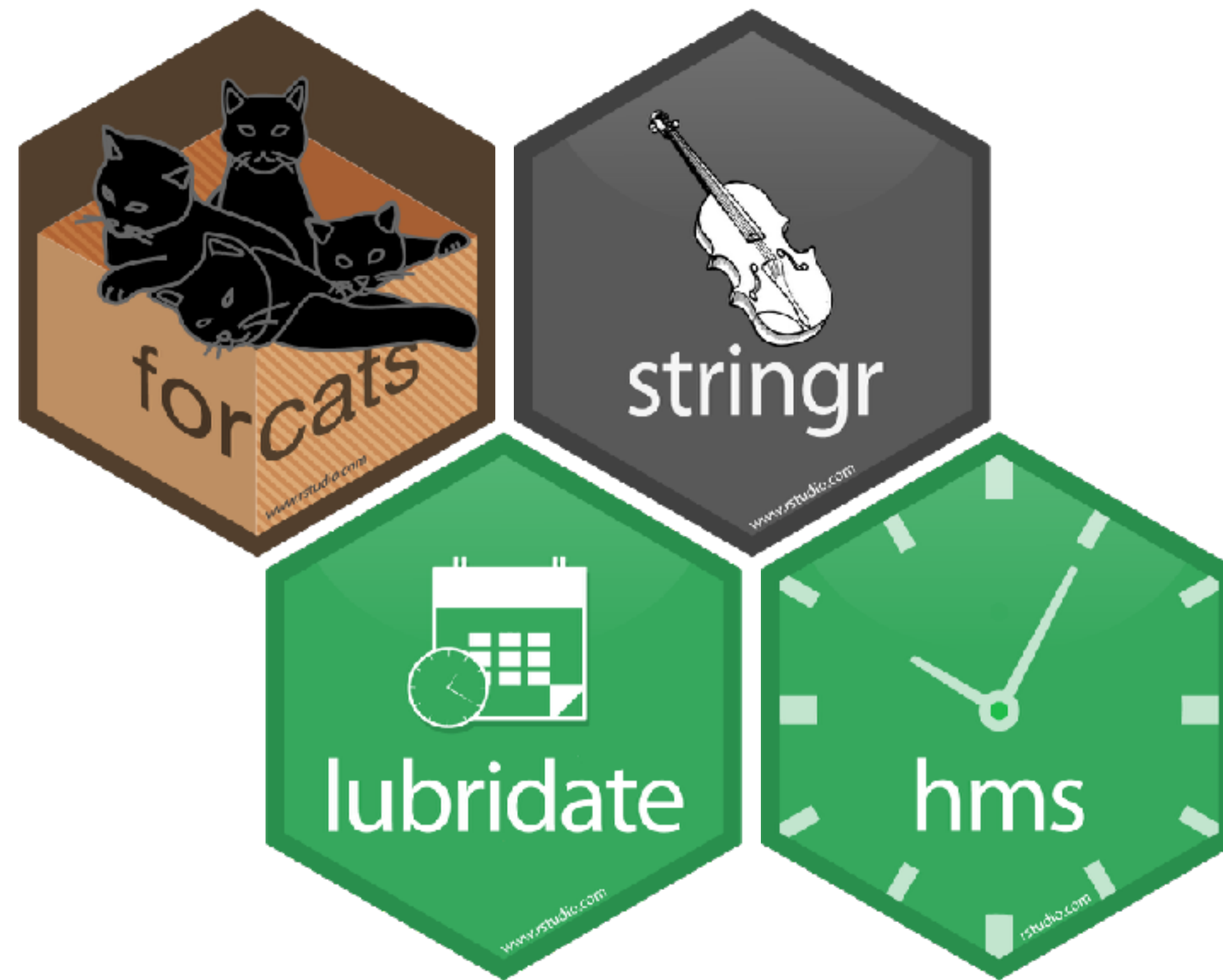


Data Types



gss_cat

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 <int>	marital <fctr>	age <int>	race <fctr>	rincome <fctr>	partyid <fctr>
2000	Never married	26	White	\$8000 to 9999	Ind,near rep
2000	Divorced	48	White	\$8000 to 9999	Not str republican
2000	Widowed	67	White	Not applicable	Independent
2000	Never married	39	White	Not applicable	Ind,near rep
2000	Divorced	25	White	Not applicable	Not str democrat
2000	Married	25	White	\$20000 – 24999	Strong democrat
2000	Never married	36	White	\$25000 or more	Not str republican
2000	Divorced	44	White	\$7000 to 7999	Ind,near dem



Warm-up/Review

Using the data `gss_cat`, find the average hours of tv watched (`tvhours`) for each category of marital status (`marital`).

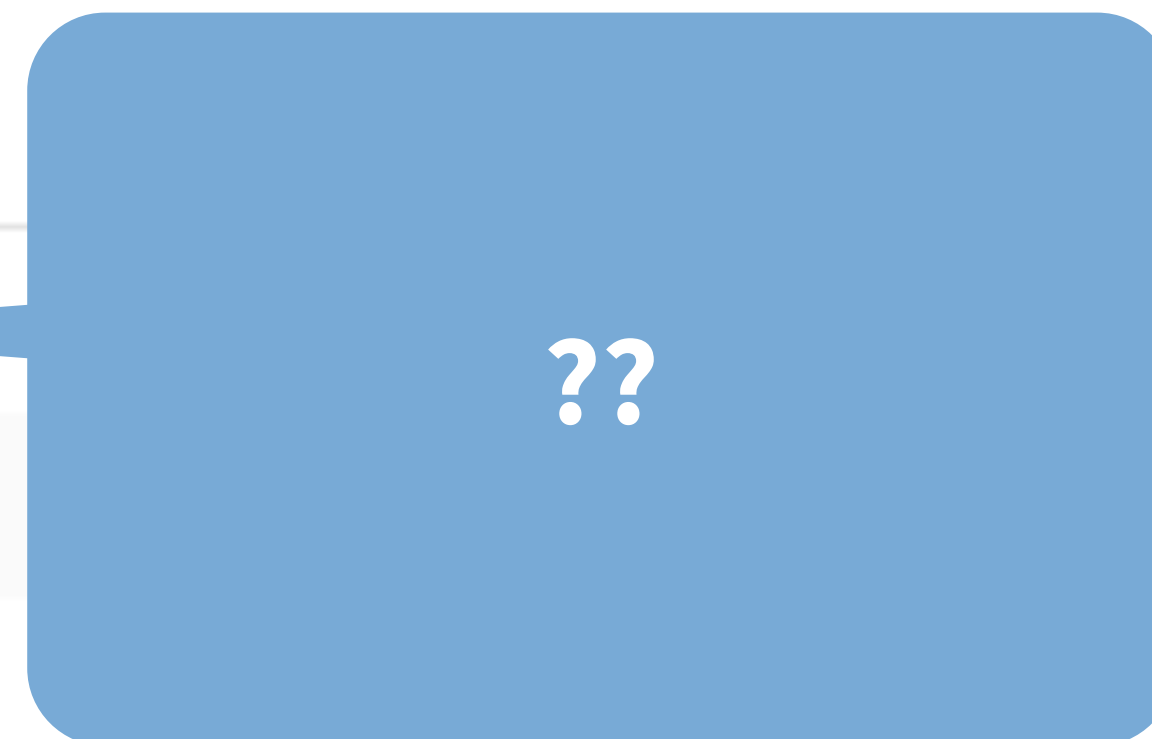
(Don't worry if you get something unexpected, we'll fix it soon)

04:00

```
gss_cat %>%  
  group_by(marital) %>%  
  summarise(avg_tv = mean(tvhours))
```

marital <fctr>	avg_tv <dbl>
No answer	NA
Never married	NA
Separated	NA
Divorced	NA
Widowed	NA
Married	NA

6 rows



Missing values

Missing values propagate

```
NA + 2
```

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

```
NA == NA
```

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

```
mean(c(1, NA, 2))
```

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

is.na()

Returns TRUE if the value is NA

```
filter(gss_cat, tvhours == NA)      # Nope!
```

```
## # A tibble: 0 x 9
```

```
filter(gss_cat, is.na(tvhours))    # Yes!
```

```
# A tibble: 10,146 x 9
```

	year	marital	age	race	rincome	partyid	relig	denom	tvho...
	<int>	<fctr>	<int>	<fctr>	<fctr>	<fctr>	<fctr>	<fctr>	<int>
1	2000	Divorced	48	White	\$8000 ...	Not str...	Prote...	Baptis...	NA
2	2000	Married	25	White	\$20000...	Strong ...	Prote...	Southe...	NA



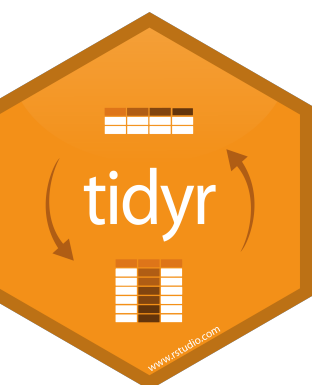
drop_na()

Drops rows that contain NA's in the specified columns.

```
drop_na(x, x2)
```

**data frame to
transform**

**column(s) to
screen for NA's**




drop_na()

Drops rows that contain NA's in the specified columns.

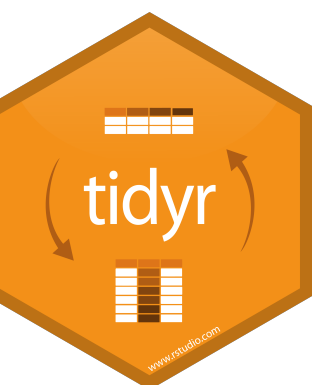
```
drop_na(x, x2)
```

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA



x1	x2
A	1
D	3



```
gss_cat %>%  
  group_by(marital) %>%  
  drop_na(tvhours) %>%  
  summarise(avg_tv = mean(tvhours))
```

marital <fctr>	avg_tv <dbl>
No answer	2.555556
Never married	3.105175
Separated	3.549618
Divorced	3.085407
Widowed	3.912000
Married	2.650425

6 rows

Your Turn 1

What kind of object is the `marital` variable?

Brainstorm with your neighbor, all the things you know about that kind of object.

Factors

It's not a number!

It's not ordered (maybe?)

R's way to handle categorical variables

Distinct values - levels

Factors



In R4DS
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
```



factors

Stored as an integer vector with a levels attribute

```
unclass(eyes)
## 1 3 3
## attr(,"levels")
## "blue" "brown" "green"
```



forcats



Simple functions for working with factors.

```
library(tidyverse)
```

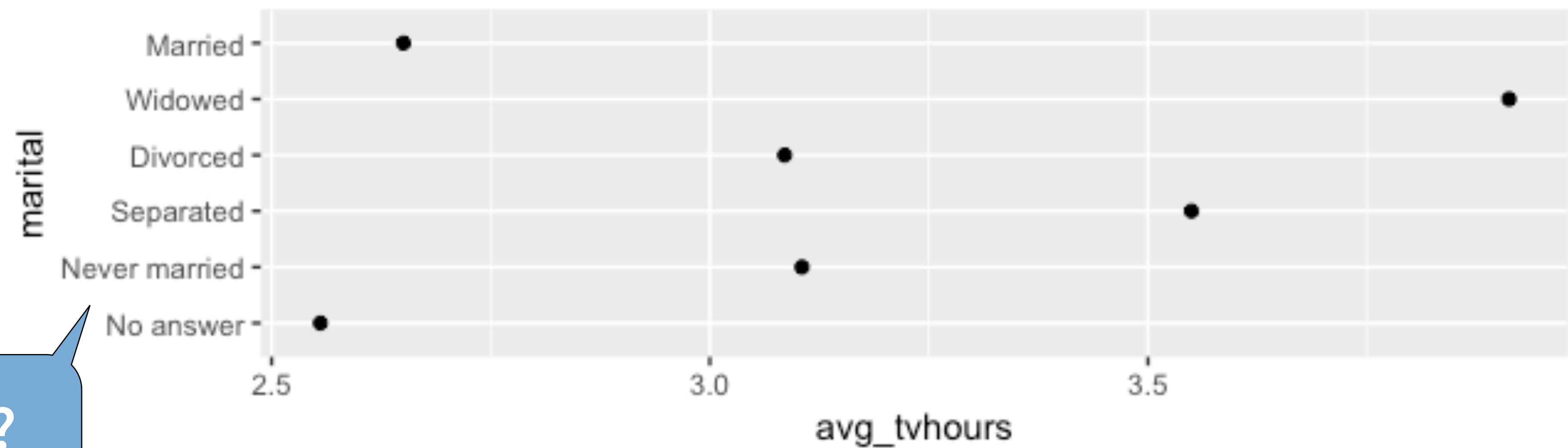
Your Turn 2

Fix your summary of average hours of tv watched (`tvhours`) by marital status (`marital`), to ignore missing values,

then create a **plot** to examine the results.

04:00

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



Why this order?

levels()

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

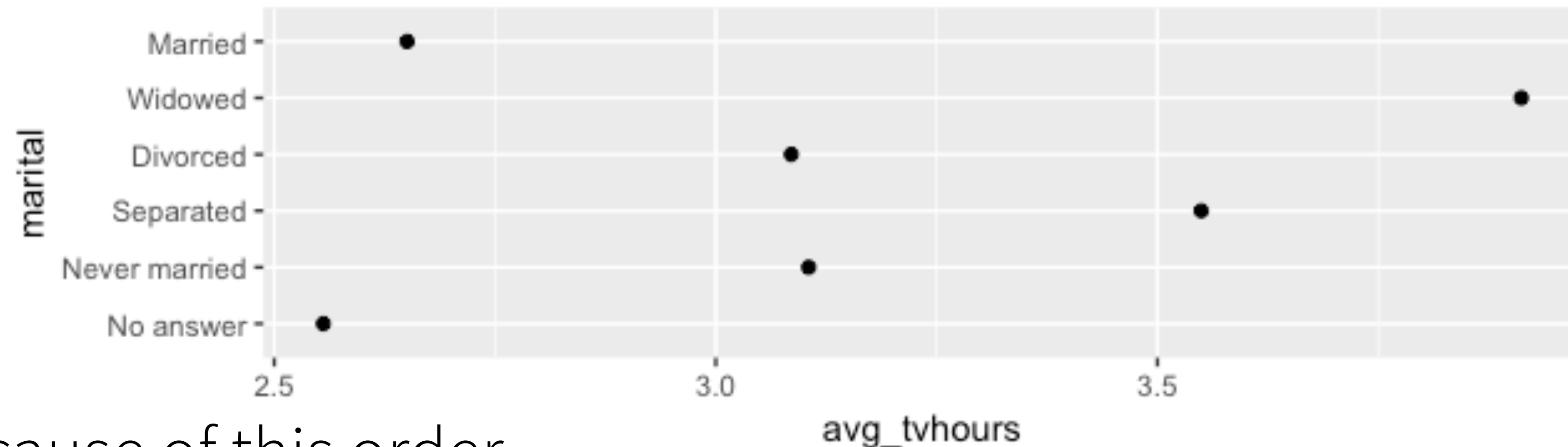
```
levels(eyes)
# [1] "blue" "brown" "green"
```

`x %>% pull(y)`
same as
`x$y`

```
gss_cat %>% pull(marital) %>% levels()
# [1] "No answer" "Never married" "Separated"
# [4] "Divorced" "Widowed" "Married"
```



Why this order?



Because of this order

```
gss_cat %>% pull(marital) %>% levels()  
# [1] "No answer"      "Never married" "Separated"  
# [4] "Divorced"       "Widowed"      "Married"
```

Most useful skills

1. Reorder the levels
2. Manipulate the levels



Reordering levels

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)

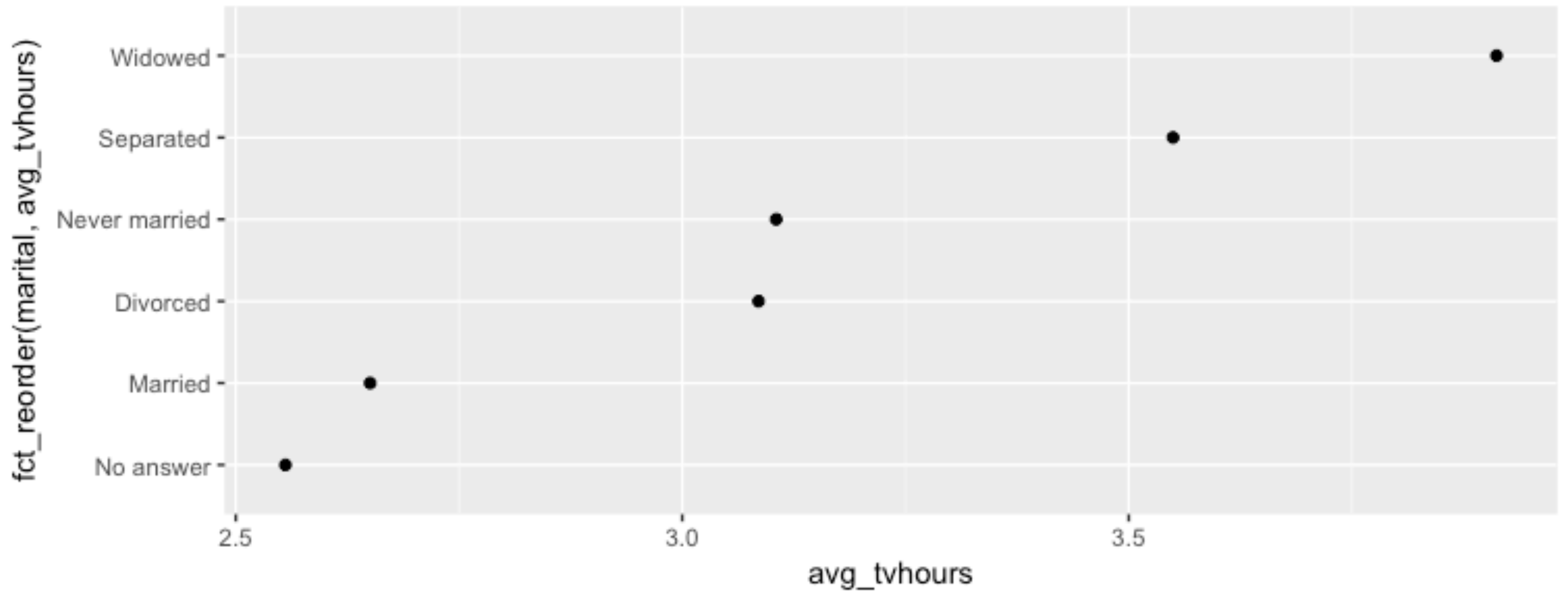
**function to
reorder by**
(in conjunction
with x)

**put in descending
order?**




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





Your Turn 3

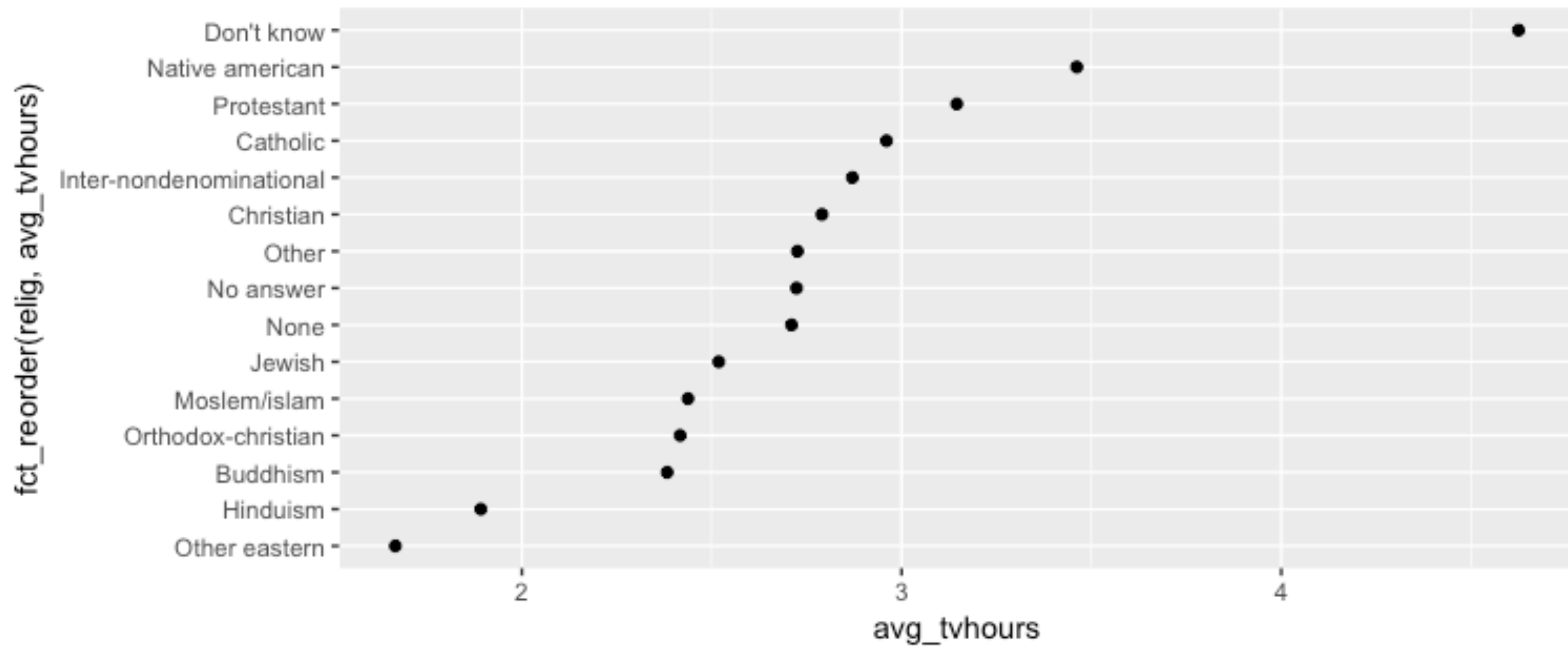
Fill in the blanks (____) to explore the average hours of tv watched by religion.

```
gss_cat %>%  
  drop_na(____) %>%  
  group_by(____) %>%  
  summarise(____) %>%  
  ggplot() +  
    geom_point(mapping = aes(x = ____, y = ____))
```

03:00

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





Other reordering functions

Values and level labels are unchanged

`fct_shuffle()` Randomize order

`fct_relevel()` Order "by hand"

`fct_infreq()` Order from most to least frequent

`fct_inorder()` Order from first to last observed

`fct_rev()` Reverse the current order

`fct_shift()` Shift the order by 1

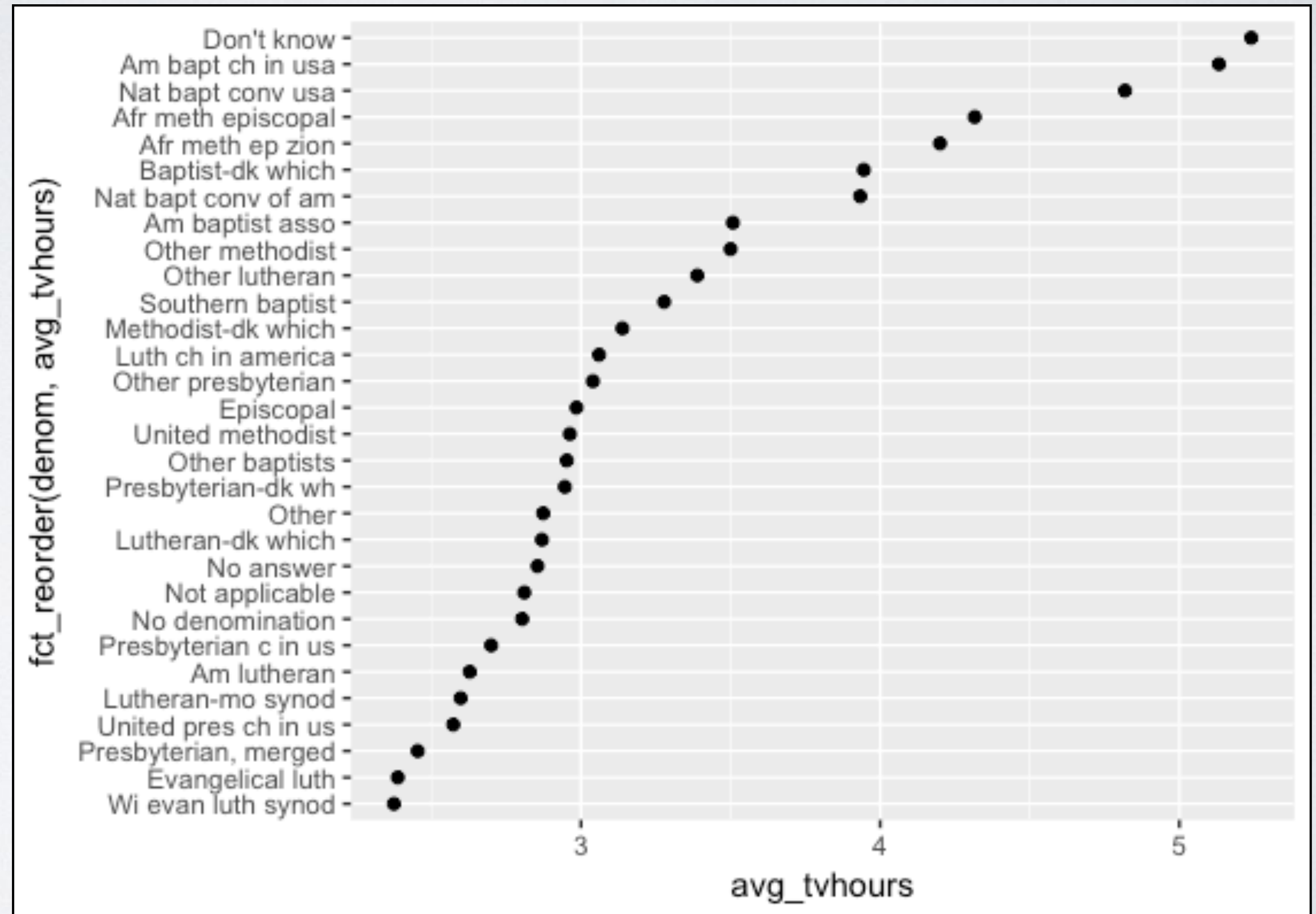
All forcats functions start
with `fct_`



Manipulating levels

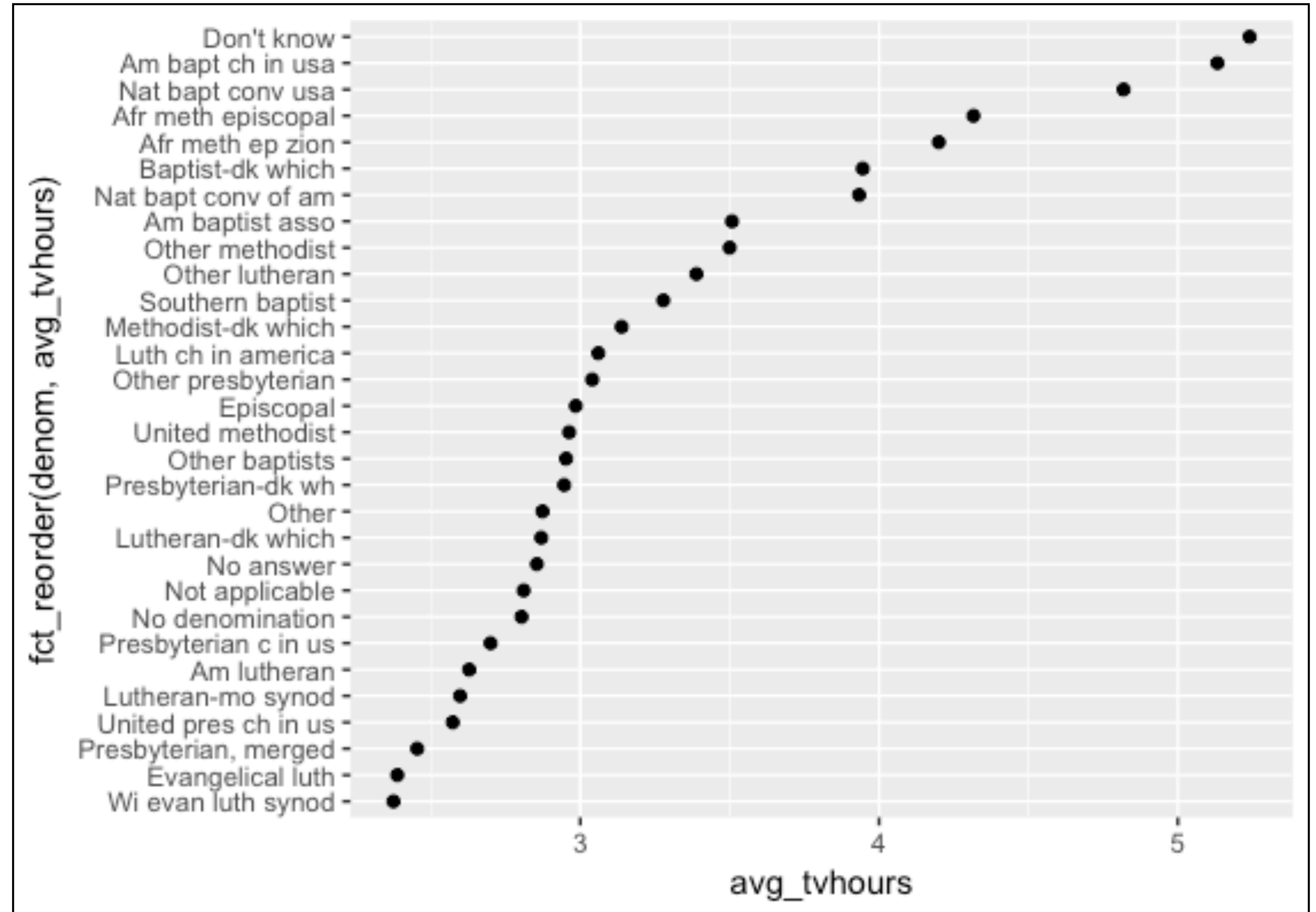
Quiz

Why is this
plot not
very
useful?



Too many
categories

Poorly labelled



First 10 rows

Obs.	denom
1	Southern baptist
2	Baptist-dk which
3	No denomination
4	Not applicable
5	Not applicable
6	Southern baptist
7	Not applicable
8	Lutheran-mo synod
9	Other
10	Southern baptist
...	...

Relabel levels

denom
Baptist - Southern
Baptist - Don't know
No denomination
Not applicable
Not applicable
Baptist - Southern
Not applicable
Lutheran - Missouri Synod
Other
Baptist - Southern
...

Reduce levels

denom
Baptist
Baptist
None
None
None
Baptist
None
Lutheran
Other
Baptist
...

Level manipulation functions

Values change to match levels

Relabel	fct_recode()	Relabel levels "by hand"
	fct_anon()	Anonymize levels
Reduce	fct_relabel()	Relabel using a function
	fct_collapse()	Collapse levels "by hand"
	fct_lump()	Lump levels with small counts together
	fct_other()	Replace levels with "Other"



fct_recode()

Changes values of levels

```
fct_recode(f, ...)
```

**factor with
levels**

**new level = old level
pairs**



```
gss_cat %>%  
  pull(denom) %>%  
  levels()
```

```
[1] "No answer"      "Don't know"  
[3] "No denomination" "Other"  
[5] "Episcopal"      "Presbyterian-dk wh"  
[7] "Presbyterian, merged" "Other presbyterian"  
[9] "United pres ch in us" "Presbyterian c in us"  
[11] "Lutheran-dk which" "Evangelical luth"  
[13] "Other lutheran"    "Wi evan luth synod"  
[15] "Lutheran-mo synod" "Luth ch in america"  
[17] "Am lutheran"       "Methodist-dk which"  
[19] "Other methodist"  "United methodist"  
[21] "Afr meth ep zion"  "Afr meth episcopal"  
[23] "Baptist-dk which" "Other baptists"  
[25] "Southern baptist"  "Nat bapt conv usa"  
[27] "Nat bapt conv of am" "Am bapt ch in usa"  
[29] "Am baptist asso"   "Not applicable"
```



```
gss_cat %>%
  mutate(denom = fct_recode(denom,
    "Baptist - Southern" = "Southern baptist")
  ) %>%
  pull(denom) %>%
  levels()
```

[1] "No answer"	"Don't know"
[3] "No denomination"	"Other"
[5] "Episcopal"	"Presbyterian-dk wh"
[7] "Presbyterian, merged"	"Other presbyterian"
[9] "United pres ch in us"	"Presbyterian c in us"
[11] "Lutheran-dk which"	"Evangelical luth"
[13] "Other lutheran"	"Wi evan luth synod"
[15] "Lutheran-mo synod"	"Luth ch in america"
[17] "Am lutheran"	"Methodist-dk which"
[19] "Other methodist"	"United methodist"
[21] "Afr meth ep zion"	"Afr meth episcopal"
[23] "Baptist-dk which"	"Other baptists"
[25] "Baptist - Southern"	"Nat bapt conv usa"
[27] "Nat bapt conv of am"	"Am bapt ch in usa"
[29] "Am baptist asso"	"Not applicable"



**factor with
levels**

```
gss_cat %>%  
  mutate(denom = fct_recode(denom,  
    "Baptist - Southern" = "Southern baptist")  
  ) %>%  
  pull(denom) %>%  
  levels()
```

**new level = old level
pairs**

[1]	"No answer"	"Don't know"
[3]	"No denomination"	"Other"
[5]	"Episcopal"	"Presbyterian-dk wh"
[7]	"Presbyterian, merged"	"Other presbyterian"
[9]	"United pres ch in us"	"Presbyterian c in us"
[11]	"Lutheran-dk which"	"Evangelical luth"
[13]	"Other lutheran"	"Wi evan luth synod"
[15]	"Lutheran-mo synod"	"Luth ch in america"
[17]	"Am lutheran"	"Methodist-dk which"
[19]	"Other methodist"	"United methodist"
[21]	"Afr meth ep zion"	"Afr meth episcopal"
[23]	"Baptist-dk which"	"Other baptists"
[25]	"Baptist - Southern"	"Nat bapt conv usa"
[27]	"Nat bapt conv of am"	"Am bapt ch in usa"
[29]	"Am baptist asso"	"Not applicable"



Your Turn 4

Edit the code to also relabel some other Baptist denominations:

- "Baptist-dk which"
- "Other baptists"

02:00

```
gss_cat %>%  
  mutate(denom = fct_recode(denom,  
    "Baptist - Southern" = "Southern baptist",  
    "Baptist - Don't know" = "Baptist-dk which",  
    "Baptist - Other" = "Other baptists")  
  ) %>%  
  pull(denom) %>%  
  levels()
```

```
gss_cat %>%
```

```
  mutate(denom = fct_recode(denom,
```

```
    "Baptist - Southern" = "Southern baptist",
```

```
    "Baptist-dk which" = "Baptist - Don't know",
```

```
    "Baptist - Other" = "Other baptists")
```

```
)
```

```
# Unknown levels in `f`: Baptist - Don't know
```

Common
mistake

Whoops, around
the wrong way!

Message, but no
warning or error!

Your Turn 5

What does the function `detect_denom()` do?

02:00

```
gss_cat %>% pull(denom) %>% levels()
```

```
[1] "No answer"          "Don't know"
[3] "No denomination"    "Other"
[5] "Episcopal"          "Presbyterian-dk wh"
[7] "Presbyterian, merged" "Other presbyterian"
[9] "United pres ch in us" "Presbyterian c in us"
[11] "Lutheran-dk which"   "Evangelical luth"
[13] "Other lutheran"      "Wi evan luth synod"
[15] "Lutheran-mo synod"   "Luth ch in america"
[17] "Am lutheran"         "Methodist-dk which"
[19] "Other methodist"     "United methodist"
[21] "Afr meth ep zion"    "Afr meth episcopal"
[23] "Baptist-dk which"    "Other baptists"
[25] "Southern baptist"    "Nat bapt conv usa"
[27] "Nat bapt conv of am" "Am bapt ch in usa"
[29] "Am baptist asso"     "Not applicable"
```



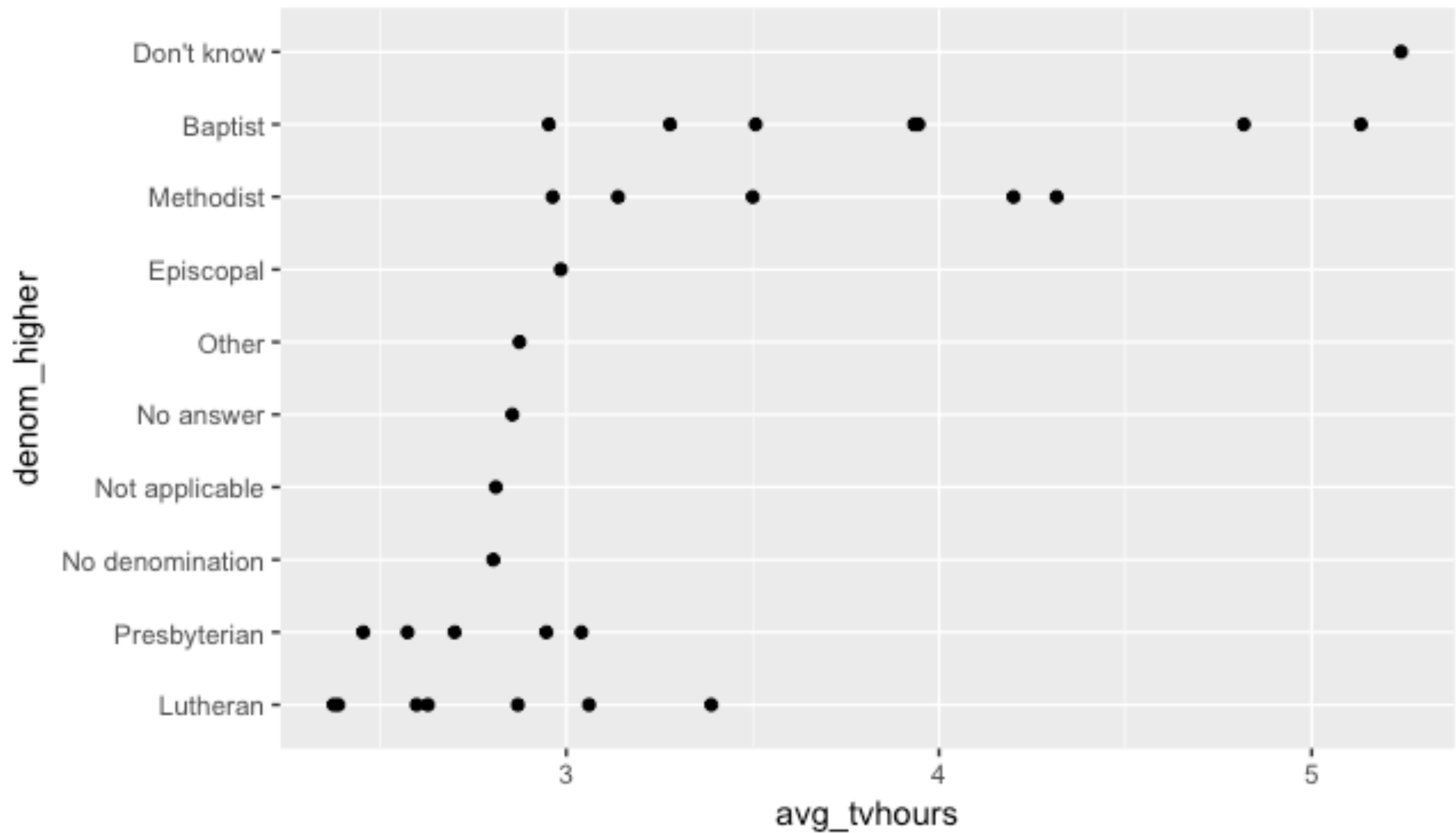
```
gss_cat %>% pull(denom) %>% levels() %>% detect_denom()
```

[1]	"No answer"	"Don't know"
[3]	"No denomination"	"Other"
[5]	"Episcopal"	"Presbyterian"
[7]	"Presbyterian"	"Presbyterian"
[9]	"Presbyterian"	"Presbyterian"
[11]	"Lutheran"	"Lutheran"
[13]	"Lutheran"	"Lutheran"
[15]	"Lutheran"	"Lutheran"
[17]	"Lutheran"	"Methodist"
[19]	"Methodist"	"Methodist"
[21]	"Methodist"	"Methodist"
[23]	"Baptist"	"Baptist"
[25]	"Baptist"	"Baptist"
[27]	"Baptist"	"Baptist"
[29]	"Baptist"	"Not applicable"

converts to a
higher level
grouping

Use with `fct_relabel()` to collapse levels

```
gss_cat %>%  
  drop_na(tvhours) %>%  
  mutate(denom_higher = fct_relabel(denom, detect_denom) %>%  
    fct_reorder(tvhours, mean)) %>%  
  group_by(denom_higher, denom) %>%  
  summarise(avg_tvhours = mean(tvhours)) %>%  
  ggplot() +  
    geom_point(mapping = aes(x = avg_tvhours,  
      y = denom_higher))
```



Strings



In R4DS
Strings

(character) strings

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

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


stringr



Simple functions for working with character **strings**.

```
library(tidyverse)
```

Most useful skills

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

```
detect_denom <- function(x){  
  case_when(  
    str_detect(x, "[Bb]ap") ~ "Baptist",  
    str_detect(x, "[Pp]res") ~ "Presbyterian",  
    str_detect(x, "[Ll]uth") ~ "Lutheran",  
    str_detect(x, "[Mm]eth") ~ "Methodist",  
    TRUE ~ x  
  )  
}
```

```
detect_denom <- function(x){  
  case_when(  
    str_detect(x, "[Bb]ap") ~ "Baptist",  
    str_detect(x, "[Pp]res") ~ "Presbyterian",  
    str_detect(x, "[Ll]uth") ~ "Lutheran",  
    str_detect(x, "[Mm]eth") ~ "Methodist",  
    TRUE ~ x  
  )  
}
```

str_detect()

Test whether a pattern appears within a string.

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

stringr
functions start
with str_

vector of strings
to find patterns in

a string that
represents a regular
expression

Returns **TRUE** or **FALSE**



Your Turn 6

```
strings <- c("Apple", "Pineapple", "Orange")
```

With your neighbor, **predict** what these might return:

```
str_detect(strings, pattern = "pp")
```

```
str_detect(strings, pattern = "apple")
```

```
str_detect(strings, pattern = "[Aa]pple")
```

Then **run** them!

A digital timer display showing the time 02:00 in a black, segmented font on a white background.

```
str_detect(strings, pattern = "pp")
```

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

Apple
Pineapple
Orange

```
str_detect(strings, pattern = "apple")
```

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

Apple
Pineapple
Orange

```
str_detect(strings, pattern = "[Aa]pple")
```

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

Apple
Pineapple
Orange

"A" or "a"

(regular expression)

```
help(package = stringr)
```

Simple, Consistent Wrappers for Common String Operations



Documentation for package 'stringr' version 1.2.0

- [DESCRIPTION file.](#)
- [User guides, package vignettes and other documentation.](#)

Help Pages

[boundary](#)

Control matching behaviour with modifier functions.

[case](#)

Convert case of a string.

[coll](#)

Control matching behaviour with modifier functions.

[fixed](#)

Control matching behaviour with modifier functions.

[fruit](#)

Sample character vectors for practicing string manipulations.

[invert_match](#)

Switch location of matches to location of non-matches.

[modifiers](#)

Control matching behaviour with modifier functions.

[regex](#)

Control matching behaviour with modifier functions.

[sequences](#)

Sample character vectors for practicing string manipulations.



Dates and times



In R4DS

Dates and Times

Most useful skills

1. Creating dates/times (i.e. *parsing*)
2. Access parts of a date
3. Deal with time zones
4. Do math with instants and time spans

Creating 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(), hm(), ms()	Period



Your Turn 7

For each of the following formats (of the same date), pick the right `ymd()` function to parse them:

- "2018 Feb 01"
- "2-1-18"
- "01/02/2018"




```
ymd("2018 Feb 02")
```

```
# [1] "2018-02-02"
```

```
mdy("2-1-18")
```

```
# [1] "2018-02-01"
```

```
dmy("01/02/2018")
```

```
# [1] "2018-02-01"
```

Accessing components

Accessing components

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

```
date <- ymd("2018-02-01")  
year(date)  
## 2018
```

Accessing 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("2018-02-01"))  
## 5  
  
wday(ymd("2018-02-01"), label = TRUE)  
# [1] Thu  
# Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat  
  
wday(ymd("2018-02-01"), label = TRUE, abbr = FALSE)  
# [1] Thursday  
# 7 Levels: Sunday < Monday < Tuesday < Wednesday < ... < Saturday
```



births

Two variables from yesterday's data



date <date>	births <int>
1994-01-01	8096
1994-01-02	7772
1994-01-03	10142
1994-01-04	11248
1994-01-05	11053
1994-01-06	11406
1994-01-07	11251
1994-01-08	8653
1994-01-09	7910
1994-01-10	10498

1-10 of 3,652 rows

Previous 1 2 3 4 5 6 ... 100 Next

Your Turn 8

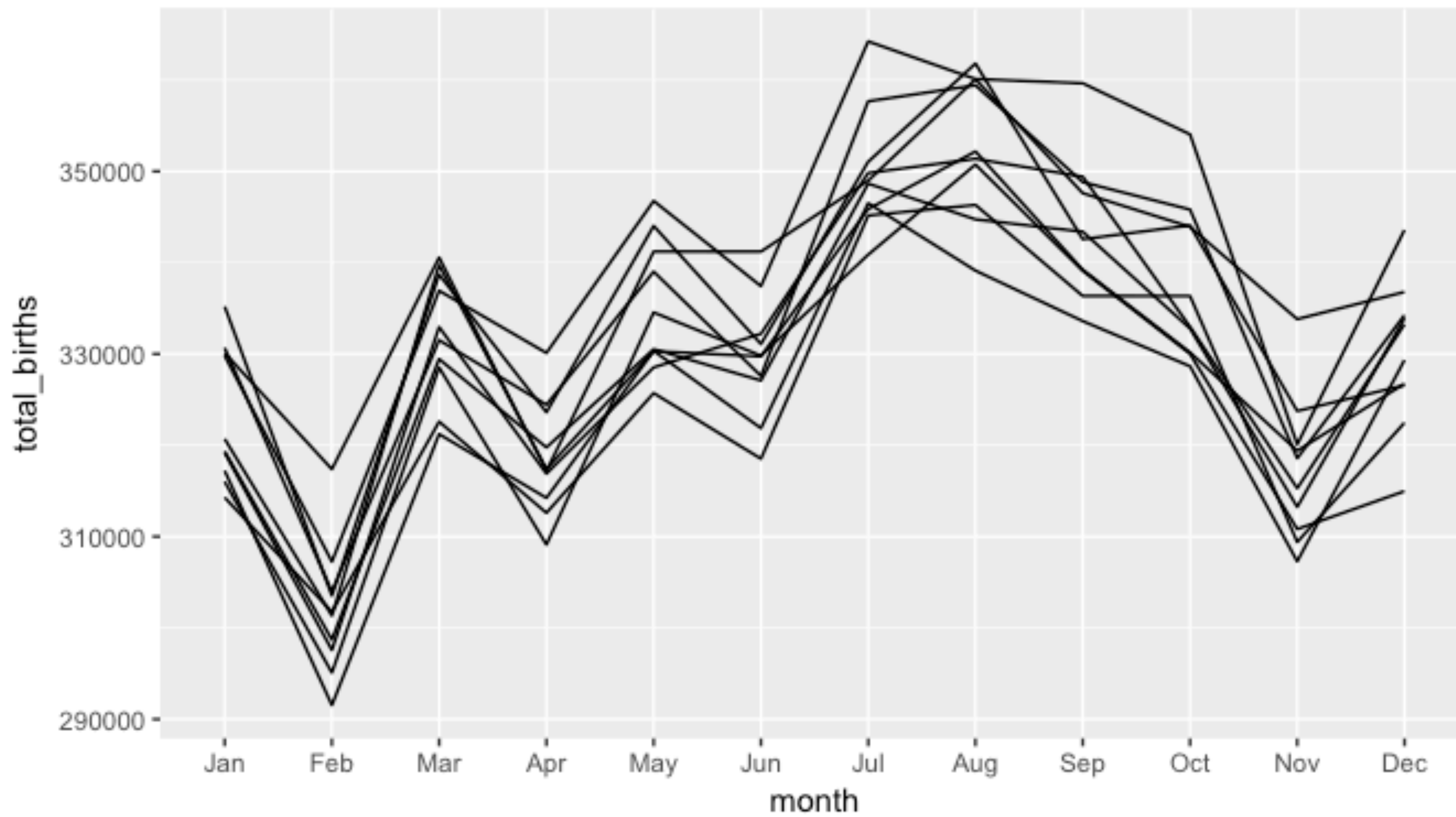
Fill in the blanks to:

- Extract the month from **date**.
- Extract the year from **date**.
- Find the total births for each year/month.
- Plot the results as a line chart.

05:00

```
births %>%  
  mutate(year = year(date),  
         month = month(date, label = TRUE)) %>%  
  group_by(year, month) %>%  
  summarise(total_births = sum(births)) %>%  
  ggplot() +  
    geom_line(aes(x = month, y = total_births, group = year))
```





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
```



Data Types

