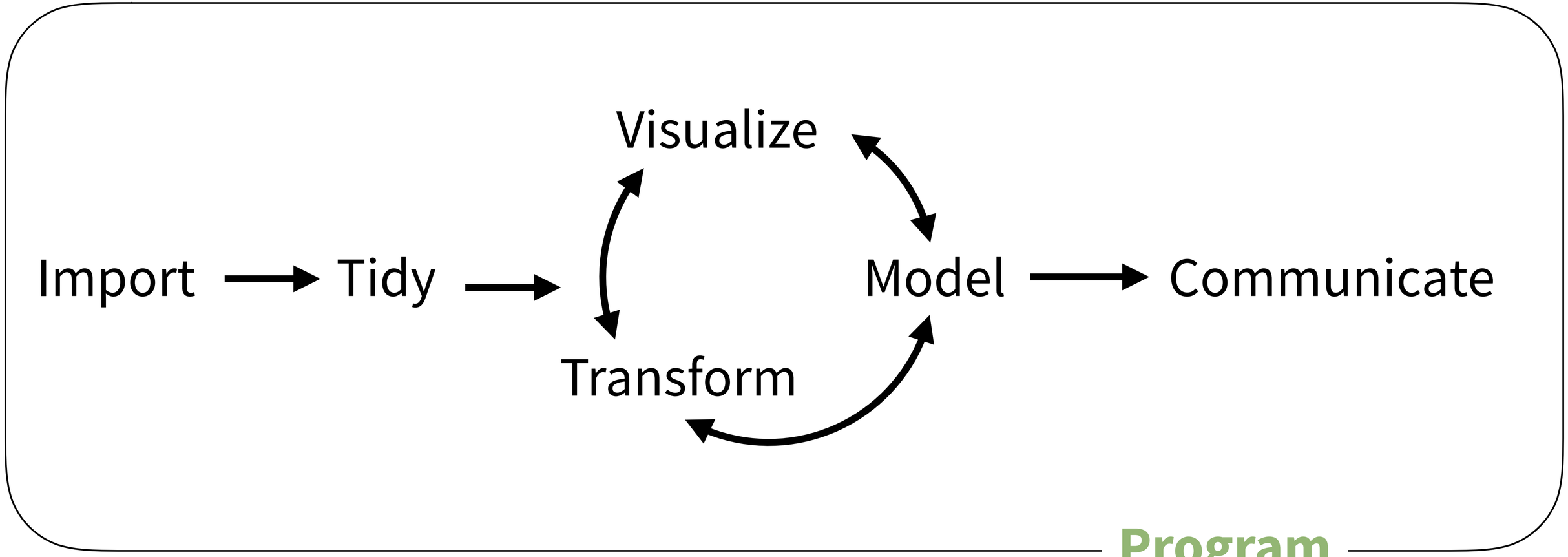
Data types with

Quiz

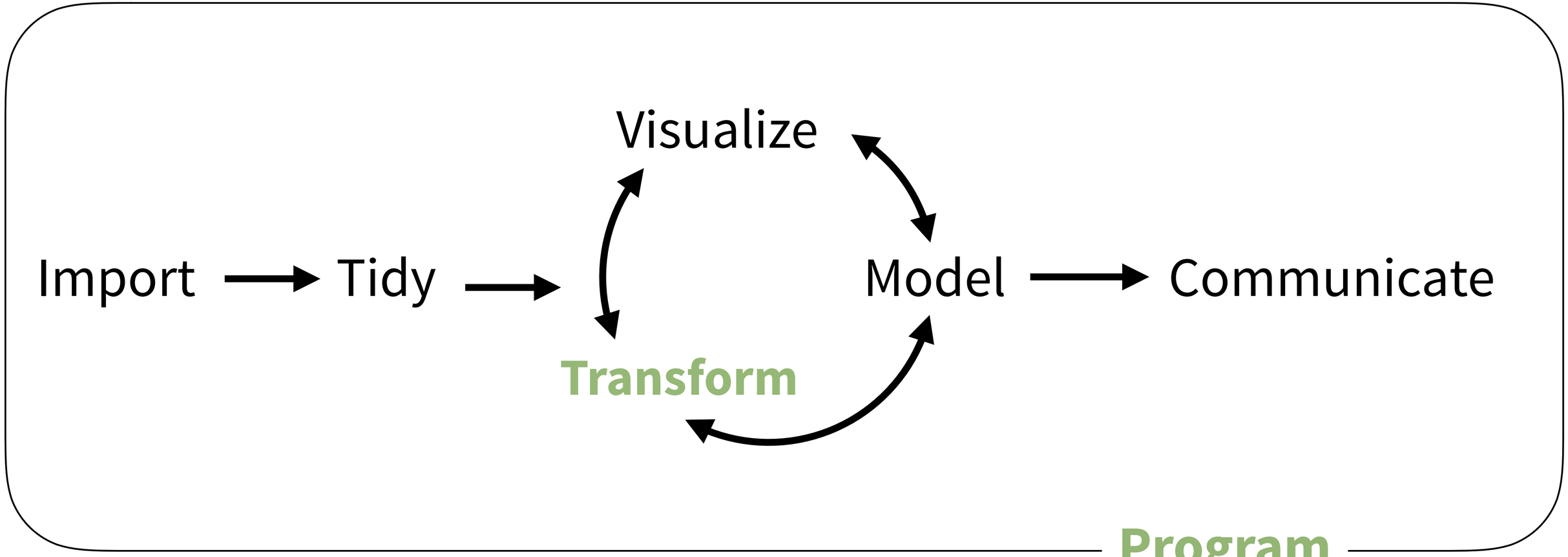
What types of data are in this data set?

	time_hour	name	air_time	distance	day	delayed
1	2013-01-01 05:00:00	United Air Lines Inc.	13620s (~3.78 hours)	1400	Tuesday	TRUE
2	2013-01-01 05:00:00	United Air Lines Inc.	13620s (~3.78 hours)	1416	Tuesday	TRUE
3	2013-01-01 05:00:00	American Airlines Inc.	9600s (~2.67 hours)	1089	Tuesday	TRUE
4	2013-01-01 05:00:00	JetBlue Airways	10980s (~3.05 hours)	1576	Tuesday	FALSE
5	2013-01-01 06:00:00	Delta Air Lines Inc.	6960s (~1.93 hours)	762	Tuesday	FALSE
6	2013-01-01 05:00:00	United Air Lines Inc.	9000s (~2.5 hours)	719	Tuesday	TRUE
7	2013-01-01 06:00:00	JetBlue Airways	9480s (~2.63 hours)	1065	Tuesday	TRUE
8	2013-01-01 06:00:00	ExpressJet Airlines Inc.	3180s (~53 minutes)	229	Tuesday	FALSE
9	2013-01-01 06:00:00	JetBlue Airways	8400s (~2.33 hours)	944	Tuesday	FALSE
10	2013-01-01 06:00:00	American Airlines Inc.	8280s (~2.3 hours)	733	Tuesday	TRUE
11	2013-01-01 06:00:00	JetBlue Airways	8940s (~2.48 hours)	1028	Tuesday	FALSE

(Applied) **Data Science**



(Applied) **Data Science**



Set up

In addition to the core tidyverse, load babynames and nycflights13

```
library(tidyverse)
library(babynames)
library(nycflights13)
```

Logicals

Most useful skills

1. Math with logicals

Math

When you do math with logicals, **TRUE becomes 1** and **FALSE becomes 0**.

Math

When you do math with logicals, **TRUE becomes 1** and **FALSE becomes 0**.

- The **sum** of a logical vector is the **count of TRUEs**

```
sum(c(TRUE, FALSE, TRUE, TRUE))
##  3
```

Math

When you do math with logicals, **TRUE becomes 1** and **FALSE becomes 0**.

- The **sum** of a logical vector is the **count of TRUEs**

```
sum(c(TRUE, FALSE, TRUE, TRUE))
##  3
```

- The **mean** of a logical vector is the **proportion of TRUEs**

```
mean(c(1, 2, 3, 4) < 4)
##  0.75
```

Your Turn

Decide in your group: what proportion of flights do you think end up delayed?

Create a logical variable in flights that displays whether a flight was delayed (arr_delay > 0). Remove all NAs in the variable.

Then create a summary table that shows:

1. How many flights were delayed
2. What proportion of flights were delayed

```
flights %>%
  mutate(delayed = arr_delay > 0) %>%
  drop_na(delayed) %>%
  summarise(total = sum(delayed), prop = mean(delayed))
## # A tibble: 1 × 2
##   total      prop
##   <int>     <dbl>
## 1 133004 0.4063101
```

Strings

(character) **strings**

Anything surrounded by quotes(") or single quotes(').

```
> "one"
> "1"
> "one’s"
> '"Hello World"'
> "foo
+
+
+ oops. I’m stuck in a string."
```

Most useful skills

1. How to extract/ replace substrings
2. How to find matches for patterns
3. Regular expressions

stringr

Simple, consistent functions for working with strings.

```
# install.packages("tidyverse")
library(stringr)
```

```
install.packages("tidyverse")
```

does the equivalent of

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("tidyr")
install.packages("readr")
install.packages("purrr")
install.packages("tibble")
install.packages("hms")
install.packages("stringr")
install.packages("lubridate")
install.packages("forcats")
install.packages("DBI")
install.packages("haven")
install.packages("httr")
install.packages("jsonlite")
install.packages("readxl")
install.packages("rvest")
install.packages("xml2")
install.packages("modelr")
install.packages("broom")
```

```
library("tidyverse")
```

does the equivalent of

```
library("ggplot2")
library("dplyr")
library("tidyr")
library("readr")
library("purrr")
library("tibble")
```

```
install.packages("tidyverse")
```

does the equivalent of

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("tidyr")
install.packages("readr")
install.packages("purrr")
install.packages("tibble")
install.packages("hms")
install.packages("stringr")
install.packages("lubridate")
install.packages("forcats")
install.packages("DBI")
install.packages("haven")
install.packages("httr")
install.packages("jsonlite")
install.packages("readxl")
install.packages("rvest")
install.packages("xml2")
install.packages("modelr")
install.packages("broom")
```

```
library("tidyverse")
```

does the equivalent of

```
library("ggplot2")
library("dplyr")
library("tidyr")
library("readr")
library("purrr")
library("tibble")
```

```
View(babynames)
```

	year	sex	name	n	prop
1	1880	F	Mary	7065	0.0723835869
2	1880	F	Anna	2604	0.0266789611
3					05214897
4					8657856
5					8884278
6					1672045
7					0811946
8					144869628
9	1880	F	Bertha	1320	0.0135238973
10	1880	F	Sarah	1288	0.0131960453
11	1880	F	Annie	1258	0.0128886840

Are girls names more likely to end in a vowel? How much more likely?

str_sub()

Extract or replace portions of a string with **str_sub()**

```
str_sub(string, start = 1L, end = -1L)
```

string(s) to manipulate

position of first character to extract within each string

position of last character to extract within each string

Quiz

What will this return?

```
str_sub("Garrett", 1, 2)
```

Quiz

What will this return?

```
str_sub("Garrett", 1, 2)
```

"Ga"

Quiz

What will this return?

```
str_sub("Garrett", 1, 1)
```

Quiz

What will this return?

```
str_sub("Garrett", 1, 1)
```

"G"

Quiz

What will this return?

```
str_sub("Garrett", 2)
```

Quiz

What will this return?

```
str_sub("Garrett", 2)
```

"arrett"

Quiz

What will this return?

```
str_sub("Garrett", -3)
```

Quiz

What will this return?

```
str_sub("Garrett", -3)
```

```
"ett"
```

Quiz

What will this return?

```
g <- "Garrett"
str_sub(g, -3) <- "eth"
g
```

Quiz

What will this return?

```
g <- "Garrett"
str_sub(g, -3) <- "eth"
g
```

"Garreth"

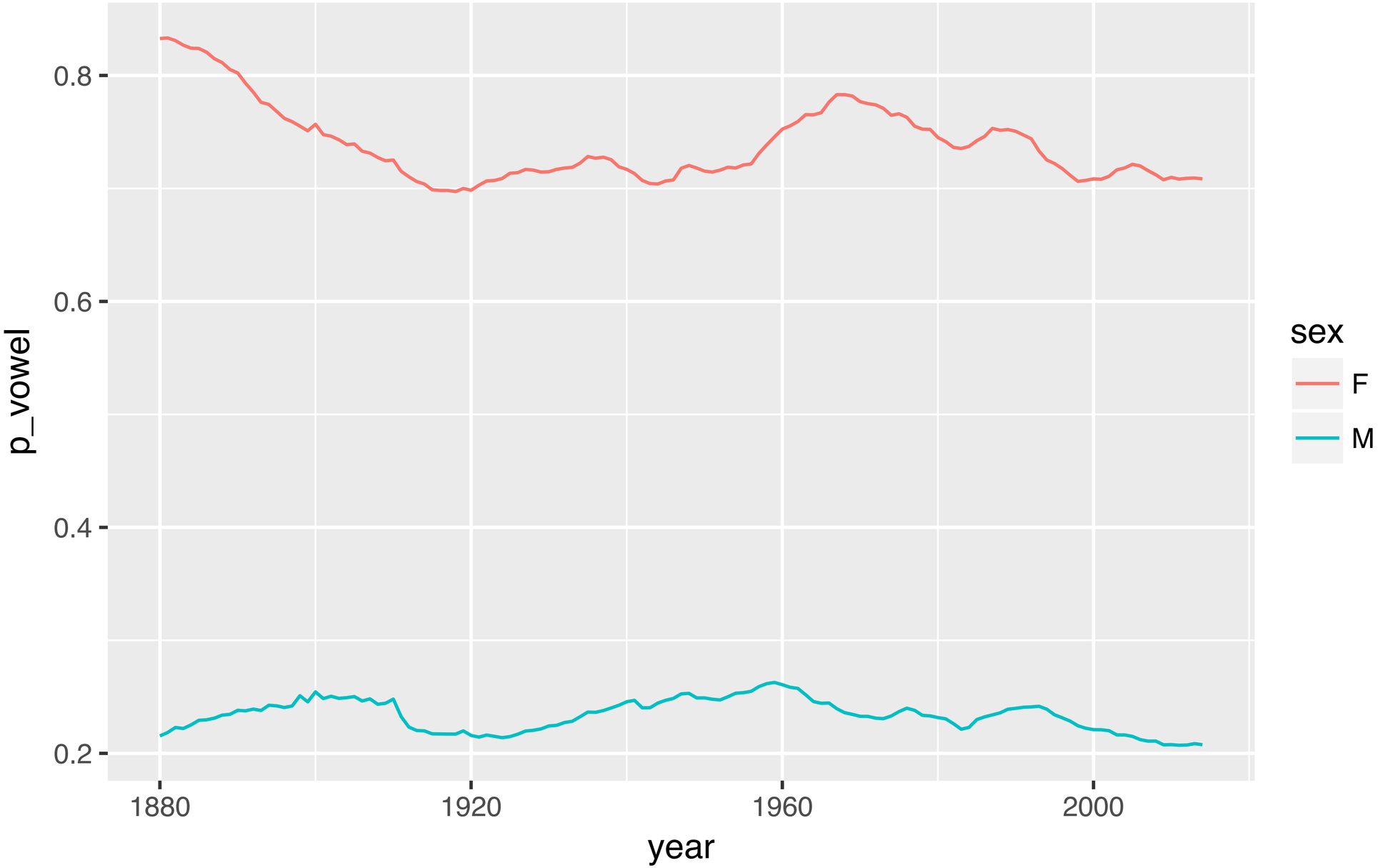
Your Turn

In your group:

1. Isolate the last letter of every name
2. and create a logical variable that displays whether the last letter is one of "a", "e", "i", "o", "u", or "y".
3. Use a weighted mean to calculate the proportion of children whose name ends in a vowel (by year and sex)
4. and then display the results as a line plot.

```
babynames %>%
  mutate(last = str_sub(name, -1),
    vowel = last %in% c("a", "e", "i", "o", "u", "y")) %>%
  group_by(year, sex) %>%
  summarise(p_vowel = weighted.mean(vowel, n)) %>%
  ggplot(aes(year, p_vowel, color = sex)) +
    geom_line()
```

Proportion of names that end in a vowel



Your Turn

Run **help(package = stringr)** to open the help directory for stringr.

Read through the function descriptions and find the function that determines whether two strings match.

str_detect()

Test whether a pattern appears within a string.

```
str_detect(string, pattern)
```

vector of strings to find patterns in

a string that represents a regular expression

Quiz

What will this return?

```
strings <- c("Apple", "Orange")
str_detect(strings, "pp")
```

Quiz

What will this return?

```
strings <- c("Apple", "Orange")
str_detect(strings, "pp")
```

TRUE FALSE

Quiz

What will this return?

```
strings <- c("Apple", "Pineapple")
str_detect(strings, "apple")
```

Quiz

What will this return?

```
strings <- c("Apple", "Pineapple")
str_detect(strings, "apple")
```

```
FALSE  TRUE
```

Quiz

What will this return?

```
"AC\DC"
```

Quiz

What will this return?

```
"AC\DC"
```

```
Error: '\D' is an unrecognized escape
in character string starting ""AC\D"
```

special characters

A backslash followed by one or more characters which stand for a special type of character.

Use **writeLines()** to see the characters represented by a string.

```
## Use an escape to include a special character
writeLines("\"")
## "


writeLines("\\")
## \
```

A string is not a sequence of characters itself, but a representation of a sequence of characters

```
writeLines("AC\\DC")
## AC\DC
```

Quiz

What will this return?

```
str_detect("AC\\DC", "\\")
```

Quiz

What will this return?

```
str_detect("AC\\DC", "\\")
```

```
  Error in stri_detect_regex(string,
pattern, opts_regex = opts(pattern)) :
  Unrecognized backslash escape sequence
in pattern. (U_REGEX_BAD_ESCAPE_SEQUENCE)
```

regular expressions

regex is a "language" for describing a pattern of characters, with patterns of characters, e.g.

\w*[aeiouy]\b

```
writeLines("\w*[aeiouy]\b")
## ERROR" \w not a recognized special character

writeLines("\\w*[aeiouy]\\b")
## "\w*[aeiouy]\b"
```

```
writeLines("\\")
## \
```

This is not a valid regular expression (looks like an incomplete special character)

We want to find this

```
\
```

…which is represented by this regex

```
\\
```

…which is represented by this string

```
"\\\\"
```

Write your patterns at this level

We want to find this

```
?
```

…which is represented by this regex

```
\
```

…which is represented by this string

```
"\\"
```

We want to find this

```
abc
```

…which is represented by this regex

```
abc
```

…which is represented by this string

```
"abc"
```

Your Turn

How many names end in a vowel anyways?

1. Winnow babynames to distinct names with:

```
babynames %>% distinct(name)
```

2. Use str_detect() to detect names that end in a vowel (pattern = "\\w*[aeiouy]\\b")
3. Calculate the number of names that end in a vowel

```
babynames %>%
  distinct(name) %>%
  mutate(vowel = str_detect(name, "\\w*[aeiouy]\\b")) %>%
  summarise(n = n(), n_vowel = sum(vowel))
# A tibble: 1 × 2
      n n_vowel
  <int>   <int>
1 93889   55324
```

str_view()

Displays the matches of a regular expression in the Viewer pane

```
str_view(c("who", "what", "why"), "\\w*[aeiouy]\\b")
```

```
who
what
why
```

Factors

factors

R's representation of categorical data. Consists of:

1. A set of **values**
2. An ordered set of **valid levels**

```
eyes <- factor(x = c("blue", "green", "green"),
               levels = c("blue", "brown", "green"))
eyes
## [1] blue  green green
## Levels: blue brown green
```

Most useful skills

1. Reorder the levels
2. Recode the levels
3. Collapse levels

gss_cat

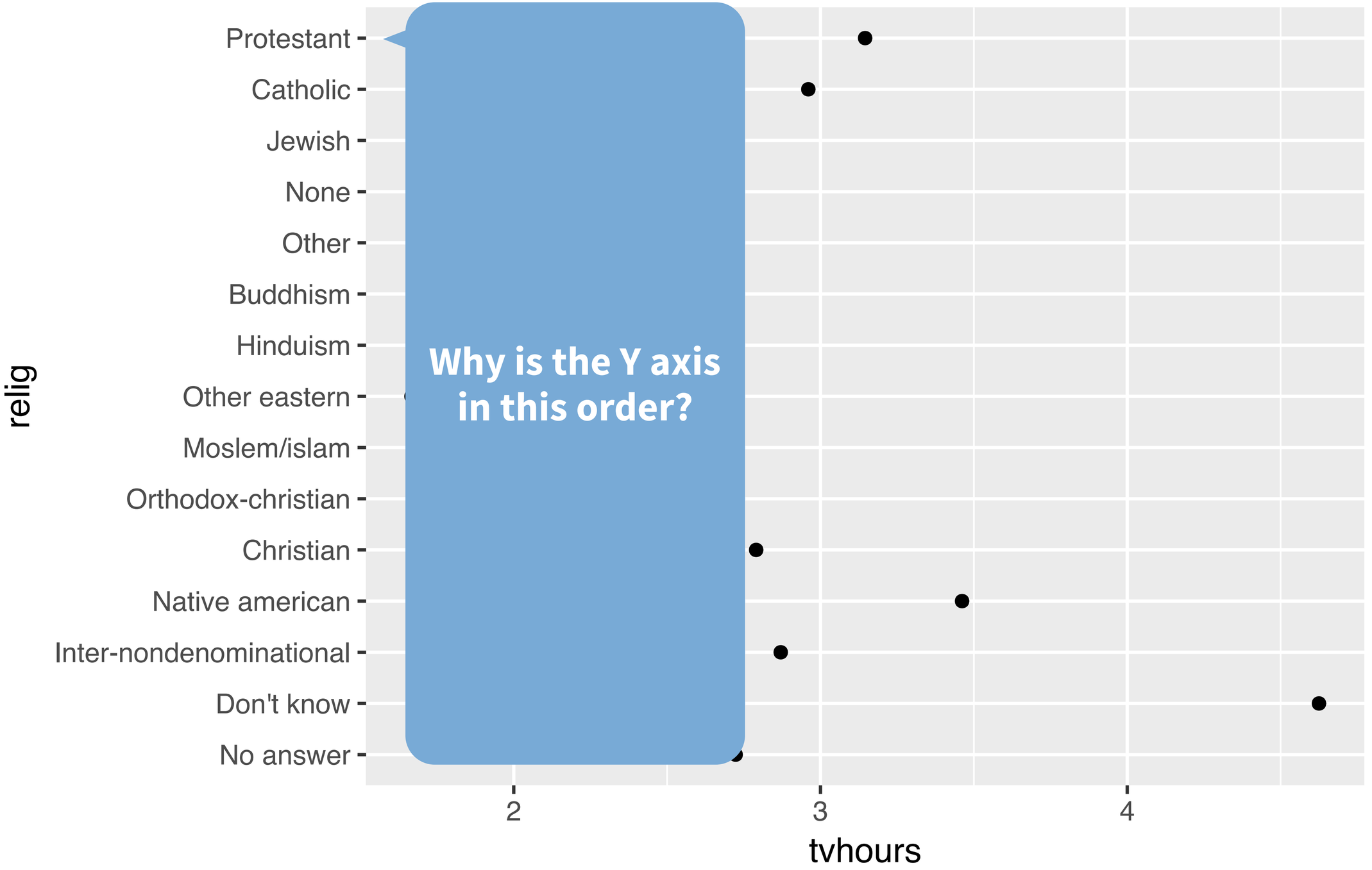
```
library(forcats)
View(gss_cat)
```

A sample of data from the General Social Survey, a long-running US survey conducted by NORC at the University of Chicago.

year	marital	age	race	rincome	partyid	relig	denom	tvhours
2000	Never married	26	White	$8000 to 9999	Ind,near rep	Protestant	Southern baptist	12
2000	Divorced	48	White	$8000 to 9999	Not str republican	Protestant	Baptist-dk which	*NA*
2000	Widowed	67	White	Not applicable	Independent	Protestant	No denomination	2
2000	Never married	39	White	Not applicable	Ind,near rep	Orthodox-christian	Not applicable	4
2000	Divorced	25	White	Not applicable	Not str democrat	None	Not applicable	1
2000	Married	25	White	$20000 - 24999	Strong democrat	Protestant	Southern baptist	*NA*
2000	Never married	36	White	$25000 or more	Not str republican	Christian	Not applicable	3
2000	Divorced	44	White	$7000 to 7999	Ind,near dem	Protestant	Lutheran-mo synod	*NA*
2000	Married	44	White	$25000 or more	Not str democrat	Protestant	Other	0

Which religions watch the most TV?

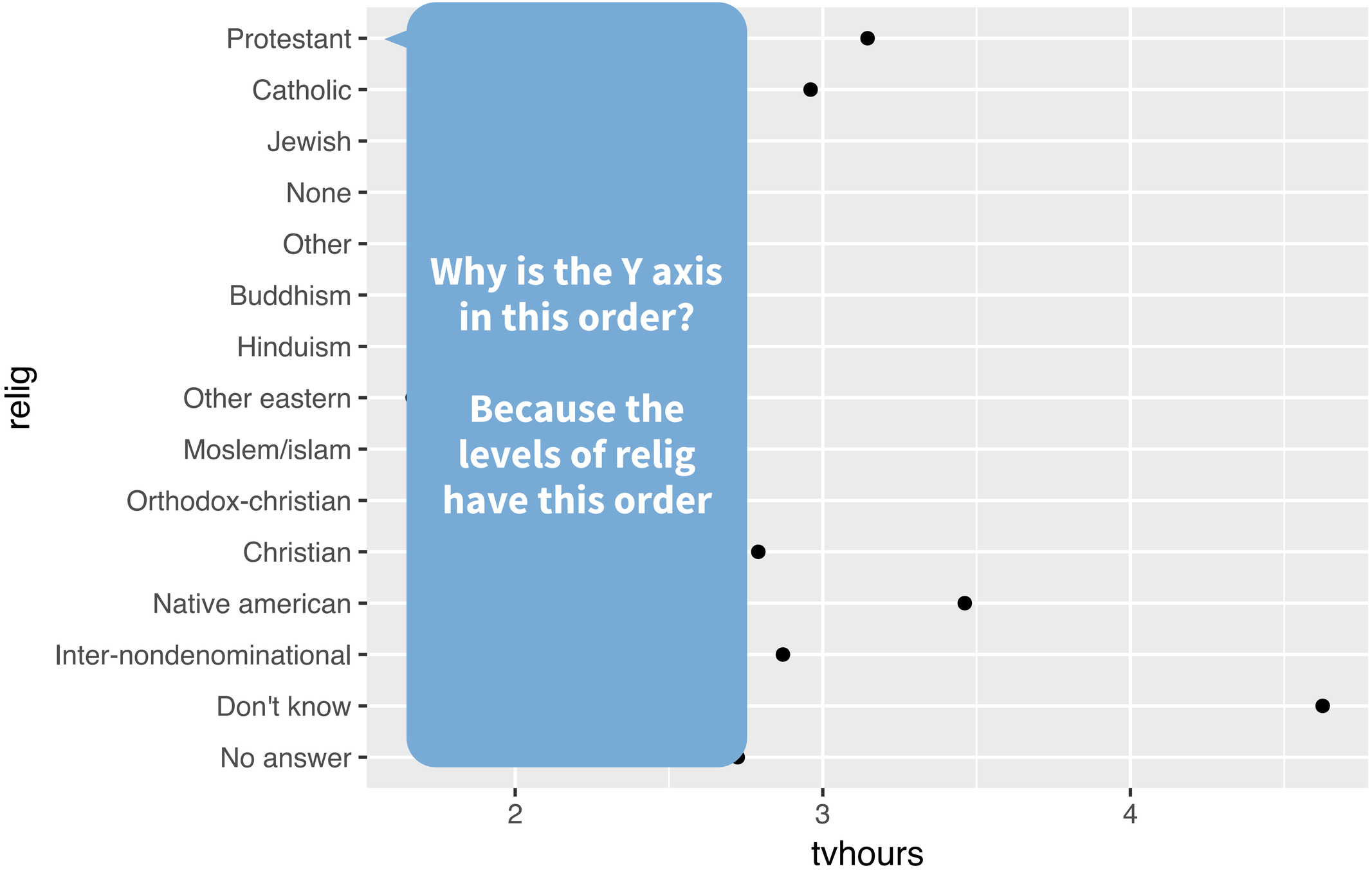
```
gss_cat %>%
  drop_na(tvhours) %>%
  group_by(relig) %>%
  summarise(tvhours = mean(tvhours)) %>%
  ggplot(aes(tvhours, relig)) +
    geom_point()
```


Protestant
Catholic
Jewish
None
Other
Buddhism
Hinduism
Other eastern
Moslem/islam
Orthodox-christian
Christian
Native american
Inter-nondenominational
Don't know
No answer
relig
2
3
4
tvhours
Why is the Y axis in this order?

levels()

Use **levels()** to access a factor's levels

```
levels(gss_cat$relig)
##  [1] "No answer"                "Don't know"
## [3] "Inter-nondenominational" "Native american"
## [5] "Christian"                "Orthodox-christian"
## [7] "Moslem/islam"             "Other eastern"
## [9] "Hinduism"                 "Buddhism"
## [11] "Other"                    "None"
## [13] "Jewish"                   "Catholic"
## [15] "Protestant"               "Not applicable"
```


Protestant
Catholic
Jewish
None
Other
Buddhism
Hinduism
Other eastern
Moslem/islam
Orthodox-christian
Christian
Native american
Inter-nondenominational
Don't know
No answer
relig
2
3
4
tvhours
Why is the Y axis in this order?
Because the levels of relig have this order

Reordering levels

forcats

Simple functions for working with factors.

```
# install.packages("tidyverse")
library(forcats)
```

fct_reorder()

Reorders the levels of a factor based on the result of fun(x) applied to each group of cases (grouped by level).

```
fct_reorder(f, x, fun = median, …, .desc = FALSE)
```

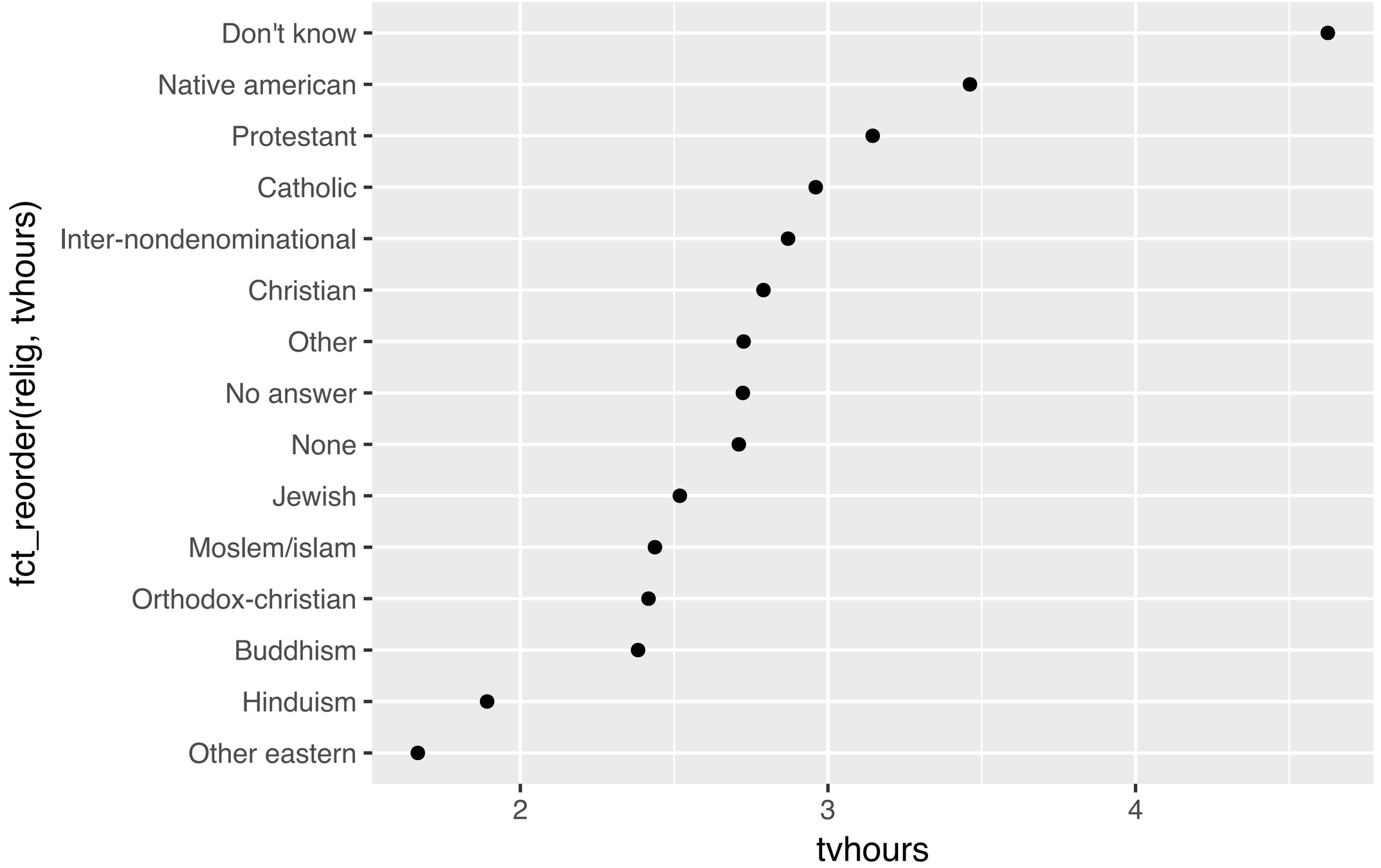
factor to reorder

variable to reorder by (in conjunction with fun)

function to reorder by (in conjunction with x)

put in descending order?

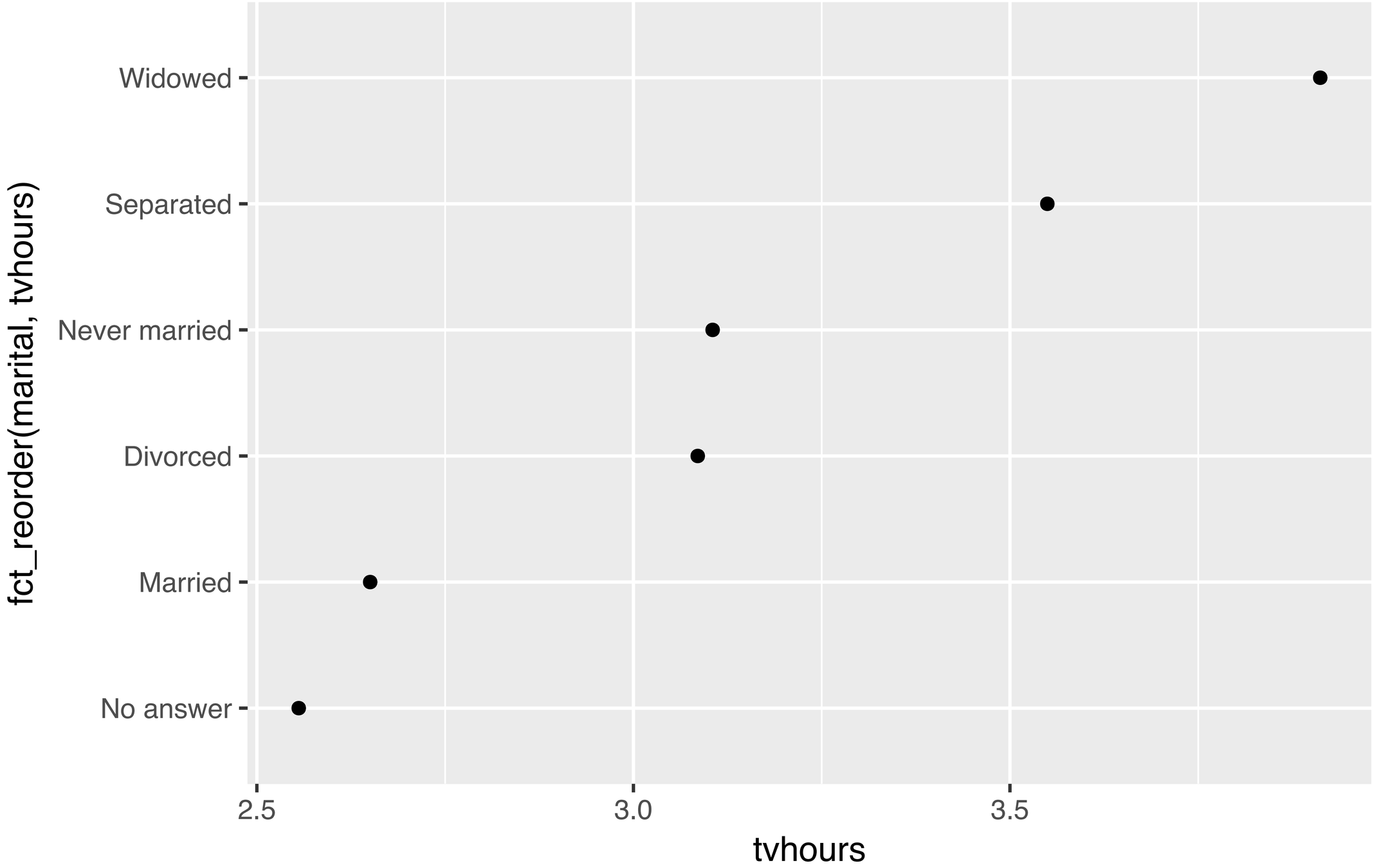
```
gss_cat %>%
  drop_na(tvhours) %>%
  group_by(relig) %>%
  summarise(tvhours = mean(tvhours)) %>%
  ggplot(aes(tvhours, fct_reorder(relig, tvhours))) +
    geom_point()
```


Don't know
Native american
Protestant
Catholic
Inter-nondenominational
Christian
Other
No answer
None
Jewish
Moslem/islam
Orthodox-christian
Buddhism
Hinduism
Other eastern
fct_reorder(relig, tvhours)
2
3
4
tvhours

Your Turn

Repeat the previous exercise to make a sensible graph of average TV consumption by marital status.

```
gss_cat %>%
  drop_na(tvhours) %>%
  group_by(marital) %>%
  summarise(tvhours = mean(tvhours)) %>%
  ggplot(aes(tvhours, fct_reorder(marital, tvhours))) +
    geom_point()
```


Widowed
Separated
Never married
Divorced
Married
No answer
fct_reorder(marital, tvhours)
2.5
3.0
3.5
tvhours

fct_reorder2()

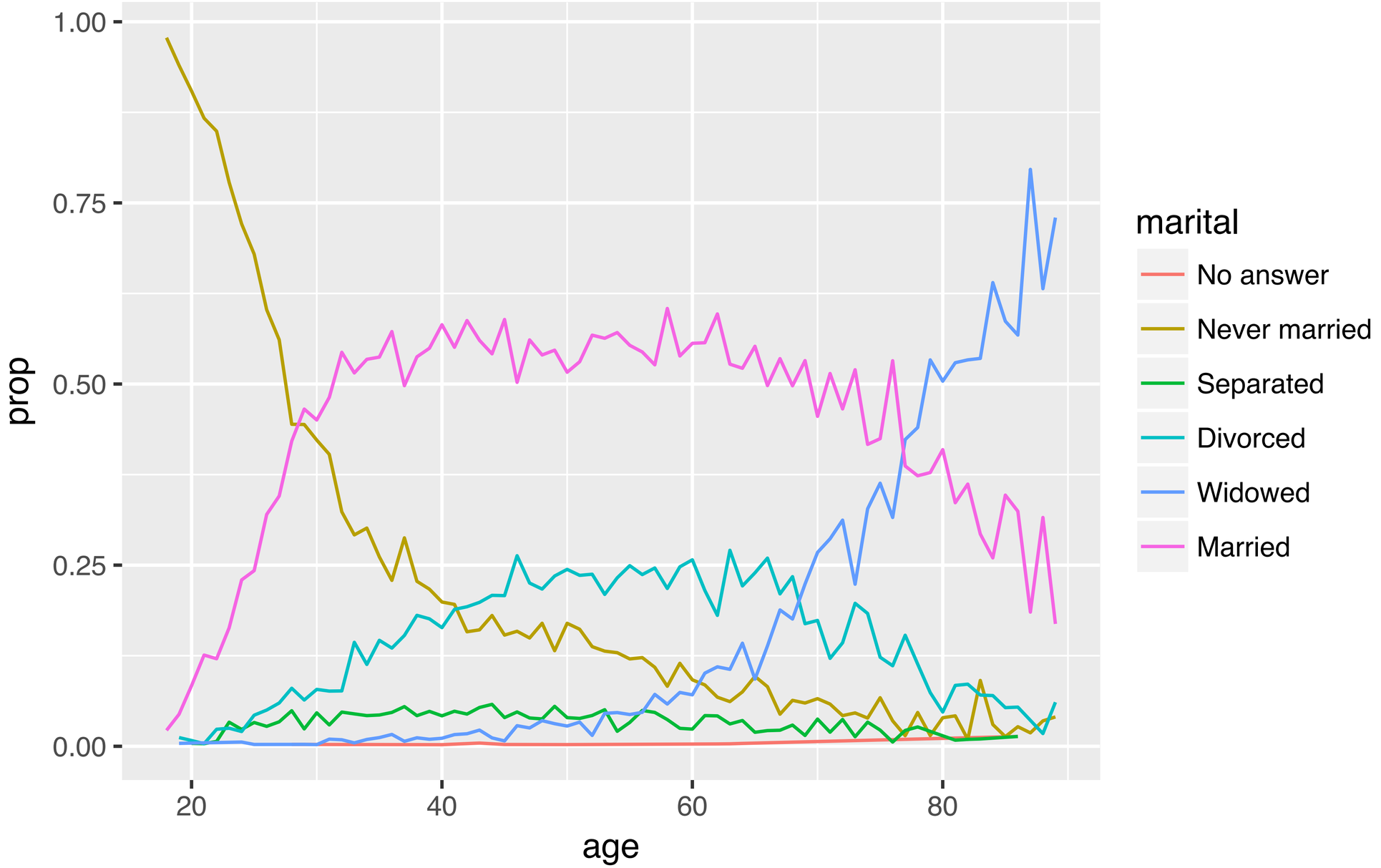
Reorders the levels of a factor by the **Y values associated with the largest X values**.

```
fct_reorder2(f, x, y, fun = last2, …, .desc = FALSE)
```

X variable

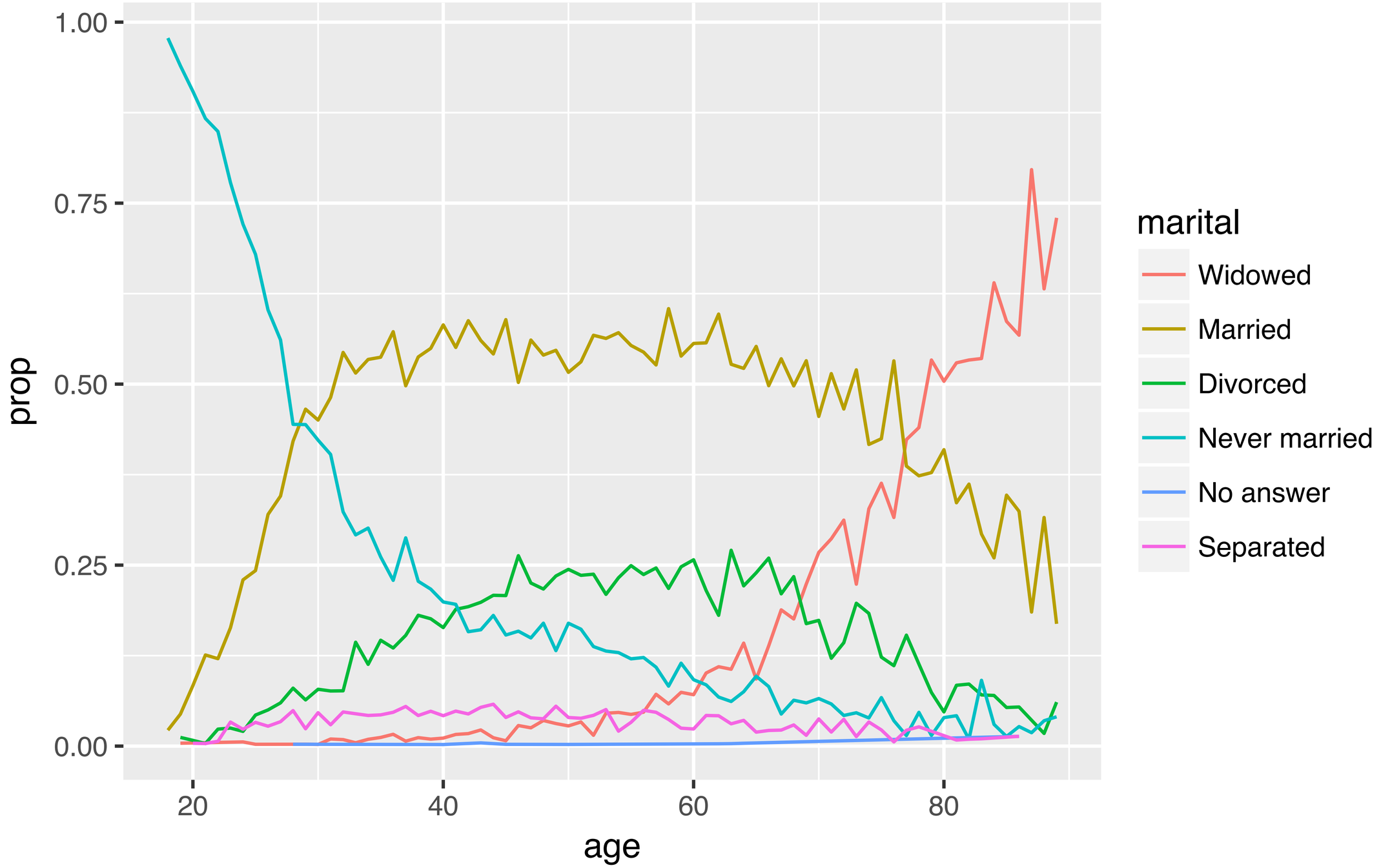
Y variable

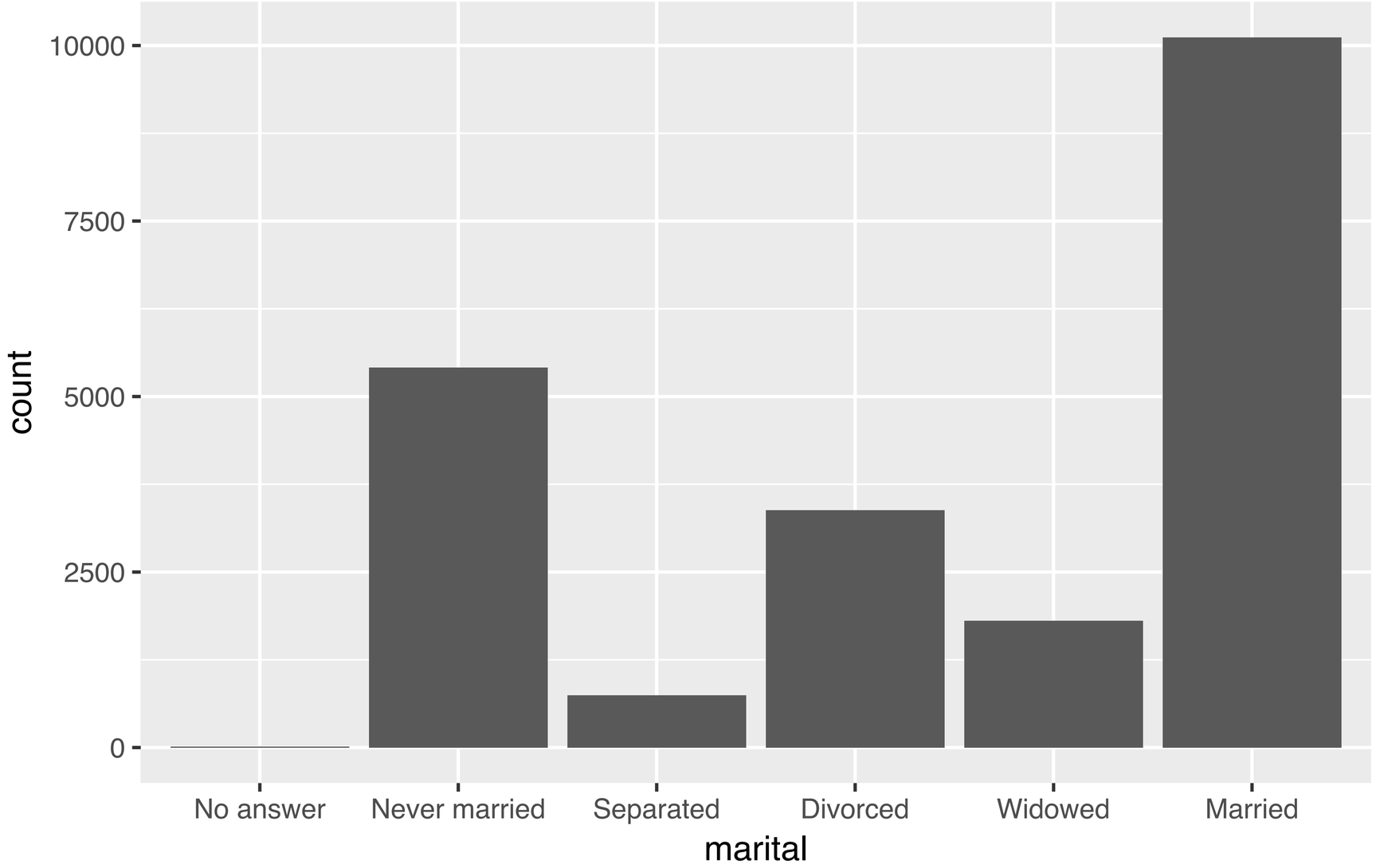
Proportion who report each marital status by age



```
gss_cat %>%
  drop_na(age) %>%
  group_by(age, marital) %>%
  count() %>%
  mutate(prop = n / sum(n)) %>%
  ggplot(aes(age, prop, color = marital)) +
    geom_line()
```

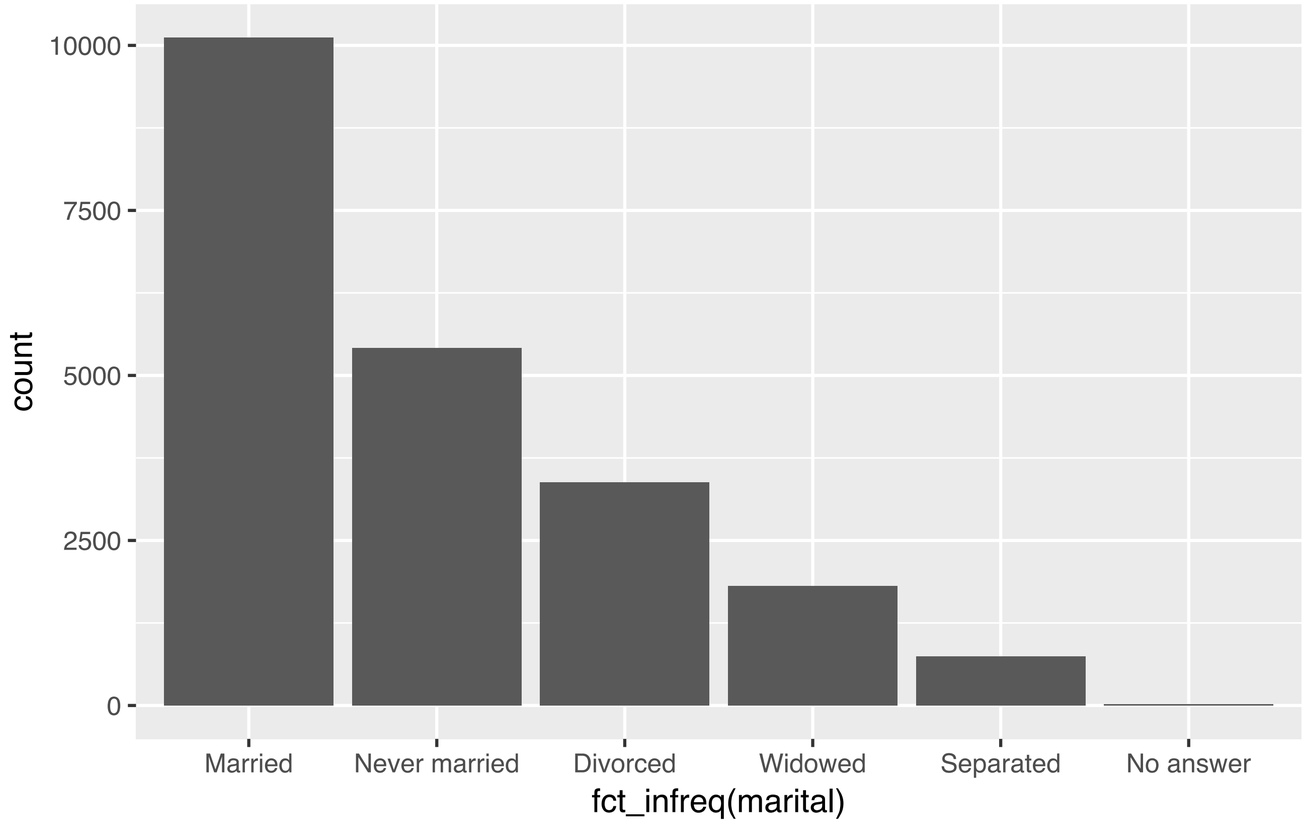
```
gss_cat %>%
  drop_na(age) %>%
  group_by(age, marital) %>%
  count() %>%
  mutate(prop = n / sum(n)) %>%
  ggplot(aes(age, prop,
    color = fct_reorder2(marital, age, prop)) +
    geom_line()
```


1.00
0.75
0.50
0.25
0.00
prop
20
40
60
80
age
marital
Widowed
Married
Divorced
Never married
No answer
Separated




```
gss_cat %>%
  ggplot(aes(marital)) + geom_bar()
```

fct_infreq



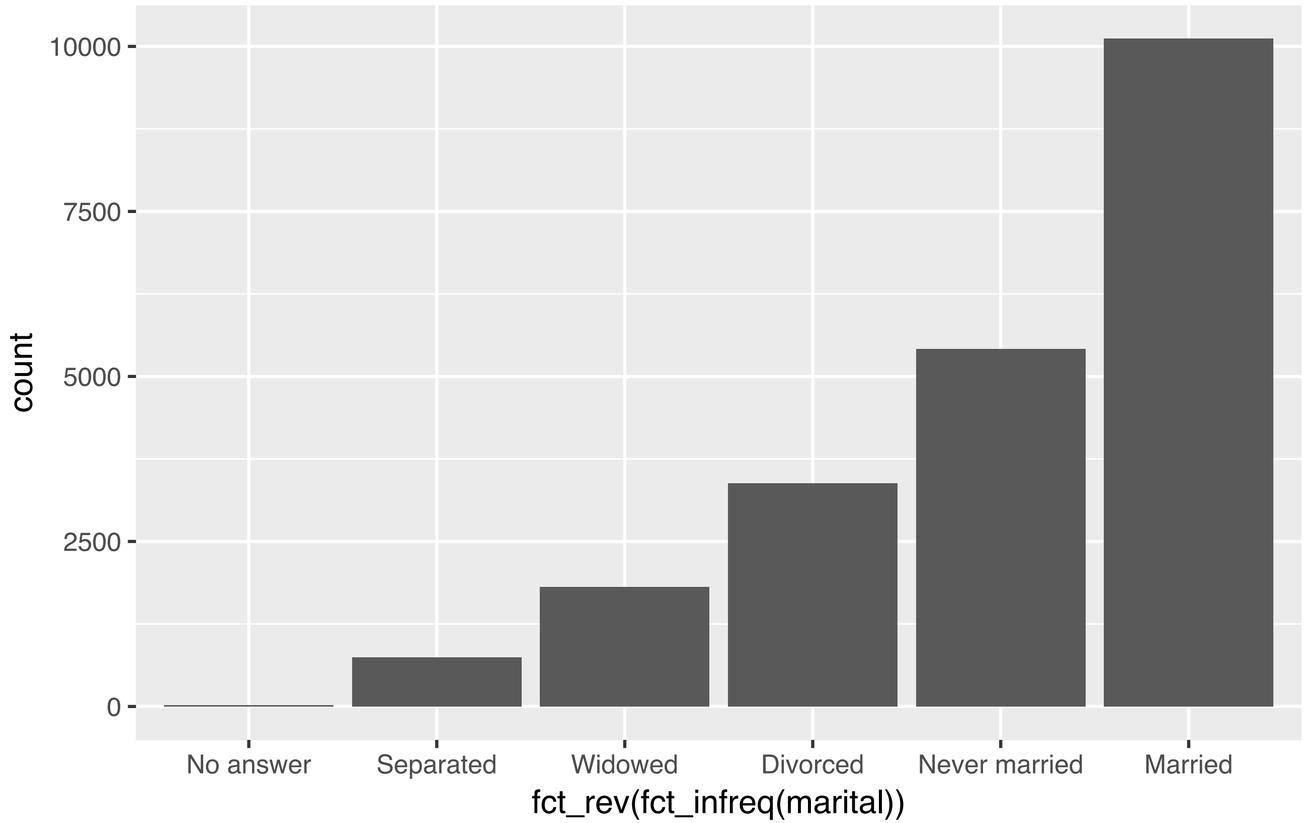

```
gss_cat %>%
  ggplot(aes(fct_infreq(marital))) + geom_bar()
```

Quiz

How is **fct_rev()** going to alter the graph?

```
gss_cat %>%
  ggplot(aes(fct_rev(fct_infreq(marital)))) + geom_bar()
```

fct_rev




```
gss_cat %>%
  ggplot(aes(fct_rev(fct_infreq(marital)))) + geom_bar()
```

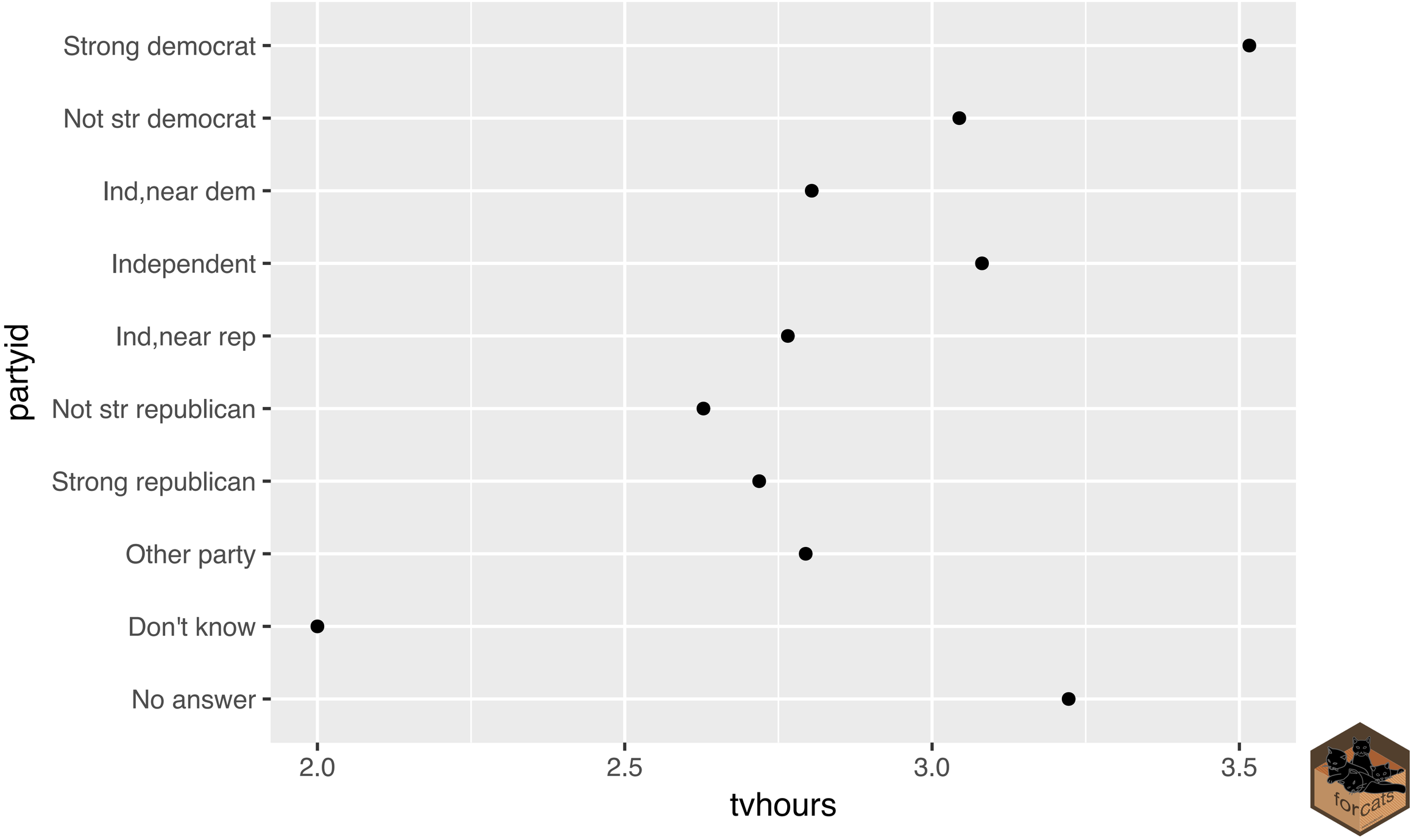
Changing level values

Your Turn

Do you think liberals or conservatives watch more TV?

Compute average tv hours by party ID an then plot the results.

```
gss_cat %>%
    drop_na(tvhours) %>%
    group_by(partyid) %>%
    summarise(tvhours = mean(tvhours)) %>%
    ggplot(aes(tvhours, partyid)) +
      geom_point()
```


Strong democrat
Not str democrat
Ind,near dem
Independent
Ind,near rep
Not str republican
Strong republican
Other party
Don't know
No answer
partyid
2.0
2.5
3.0
3.5
tvhours

fct_recode()

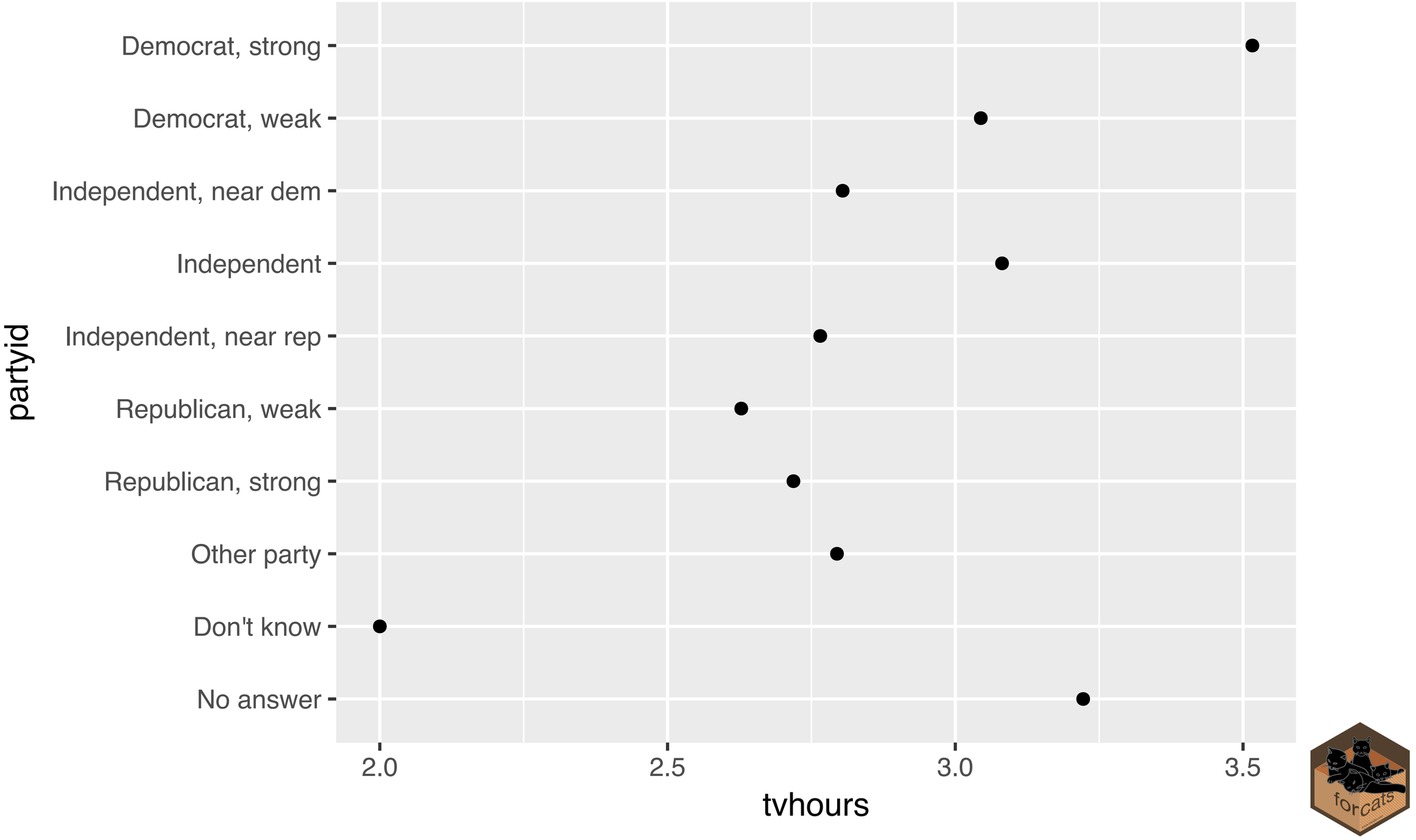
Changes values of levels

```
fct_recode(f, …)
```

factor with levels

new level = old level pairs (as a named character vector)

```
gss_cat %>%
    drop_na(tvhours) %>%
     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")) %>%
    group_by(partyid) %>%
    summarise(tvhours = mean(tvhours)) %>%
    ggplot(aes(tvhours, partyid)) +
      geom_point()
```


Democrat, strong
Democrat, weak
Independent, near dem
Independent
Independent, near rep
Republican, weak
Republican, strong
Other party
Don't know
No answer
partyid
2.0
2.5
3.0
3.5
tvhours

fct_collapse()

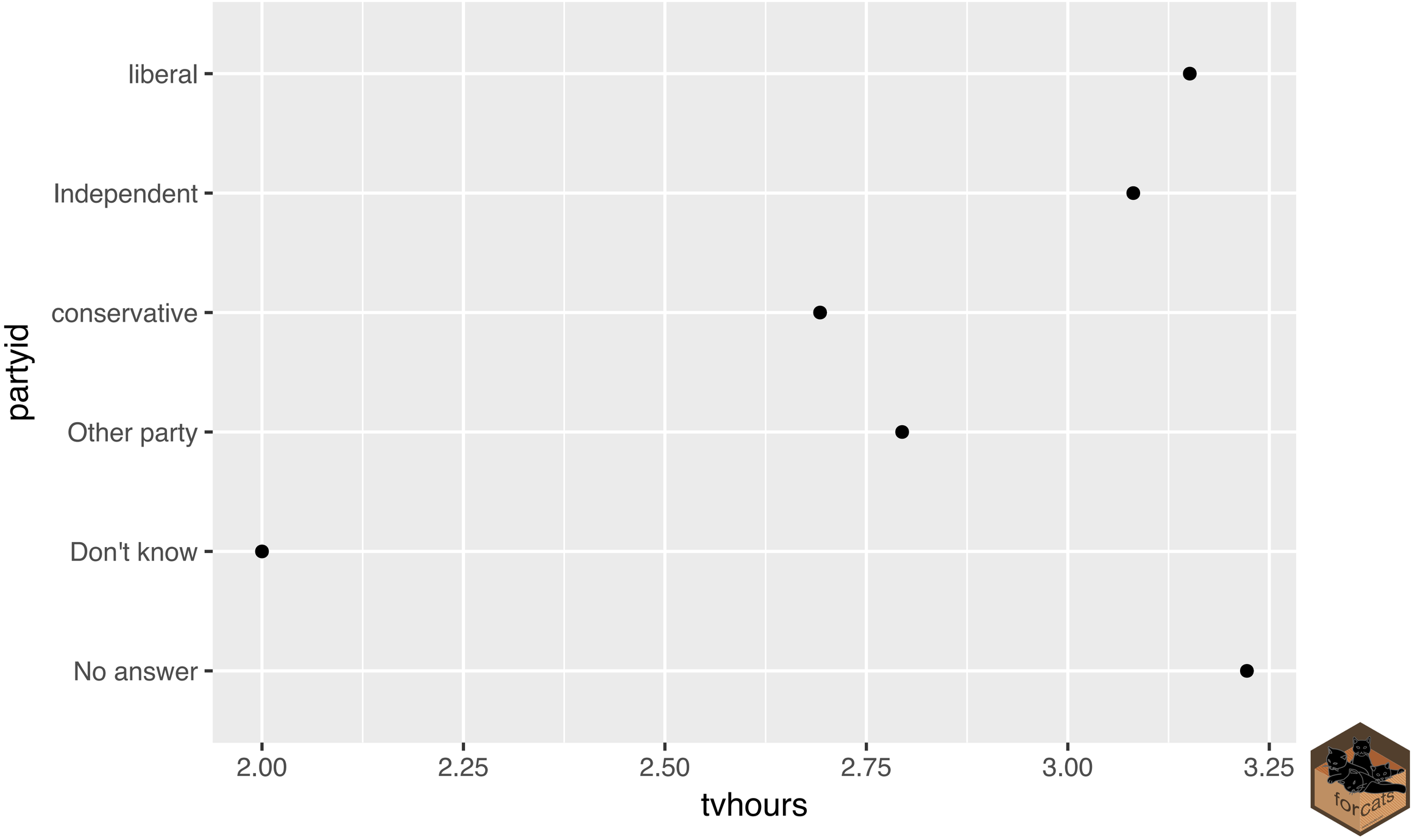
Changes multiple levels into single levels

```
fct_collapse(f, …)
```

factor with levels

named arguments set to a character vector (levels in the vector will be collapsed to the name of the argument)

```
gss_cat %>%
    drop_na(tvhours) %>%
     mutate(partyid = fct_collapse(partyid,
       conservative = c("Strong republican",
                        "Not str republican",
                        "Ind,near rep"),
       liberal = c("Strong democrat",
                   "Not str democrat",
                   "Ind,near dem"))) %>%
    group_by(partyid) %>%
    summarise(tvhours = mean(tvhours)) %>%
    ggplot(aes(tvhours, partyid)) +
      geom_point()
```


liberal
Independent
conservative
Other party
Don't know
No answer
partyid
2.00
2.25
2.50
2.75
3.00
3.25
tvhours

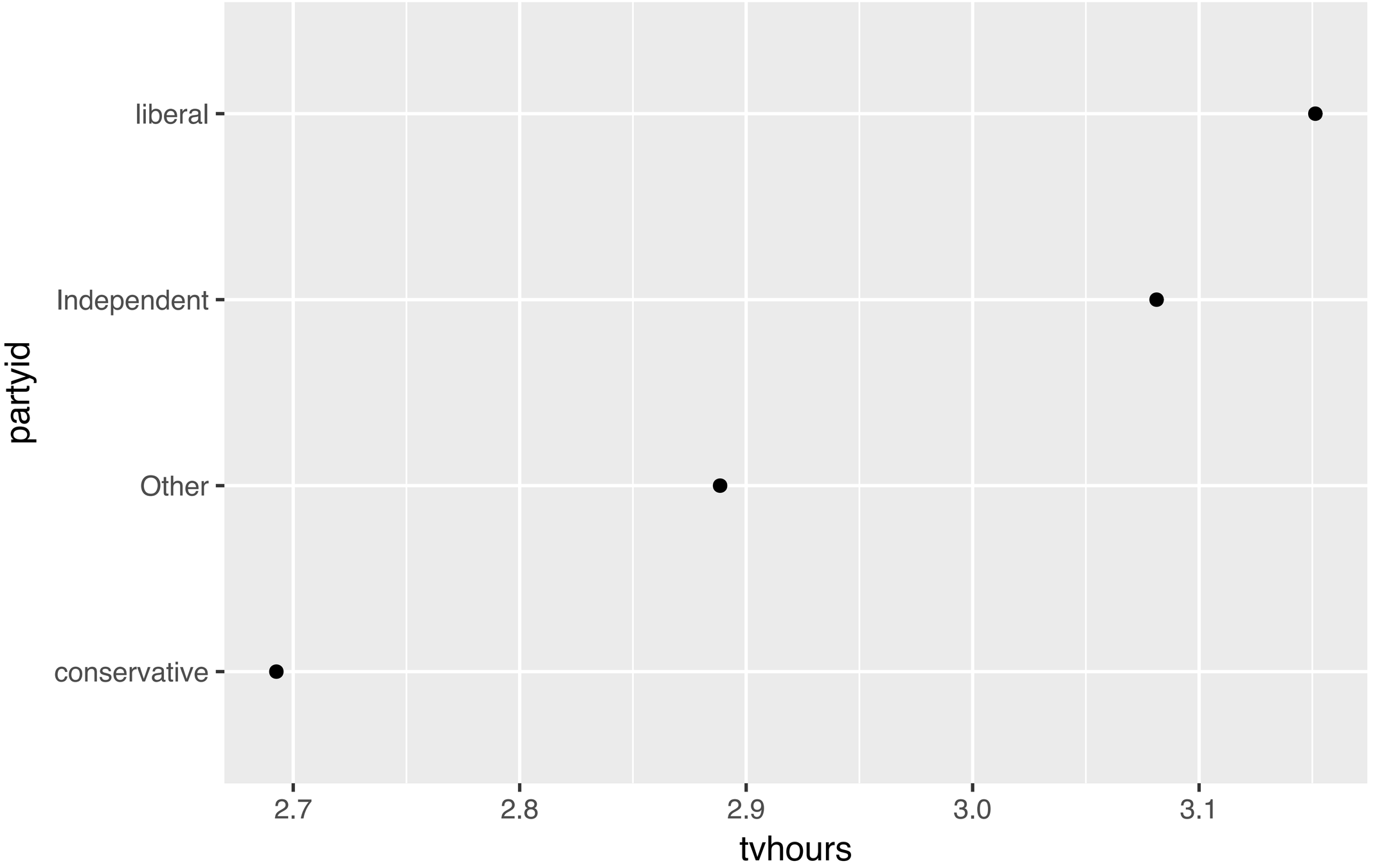
fct_lump()

Collapses levels with fewest values into a single level. By default collapses as many levels as possible such that the new level is still the smallest.

```
fct_lump(f, other_level = "Other", …)
```

factor with levels

name of new level


liberal
Independent
Other
conservative
partyid
2.7
2.8
2.9
3.0
3.1
tvhours

```
gss_cat %>%
  drop_na(tvhours) %>%
  mutate(partyid = partyid %>%
    fct_lump() %>%
    fct_collapse(
      conservative = c("Strong republican",
                       "Not str republican", "Ind,near rep"),
      liberal = c("Strong democrat", "Not str democrat",
                  "Ind,near dem")) %>%
    fct_reorder(tvhours, mean)
  ) %>%
  group_by(partyid) %>%
  summarise(tvhours = mean(tvhours)) %>%
  ggplot(aes(tvhours, partyid)) +
    geom_point()
```

Date times

Quiz

Does every year have 365 days?

Quiz

Does every day have 24 hours?

Quiz

Does every minute have 60 seconds?

Quiz

What does a month measure?

Dates and times involve:

1. **instants** - specific moments in time (e.g., Jan 1, 2017)
2. **time spans** - spans of time (e.g., a day)

Most useful skills

1. Parse a string into a date time class
2. Access and change parts of a date
3. Deal with time zones
4. Do math with instants and time spans

Parsing dates and times

lubridate

Functions for working with dates and time spans

```
# install.packages("tidyverse")
library(lubridate)
```

ymd() family

To parse strings as dates, use a y, m, d, h, m, s combo

```
ymd("2017/01/11")
mdy("January 11, 2017")
ymd_hms("2017-01-11  01:30:55")
```

Parsing functions

function	parses to
ymd_hms(), ymd_hm(), ymd_h() ydm_hms(), ydm_hm(), ydm_h() dmy_hms(), dmy_hm(), dmy_h() mdy_hms(), mdy_hm(), mdy_h()	POSIXct
ymd(), ydm(), mdy() myd(), dmy(), dym() yq()	Date (POSIXct if tz specified)

hms

A class for representing just clock times.

```
# install.packages("tidyverse")
library(hms)
```

hms

2017-01-01 **12:34:56**

Stored as the number of seconds since 00:00:00.*

```
library(hms)
hms(seconds = 56, min = 34, hour = 12)
## 12:34:56

unclass(hms(56, 34, 12))
## 45296
```

* on a typical day

hms()

2017-01-01 12:34:56

```
library(hms)
hms(seconds, minutes, hours, days)
```

Same name as hms() in lubridate

* on a typical day

hms::hms()

package name

function name

* on a typical day

hms::hms()

lubridate::hms()

* on a typical day

hms()

2017-01-01 **12:34:56**

```
library(hms)
hms::hms(seconds, minutes, hours, days)
```

Same name as hms() in lubridate

arguments order does not match name (so be explicit)

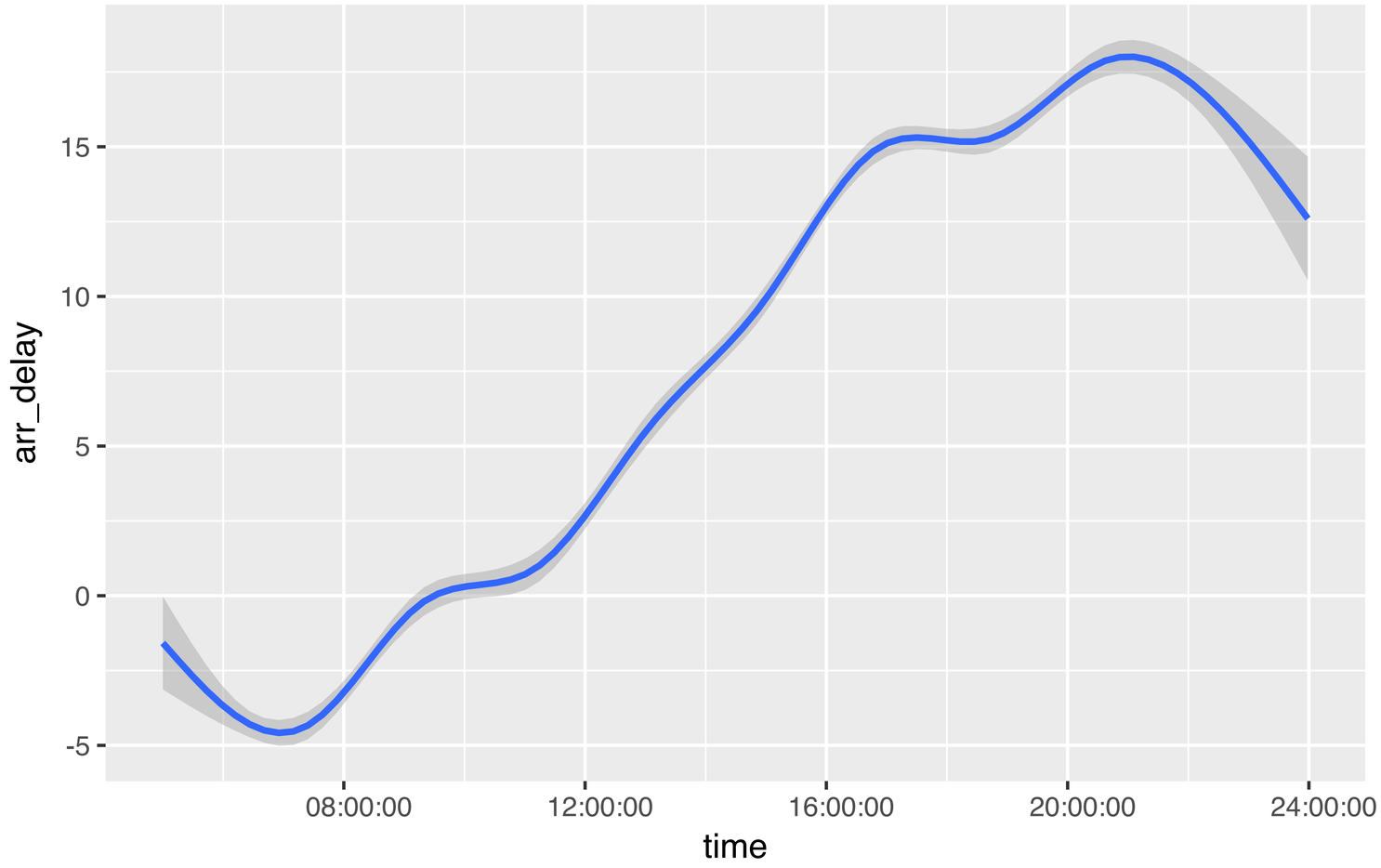
* on a typical day

Your Turn

What is the best time of day to fly?

Use the **hour** and **minute** variables in flights to compute the time of day as an hms, then use a smooth line to plot the relationship between time of day and arr_delay.

```
flights %>%
  mutate(time = hms::hms(hour = hour, minute = minute)) %>%
  ggplot(aes(time, arr_delay)) + geom_smooth()
```



Accessing and changing components

Accessing components

Extract components by name with a **singular** name

```
date <- ymd("2017-01-11")
year(date)
## 2017
```

Setting components

Use the same function to set components

```
date
## "2017-01-11"
year(date) <- 1999
date
## "1999-01-11"
```

Accessing date time components

function	extracts	extra arguments
year()	year	
month()	month	label = FALSE, abbr = TRUE
week()	week	
day()	day of month	
wday()	day of week	label = FALSE, abbr = TRUE
qday()	day of quarter	
yday()	day of year	
hour()	hour	
minute()	minute	
second()	second	

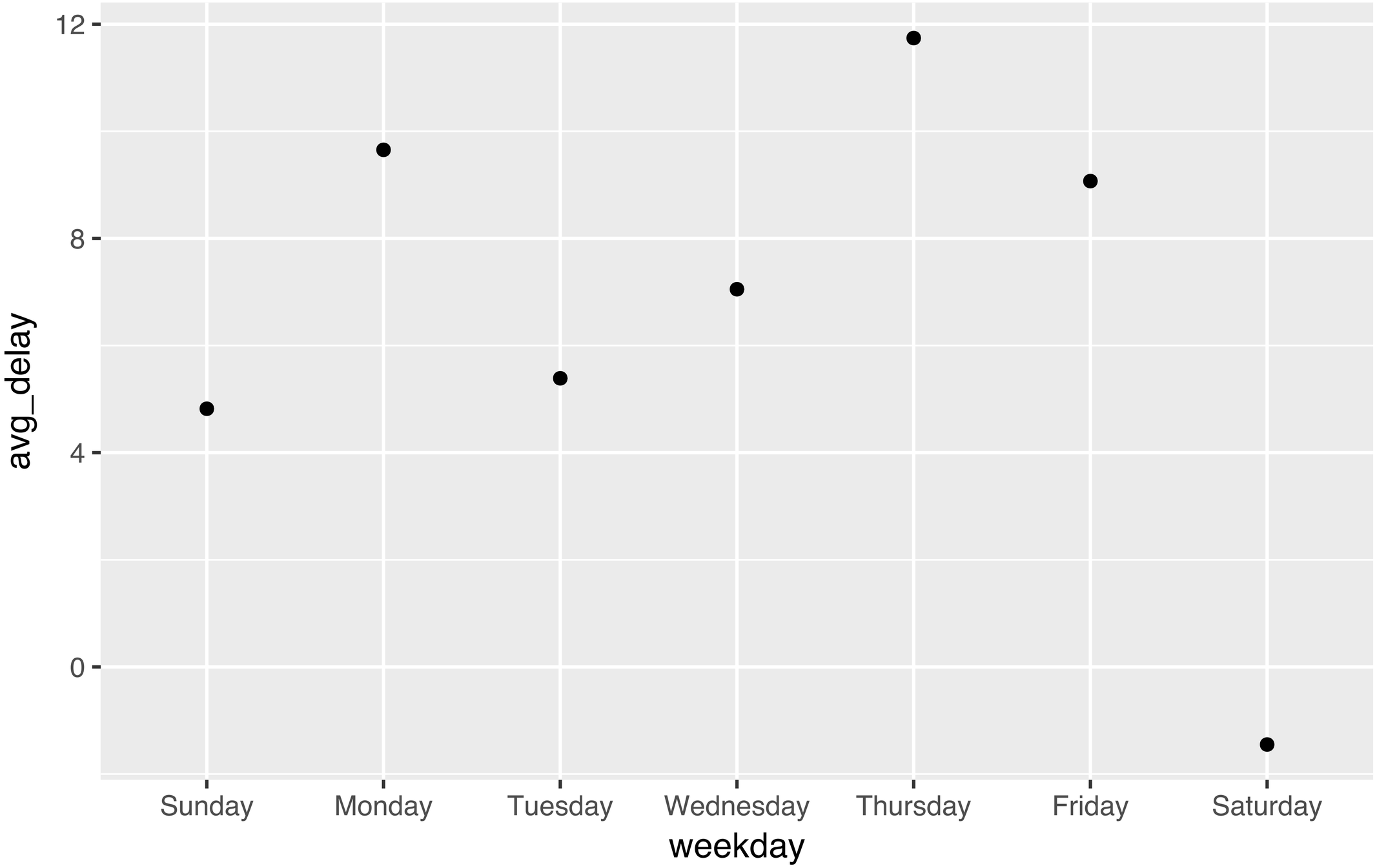
Accessing components

```
wday(ymd("2017-01-11"))
## 2017
wday(ymd("2017-01-11"), label = TRUE)
## [1] Wed
## 7 Levels: Sun < Mon < Tues < Wed < Thurs < ... < Sat
wday(ymd("2017-01-11"), label = TRUE, abbr = FALSE)
## [1] Sunday
## 7 Levels: Sunday < Monday < Tuesday < ... < Saturday
```

Your Turn

1. Use a parse function and an accessor function to determine which day of the week you were born on.
2. Extract the day of the week of each flight (as a full name) from time_hour. Calculate the average delay by day of the week and plot the results as a scatterplot.

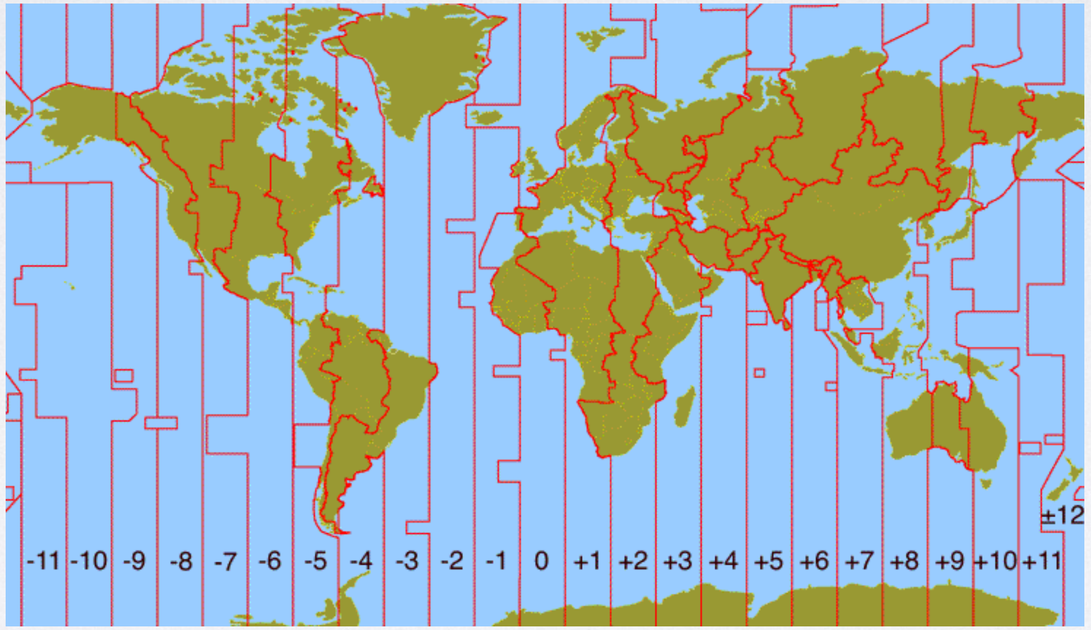
```
birthday <- mdy("February 22, 1732")
wday(birthday, label = TRUE, abbr = FALSE)
## [1] Friday
## 7 Levels: Sunday < Monday < Tuesday < ... < Saturday
```


12
8
4
0
avg_delay
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
weekday

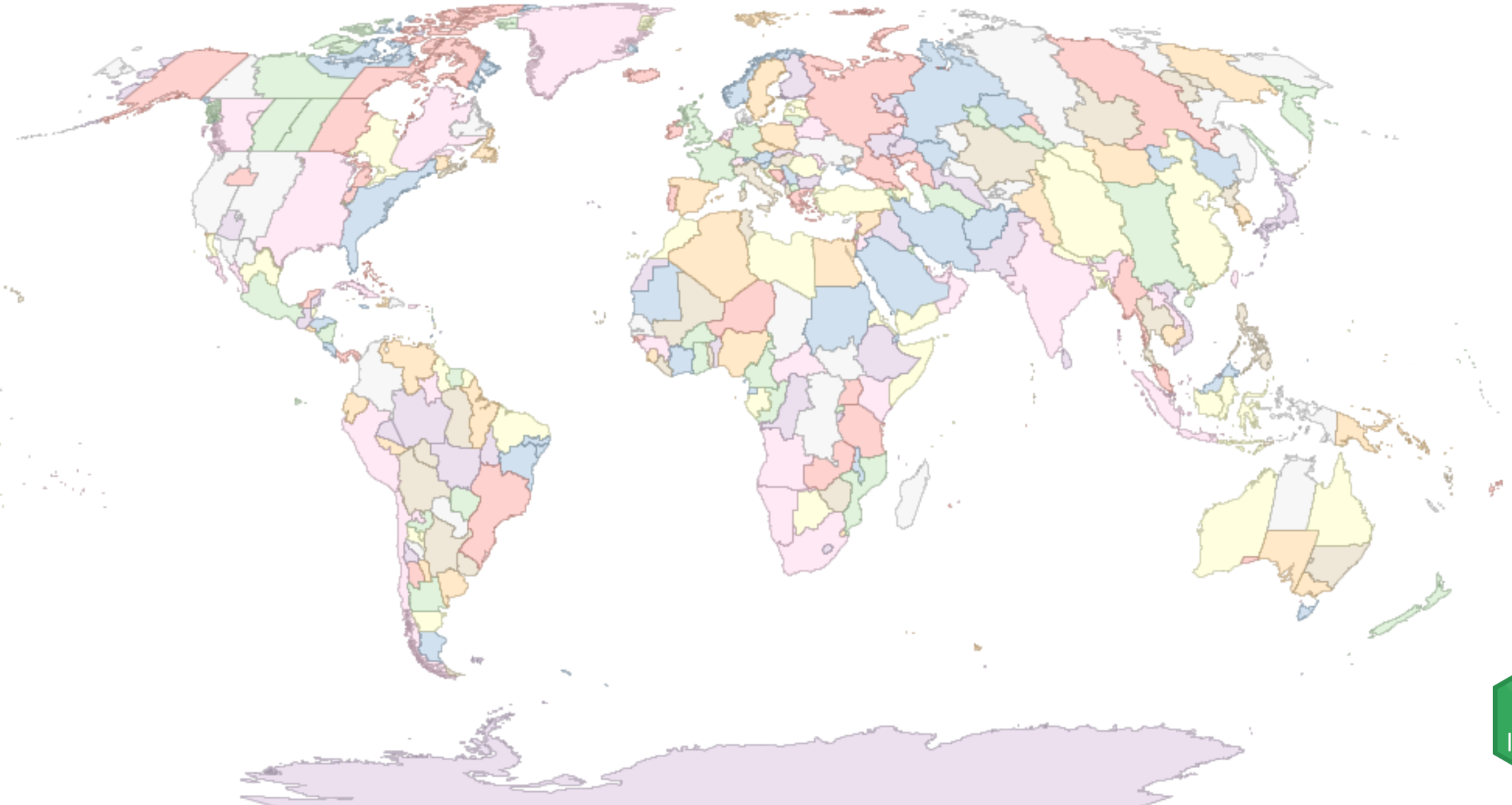
Dealing with time zones

Quiz

How many time zones are there?



589



lubridate

Time zone names

```
OlsonNames()
##   "Africa/Abidjan"
##   "Africa/Accra"
##   "Africa/Addis_Ababa"
##   "Africa/Algiers"
##   "Africa/Asmara"
##   …
##   589 total
```

Quiz

What is the "benchmark" time zone?

Quiz

What is the "benchmark" time zone?

GMT?

UTC

ymd() family

Lubridate functions use UTC by default

```
ymd_hms(…, tz = "UTC")
```

Quiz

```
nyc <- now(tz = "America/New_York")
la <- now(tz = "America/Los_Angeles")
coasts <- c(nyc, la)
```

What is the time zone of coasts?

!

R stores time zones as an attribute, which means that:

1. **every datetime in a vector must share the same time zone**
2. **time zones are frequently dropped**

Quiz

What time should this return? Decide in your group.

```
nyc <- ymd_hms("2017-01-01 00:00:00",
  tz = "America/New_York")

tz(nyc) <- "America/LosAngeles"
```

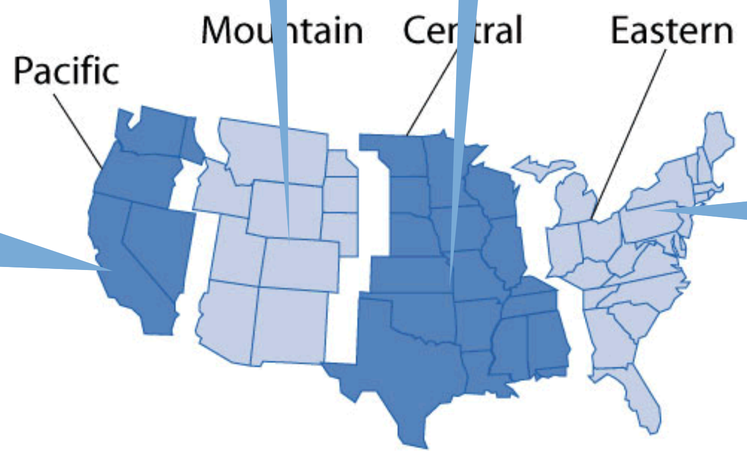
with_tz()

Creates **same instant** with the **new display**

```
with_tz(t,
  tz = "America/Denver")
## "2017-01-05 02:00:00 MST"
```

```
with_tz(t,
  tz = "America/Chicago")
## "2017-01-05 03:00:00 CST"
```

```
t <- now(
  tz = "America/Los_Angeles")
## "2017-01-05 01:00:00 PST"
```




```
with_tz(t,
  tz = "America/New_York")
## "2017-01-05 04:00:00 EST"
```

lubridate

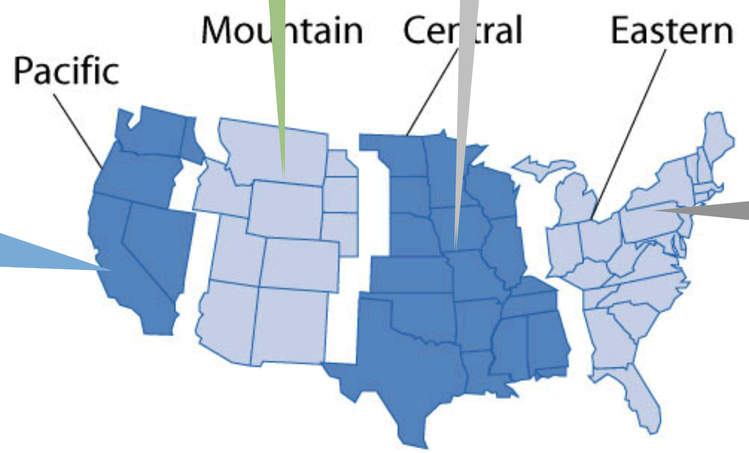
force_tz()

Creates **different instants** with the **same display**

```
force_tz(t,
  tz = "America/Denver")
## "2017-01-05 01:00:00 MST"
```

```
force_tz(t,
  tz = "America/Chicago")
## "2017-01-05 01:00:00 CST"
```

```
t <- now(
  tz = "America/Los_Angeles")
## "2017-01-05 01:00:00 PST"
```




```
force_tz(t,
  tz = "America/New_York")
## "2017-01-05 01:00:00 EST"
```

lubridate

Your Turn

```
this_instant <- now()
```

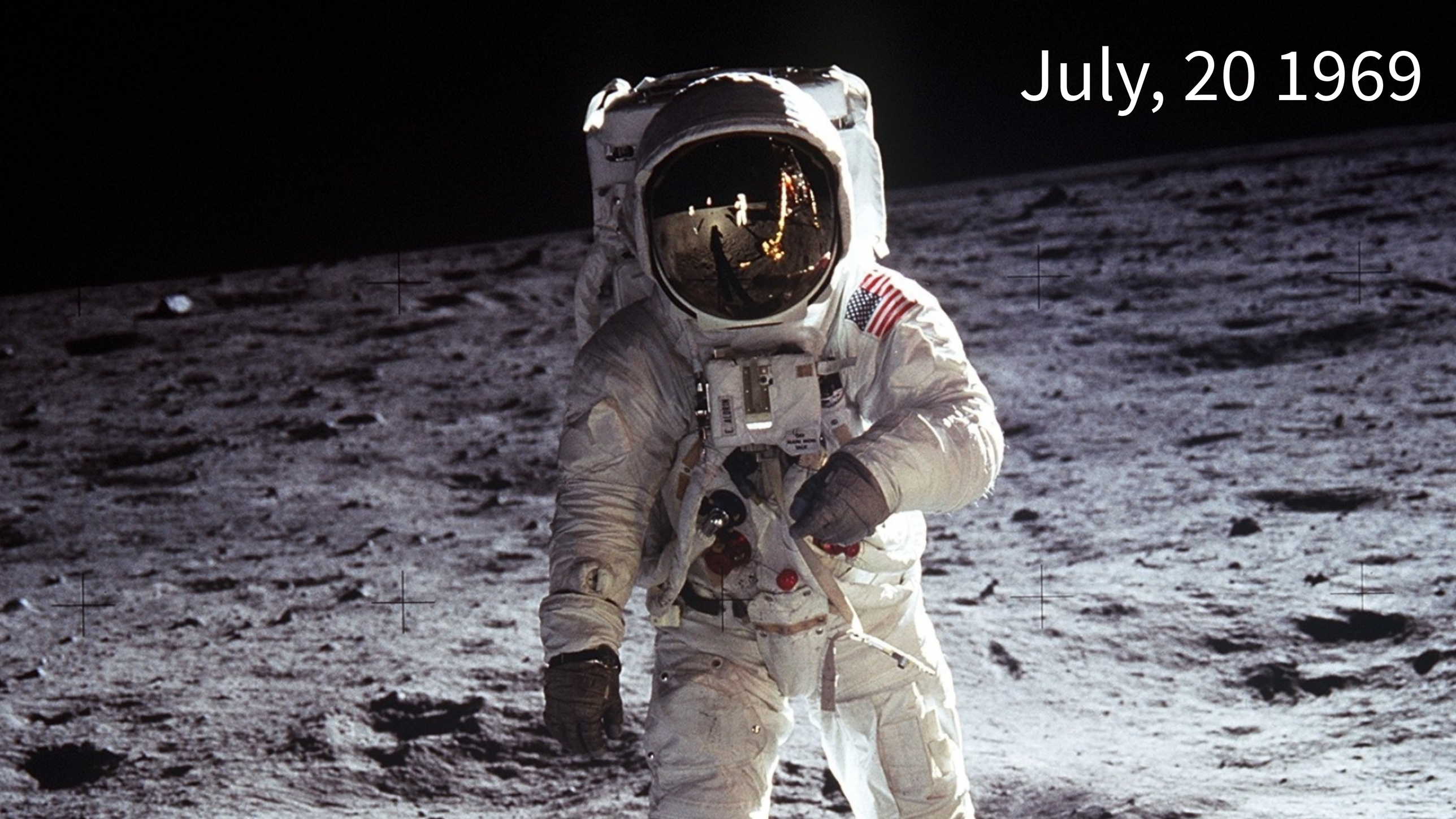
1. What time is it in Honolulu (Pacific/Honolulu) this instant?
2. What will it be here when Honolulu clocks show our time?
3. The **time_hour** column of flights is saved as UTC. Convert it to the correct time zone for New York City. Notice that the hours element of **time_hour** already matches **flights$hour**, which is ostensibly the correct hour in the America/New_York time zone.

```
this_instant %>%
  with_tz("Pacific/Honolulu")
```

```
this_instant %>%
  force_tz("Pacific/Honolulu") %>%
  with_tz("America/New_York")
```

```
flights %>%
  mutate(time_hour = with_tz(time_hour, "America/New_York")) %>%
  select(time_hour)
```

Doing math with dates and times


July, 20 1969


Nov, 25 1992

difftimes

When you subtract two dates in R you get a difftime

```
now() - as.POSIXct(today())
## Time difference of 22.29285 hours
```

But difftimes use inconsistent units (sometimes weeks, days, hours, minutes, or seconds…)

durations

A time span that is **always measured in seconds**

```
as.duration(today() - aladdin)
## "761011200s (~24.11 years)"

as.duration(aladdin - moon_landing)
## "736905600s (~23.35 years)"
```

Plus, a **better display**

Math with dates

To do math with dates add and subtract **periods**

```
weeks(1) - days(3) + hours(2)
## "4d 2H 0M 0S"
date + months(0:11)
## "2017-01-01" "2017-02-01" "2017-03-01"
## "2017-04-01" "2017-05-01" "2017-06-01"
## "2017-07-01" "2017-08-01" "2017-09-01"
## "2017-10-01" "2017-11-01" "2017-12-01"
```

Period constructors

function	makes
years(x)	x years
months(x)	x months
weeks(x)	x weeks
days(x)	x days
hours(x)	x hours
minutes(x)	x minutes
seconds(x)	x seconds
milliseconds(x)	x milliseconds
microseconds(x)	x microseconds
nanoseconds(x)	x nanoseconds
picoseconds(x)	x picoseconds

Quiz

What might worry us here?

```
march_11 + days(1)
## "2016-03-12 08:00:00 EST"
march_12 + days(1)
## "2016-03-13 08:00:00 EDT"
```

periods

A time span that is **measured in clock units**

```
march_11 + days(1)
## "2016-03-12 08:00:00 EST"
march_12 + days(1)
## "2016-03-13 08:00:00 EDT"
```

durations

If absolute time matters, use seconds or a duration

```
march_11 + ddays(1)
## "2016-03-12 08:00:00 EST"
march_12 + ddays(1)
## "2016-03-13 09:00:00 EDT"
```

Duration constructors

function	makes
dyears(x)	x years
dweeks(x)	x weeks
ddays(x)	x days
dhours(x)	x hours
dminutes(x)	x minutes
dseconds(x)	x seconds
dmilliseconds(x)	x milliseconds
dmicroseconds(x)	x microseconds
dnanoseconds(x)	x nanoseconds
dpicoseconds(x)	x picoseconds

Quiz

What should this return? Decide in your group.

```
months(1) / days(1)
```

Math with periods

```
months(1) / days(1)
## estimate only: convert to intervals for accuracy
## 30.4375
```

Quiz

Can we calculate something like this accurately?

"The month between Jan 1, 2017 and Feb 1, 2017" /days(1)

intervals

A time span that is **measured by start and end dates**

```
interval(mdy("Jan 1, 2017"), mdy("Feb 1, 2017"))
## 2017-01-01 UTC--2017-02-01 UTC

mdy("Jan 1, 2017") %--% mdy("Feb 1, 2017")
## 2017-01-01 UTC--2017-02-01 UTC
```

Divide by periods and durations to compute the length

```
month1 <- mdy("Jan 1, 2017") %--% mdy("Feb 1, 2017")

month1 / days(1)
## 31
```

Working with intervals

function	makes
int_start()	access or change the start date
int_end()	access or change the end date
int_diff()	intervals between dates in a vector
int_length()	number of seconds in the interval
int_flip()	switches start and end dates of th interval
int_shift()	shifts an interval by a time span
int_standardize()	flips interval is length is negative
int_aligns(x)	tests whether interval begin or end on same instant
int_overlaps()	tests whether intervals overlap
intersect(), union(), setdiff()	set operations for intervals
%within%	tests whether instants fall within an interval

%within%

Tests whether instants fall within an interval

```
firsts <- ymd("2017-01-01") + months(0:11)
summer <- ymd("2017-06-21") %--% ymd("2017-09-21")


firsts %within% summer
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

Your Turn

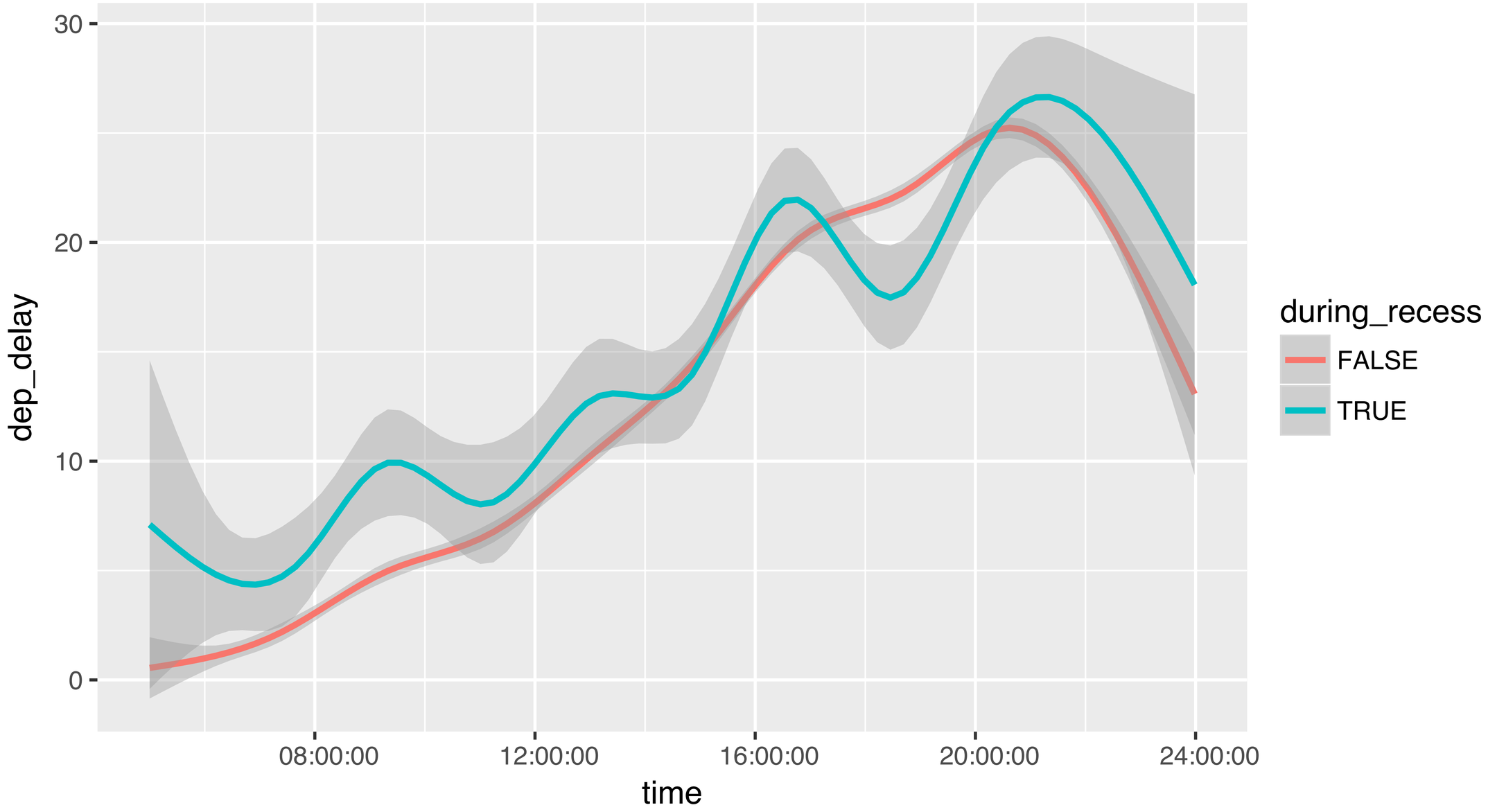
Are flight delays worse over the holidays?

1. NYC Public Schools held winter recess from December 23, 2013 to January 1, 2014. Create an interval that captures the recess.

2. Complete the code below to plot delays over time during recess vs. the rest of the year

```
flights %>%
  mutate(during_recess = ____________________________________,
    time = hms::hms(hour = hour, minute = minute)) %>%
  ggplot(aes(time, dep_delay, color = during_recess)) +
    geom_smooth()
```

```
recess <- ymd("2013-12-23") %--% ymd("2014-01-01")
flights %>%
  mutate(during_recess = time_hour %within% recess,
         time = hms::hms(hour = hour, minute = minute)) %>%
  ggplot(aes(time, dep_delay, color = during_recess)) +
    geom_smooth()
```


30
20
10
0
dep_delay
08:00:00
12:00:00
16:00:00
20:00:00
24:00:00
time
during_recess
FALSE
TRUE

Data types with