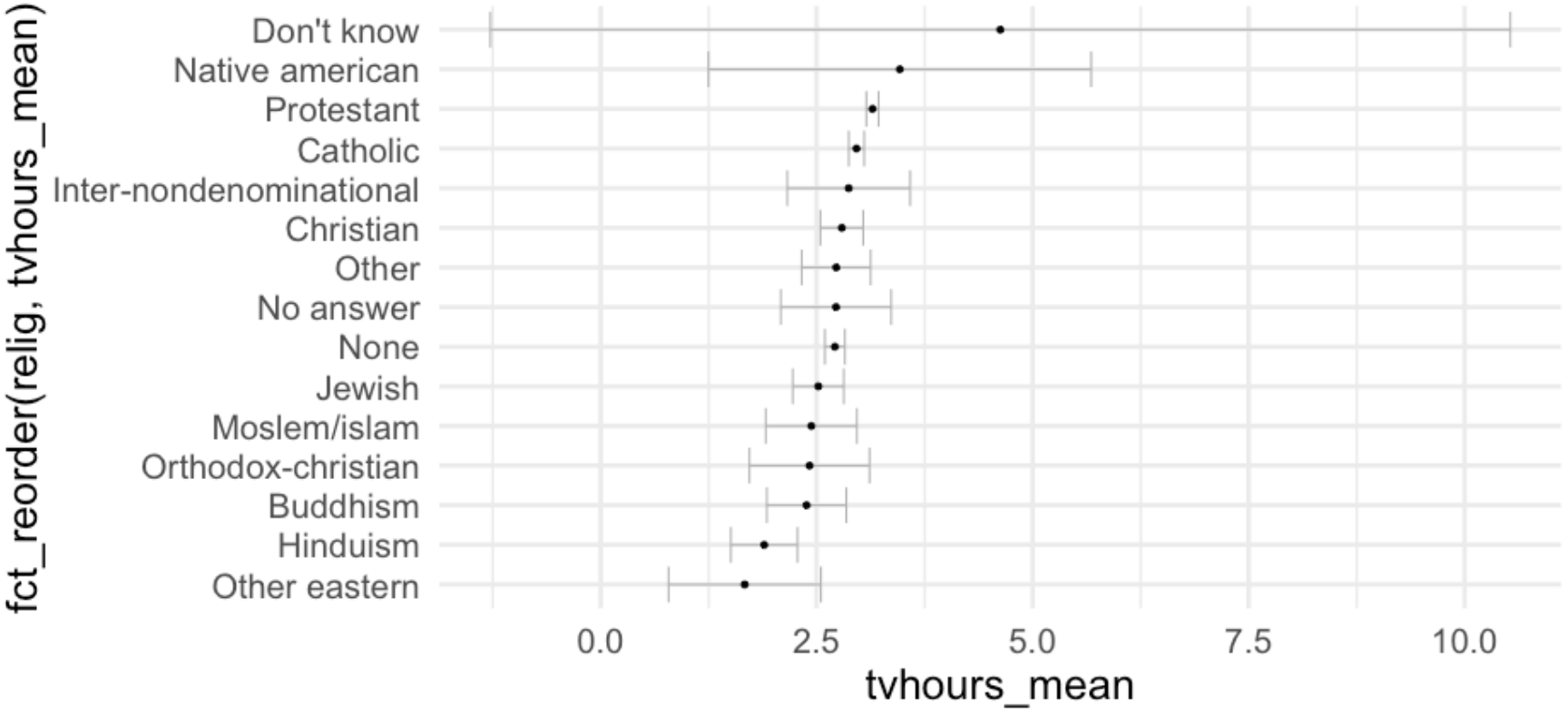
Quick aside

What if we wanted error bars?

```
relig_summary2 <- gss_cat %>%
  group_by(relig) %>%
  summarise(tvhours_mean = mean(tvhours, na.rm = TRUE),
            tvhours_se   = sqrt(var(tvhours, na.rm = TRUE) /
                                  length(na.omit(tvhours))),
            n = n())

head(relig_summary2)
```

```
## # A tibble: 6 x 4
##   relig                  tvhours_mean tvhours_se     n
##   <fct>                         <dbl>      <dbl> <int>
## 1 No answer                  2.723404  0.3257413    93
## 2 Don't know                 4.625     3.011511     15
## 3 Inter-nondenominational    2.870370  0.3625038   109
## 4 Native american            3.461538  1.130097     23
## 5 Christian                  2.790281  0.1264124   689
## 6 Orthodox-christian         2.416667  0.3549536    95
```

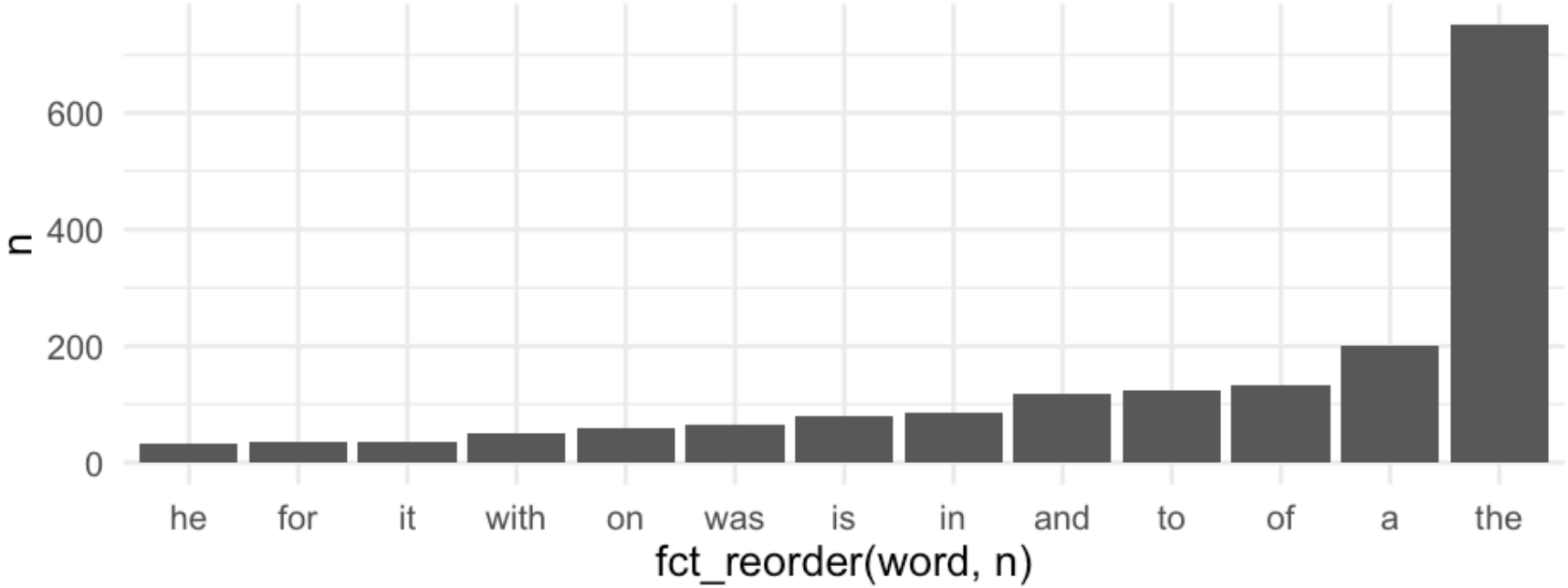
```
ggplot(relig_summary2, aes(tvhours_mean,
                           fct_reorder(relig, tvhours_mean))) +
  geom_errorbarh(aes(xmin = tvhours_mean - 1.96*tvhours_se,
                     xmax = tvhours_mean + 1.96*tvhours_se),
                 color = "gray70") +
  geom_point()
```



Revisiting our word frequency example

- An easier way to do what we did before, would be to just include the reorder call right within the call to ggplot

```
ggplot(words_freq, aes(fct_reorder(word, n), n)) +
    geom_col()
```



More on modifying factor levels

- `forcats::fct_recode` can make modifying factors more explicit

```
gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong" = "Strong republican",
    "Republican, weak" = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak" = "Not str democrat",
    "Democrat, strong" = "Strong democrat")) %>%
  count(partyid)
```

```
## # A tibble: 10 x 2
##    partyid                    n
##    <fct>                  <int>
##  1 No answer                154
##  2 Don't know                 1
##  3 Other party              393
##  4 Republican, strong      2314
##  5 Republican, weak        3032
##  6 Independent, near rep   1791
##  7 Independent             4119
##  8 Independent, near dem   2499
##  9 Democrat, weak          3690
## 10 Democrat, strong        3490
```

Collapsing levels

- `fct_recode` can also be used to collapse levels easily

```
gss_cat %>%
  mutate(partyid = fct_recode(partyid,
    "Republican, strong"    = "Strong republican",
    "Republican, weak"      = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak"        = "Not str democrat",
    "Democrat, strong"      = "Strong democrat",
    "Other"                 = "No answer",
    "Other"                 = "Don't know",
    "Other"                 = "Other party")) %>%
  count(partyid)
```

```
## # A tibble: 8 x 2
##   partyid                   n
##   <fct>                 <int>
## 1 Other                   548
## 2 Republican, strong     2314
## 3 Republican, weak       3032
## 4 Independent, near rep  1791
## 5 Independent            4119
## 6 Independent, near dem  2499
## 7 Democrat, weak         3690
## 8 Democrat, strong       3490
```

Collapse a lot of categories

- One of the more useful functions in *forcats*
- Collapse all categories into *republican*, *democrat*, *independent*, or *other*.

```
gss_cat %>%
  mutate(partyid = fct_collapse(partyid,
    other = c("No answer", "Don't know", "Other party"),
    rep = c("Strong republican", "Not str republican"),
    ind = c("Ind,near rep", "Independent", "Ind,near dem"),
    dem = c("Not str democrat", "Strong democrat")
  )) %>%
  count(partyid)
```

```
## # A tibble: 4 x 2
##   partyid     n
##   <fct>   <int>
## 1 other     548
## 2 rep      5346
## 3 ind      8409
## 4 dem      7180
```

Sometimes even better

- We can "lump" a bunch of categories together using `fct_lump`.
- Default behavior creates an "other" group that includes all but the largest group (i.e., lumps the factor into two groups).
- Can also take optional *n* argument, where *n* represents the number of groups to collapse to

```
gss_cat %>%
    mutate(rel = fct_lump(relig)) %>%
    count(rel)
```

```
## # A tibble: 2 x 2
##   rel            n
##   <fct>      <int>
## 1 Protestant 10846
## 2 Other      10637
```

Collapse to 10 religious groups

```
gss_cat %>%
    mutate(rel = fct_lump(relig, n = 10)) %>%
    count(rel)
```

```
## # A tibble: 10 x 2
##    rel                        n
##    <fct>                  <int>
##  1 Inter-nondenominational   109
##  2 Christian                 689
##  3 Orthodox-christian         95
##  4 Moslem/islam              104
##  5 Buddhism                  147
##  6 None                     3523
##  7 Jewish                    388
##  8 Catholic                 5124
##  9 Protestant              10846
## 10 Other                     458
```

Strings

Strings to process

```
library(stringr)
fruit
sentences
words
```

fruit

```
head(fruit, n = 20)
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"    "boysenberry"  "breadfruit"
## [13] "canary melon" "cantaloupe"   "cherimoya"    "cherry"
## [17] "chili pepper" "clementine"   "cloudberry"   "coconut"
```

sentences

```
sentences <- stringr::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 = 20)
```

```
##  [1] "a"         "able"      "about"     "absolute"  "accept"
##  [6] "account"   "achieve"   "across"    "act"       "active"
## [11] "actual"    "add"       "address"   "admit"     "advertise"
## [16] "affect"    "afford"    "after"     "afternoon" "again"
```

Make everything upper case

stringr

```
str_to_upper(fruit) %>%
  head()
```

```
## [1] "APPLE"       "APRICOT"     "AVOCADO"     "BANANA"      "BELL PEPPER"
## [6] "BILBERRY"
```

Make everything upper case

stringr

```
str_to_upper(fruit) %>%
  head()
```

```
## [1] "APPLE"       "APRICOT"     "AVOCADO"     "BANANA"      "BELL PEPPER"
## [6] "BILBERRY"
```

base

```
toupper(fruit) %>%
  head()
```

```
## [1] "APPLE"       "APRICOT"     "AVOCADO"     "BANANA"      "BELL PEPPER"
## [6] "BILBERRY"
```

Make everything lower case

stringr

```
str_to_lower(sentences) %>%
  head()
```

```
## [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."
```

base

```
tolower(sentences) %>%
  head()
```

```
## [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."
```

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"
```

Calculate string length

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

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

stringr

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

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

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"
```

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"
```

Trim white space

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

Trim white space

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

```
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 trim white space

```
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

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

Pad white space

```
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"
```

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"
```

Locate pattern

```
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."
## [20] "The flint sputtered and lit a pine torch."
## [21] "The old pan was covered with hard fudge."
## [22] "The store walls were lined with colored frocks."
```

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."
## [20] "The flint sputtered and lit a pine torch."
## [21] "The old pan was covered with hard fudge."
```

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 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
## [666] 1 1 0 0 0 1 2 0 1 1 2 1 0 2 1 0 0 1 1 0 1 0 1 0 0 1 2 1 1 1 0 1 1 0 1
## [701] 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 1 0 1
```

Logical tests

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

```
str_detect(sentences, "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 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [155] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [177]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [188] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [199] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Or with `grepl`

```
grepl("red", sentences)
```

```
##   [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 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [155] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [177]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [188] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [199] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [210] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE
```

Pattern matching summary

- 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.

Pattern matching summary

- 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

Pattern matching summary

- 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
    - A whole learning curve on its own - for now, it's good to know you'll have to escape certain characters.

Symbols you may need to escape

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

Lab