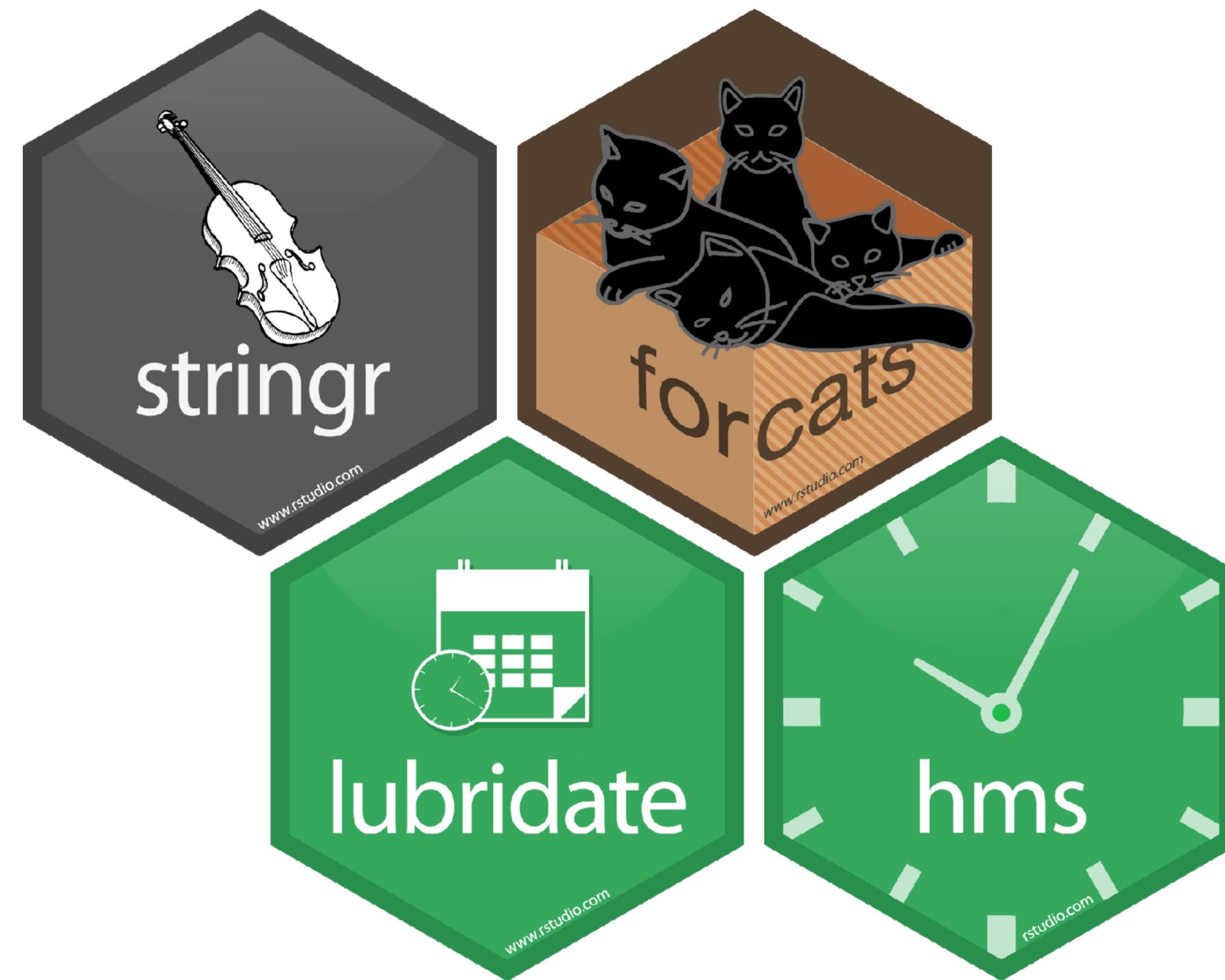


Data types with

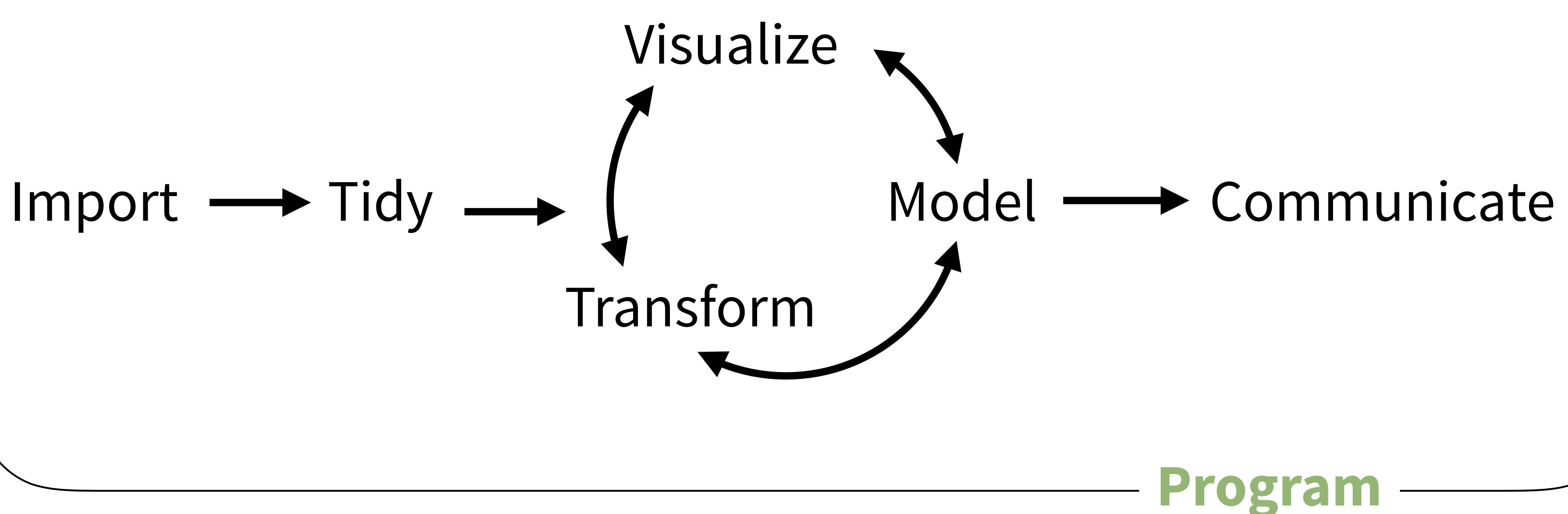


Quiz

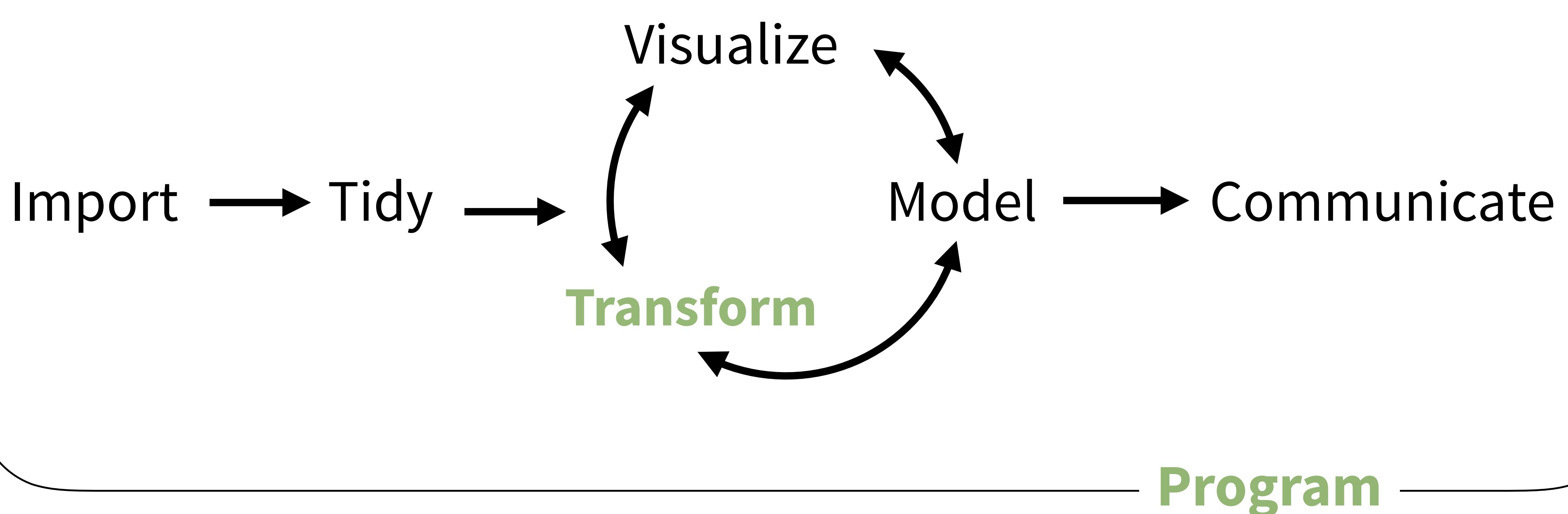
What types of data are in this data set?

	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



Logicals



Logicals

R's data type for boolean values (i.e. TRUE and FALSE).

```
typeof(TRUE)  
## "logical"
```

```
typeof(FALSE)  
## "logical"
```

```
typeof(c(TRUE, TRUE, FALSE))  
## "logical"
```



```
flights %>%  
  mutate(delayed = arr_delay > 0) %>%  
  select(arr_delay, delayed)
```

arr_delay <dbl>	delayed <lgl>
11	TRUE
20	TRUE
33	TRUE
-18	FALSE
-25	FALSE
12	TRUE
19	TRUE
-14	FALSE
-8	FALSE
8	TRUE



Warm Up

Did you fly here?

Did your flight arrive late?

```
flights %>%  
  mutate(delayed = arr_delay > 0) %>%  
  select(arr_delay, delayed)
```

arr_delay <dbl>	delayed <lgl>
11	TRUE
20	TRUE
33	TRUE
-18	FALSE
-25	FALSE
12	TRUE
19	TRUE
-14	FALSE
-8	FALSE
8	TRUE

Can we compute
the proportion of
NYC flights that
arrived late?



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 1

Use flights to create **delayed**, a variable that displays whether a flight was delayed (`arr_delay > 0`).

Then, remove all rows that contain an NA in **delayed**.

Finally, create a summary table that shows:

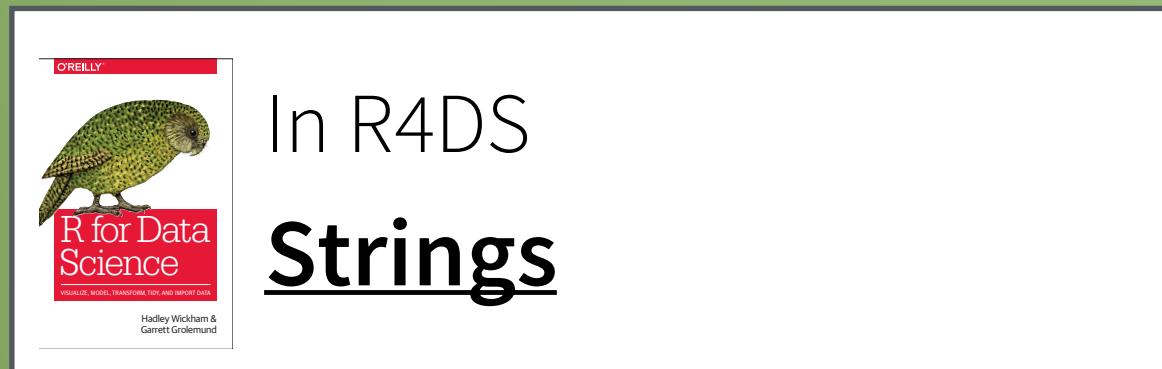
1. How many flights were delayed
2. What proportion of flights were delayed



```
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



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."
```



Warm Up

Decide in your group:

Are boys names or girls names more likely to end in a vowel?



babynames

year	sex	name	n	prop
<dbl>	<chr>	<chr>	<int>	<dbl>
1880	F	Mary	7065	7.238433e-02
1880	F	Anna	2604	2.667923e-02
1880	F	Emma		
1880	F	Elizabeth		
1880	F	Minnie		
1880	F	Margaret		
1880	F	Ida		
1880	F	Alice	1414	1.448711e-02
1880	F	Bertha	1320	1.352404e-02
1880	F	Sarah	1288	1.319618e-02

1-10 of 1,858,689 rows

Previous

1

2

3

4

5

6

...

100 Next

How can we build the proportion of boys and girls whose name ends in a vowel?



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")
```

str_sub()

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

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

**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 2

In your group, fill in the blanks to:

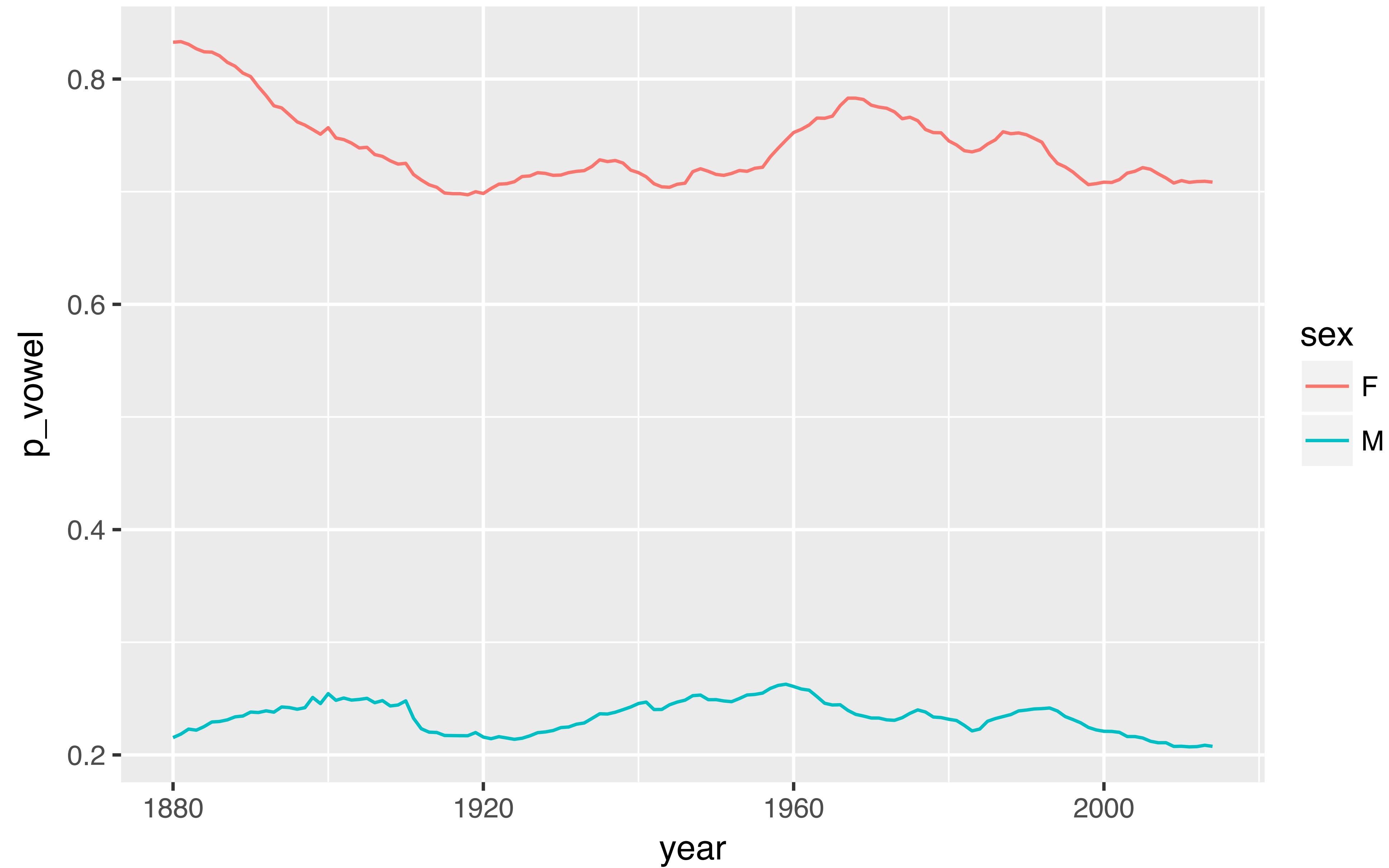
1. Isolate the last letter of every name
2. and create a logical variable that displays whether the last letter is one of "a", "e", "i", "o", "u", or "y".
3. Use a weighted mean to calculate the proportion of children whose name ends in a vowel (by year and sex)
4. and then display the results as a line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

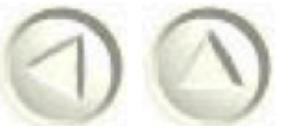


Proportion of names that end in a vowel



```
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.

[sentences](#)

Sample character vectors for practicing string manipulations.



Factors



R

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.

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



Warm Up

Decide in your group:

Which religions watch the least TV?

Do married people watch more or less TV than single people?



gss_cat

```
library(forcats)  
gss_cat
```

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

tvhours	marital	age	race	partyid	relig
<int>	<fctr>	<int>	<fctr>	<fctr>	<fctr>
12	Never married	26	White	Ind,near rep	Protestant
NA	Divorced	48	White	Not str republican	Protestant
2	Widowed	67	White	Independent	Protestant
4	Never married	39	White	Ind,near rep	Orthodox-christian
1	Divorced	25	White	Not str democrat	None
NA	Married	25	White	Strong democrat	Protestant
3	Never married	36	White	Not str republican	Christian
NA	Divorced	44	White	Ind,near dem	Protestant

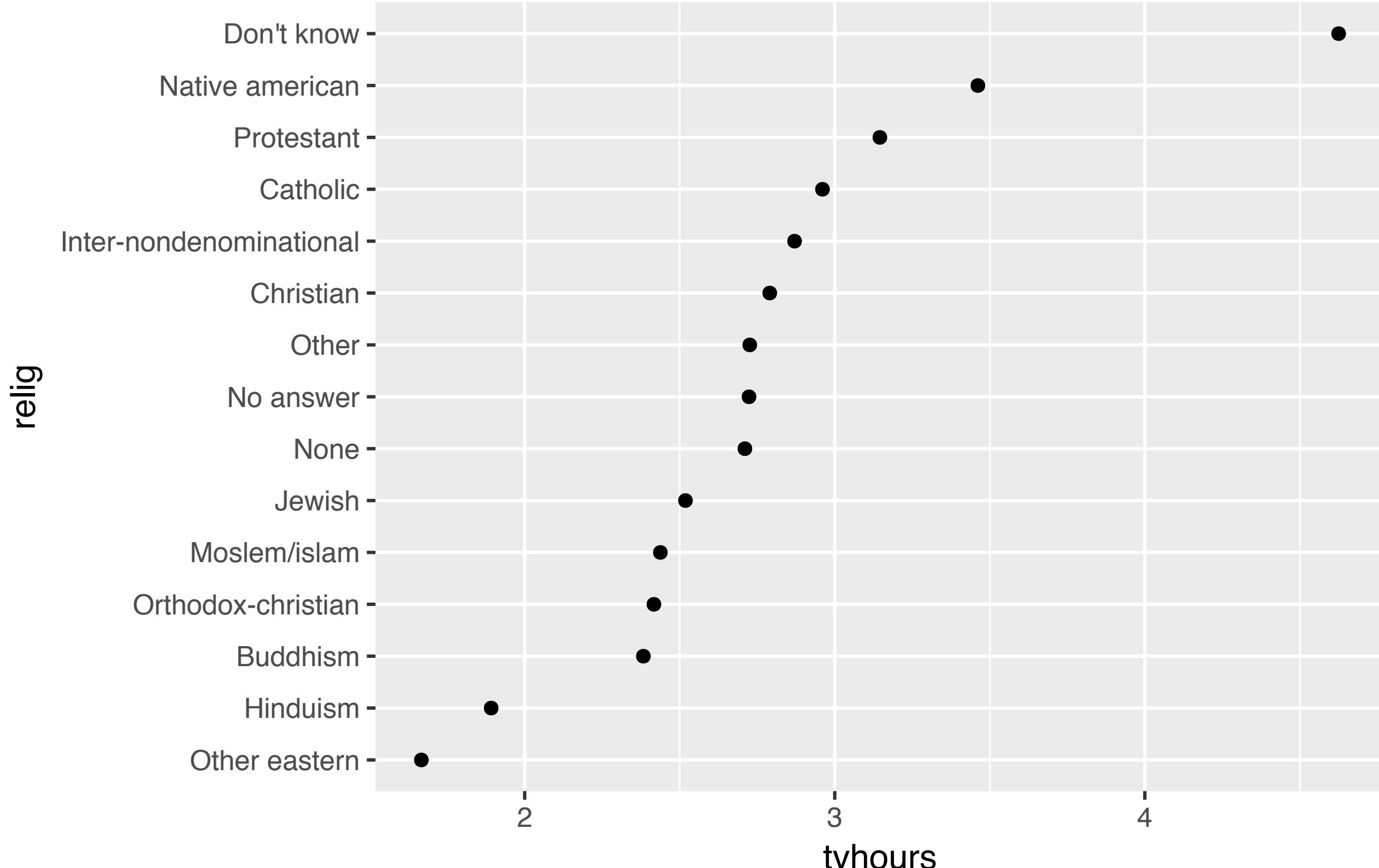
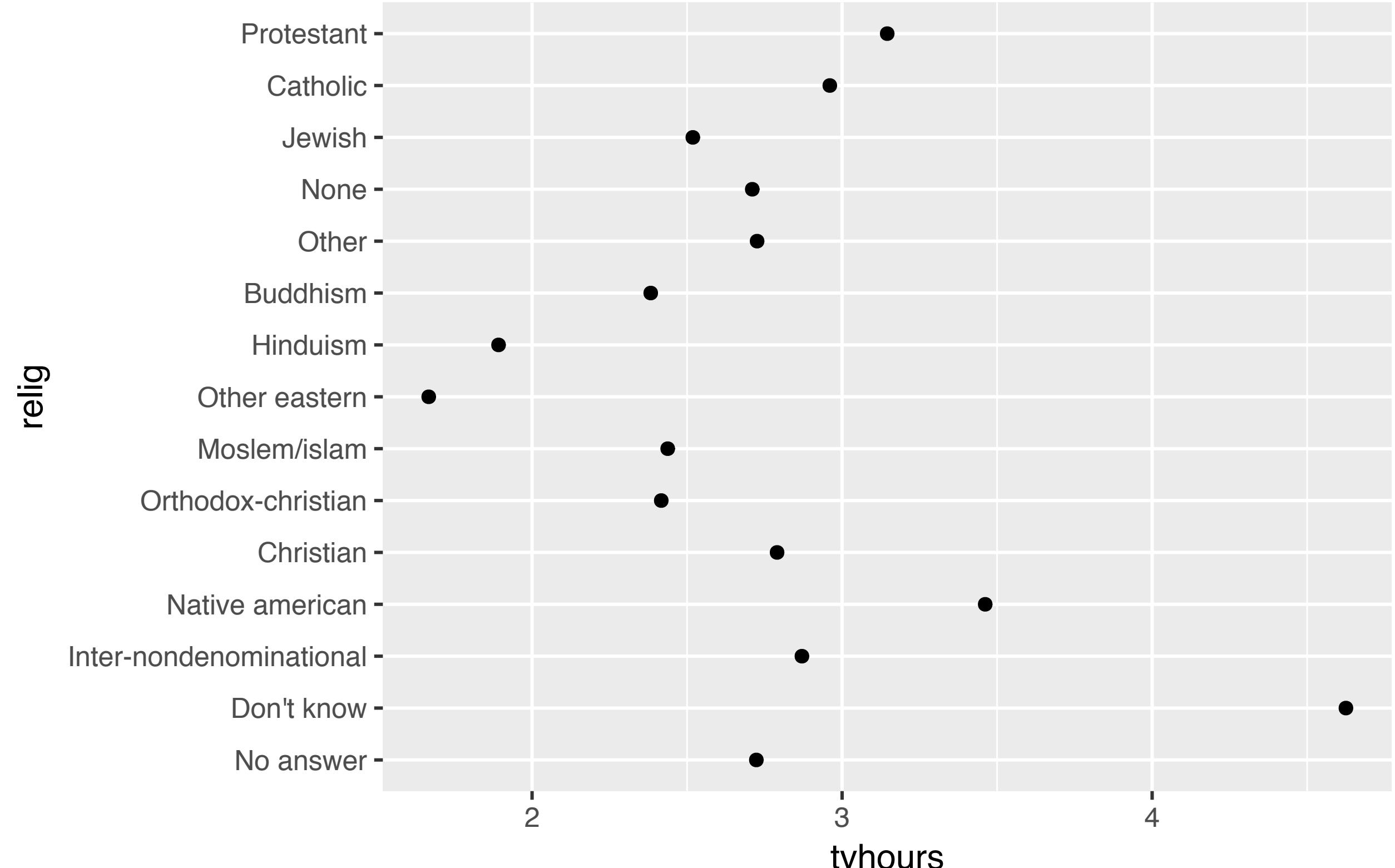


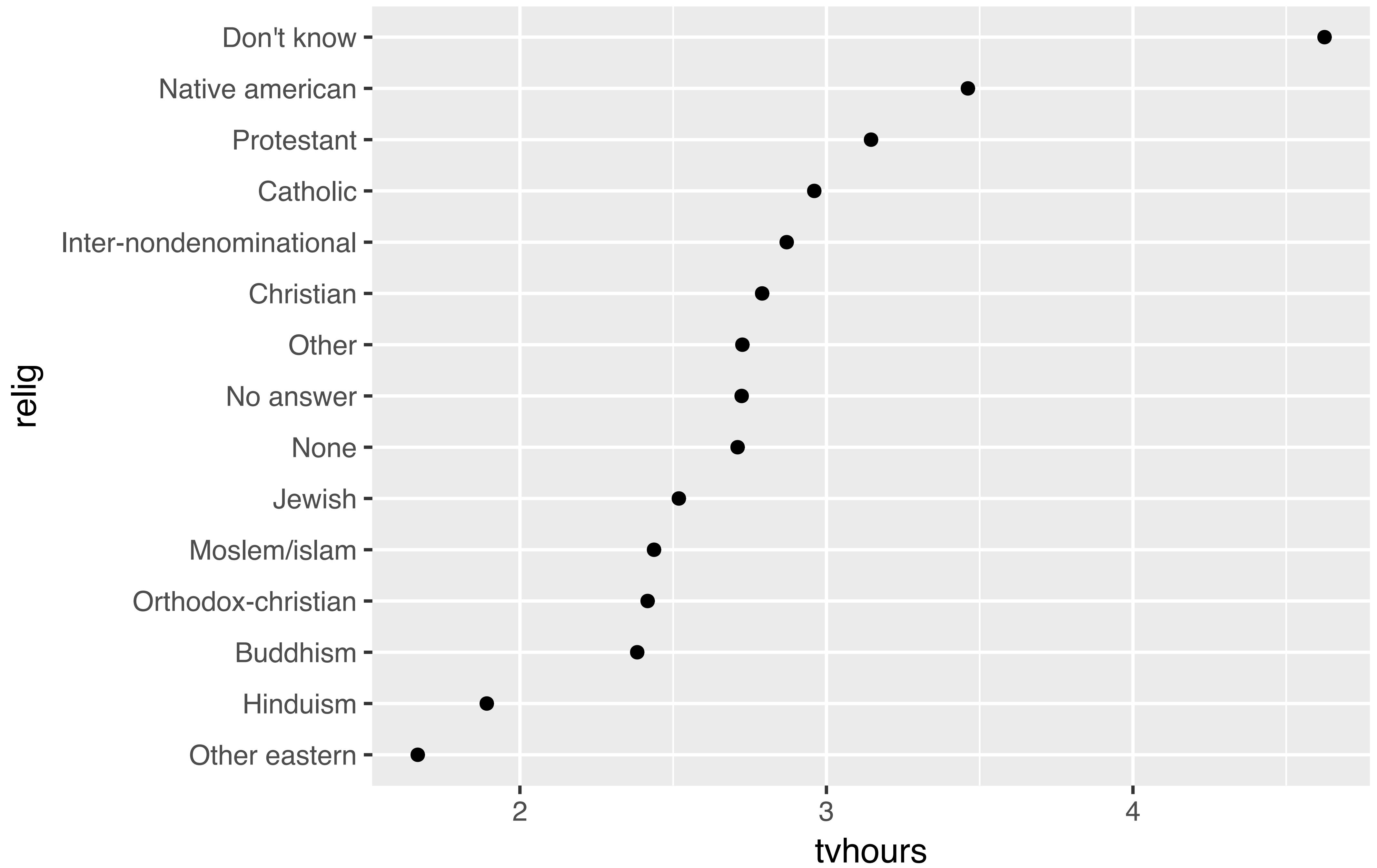
Which religions watch the least TV?

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



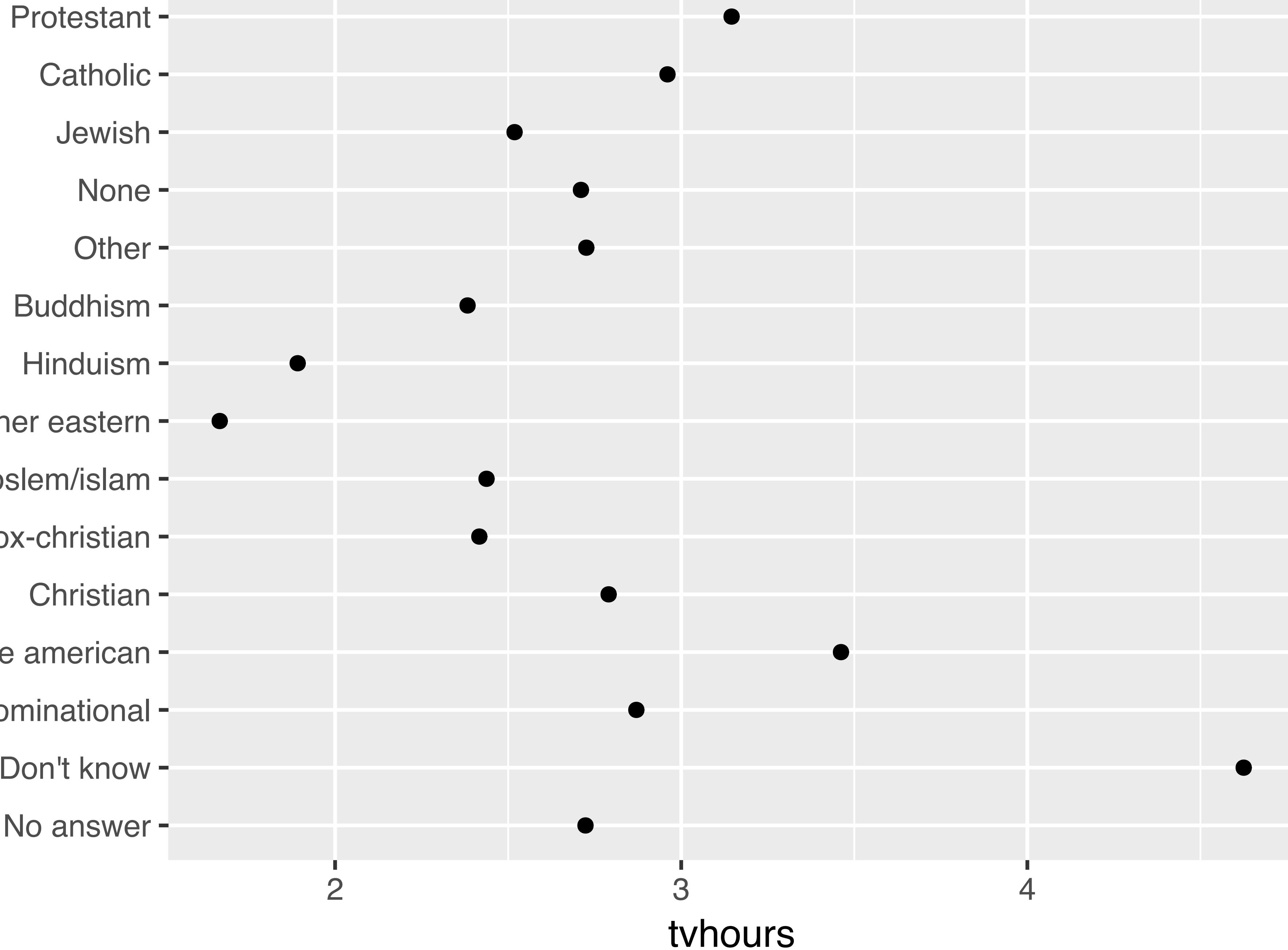
Which do you prefer?





Why is the Y axis in this order?

relig



levels()

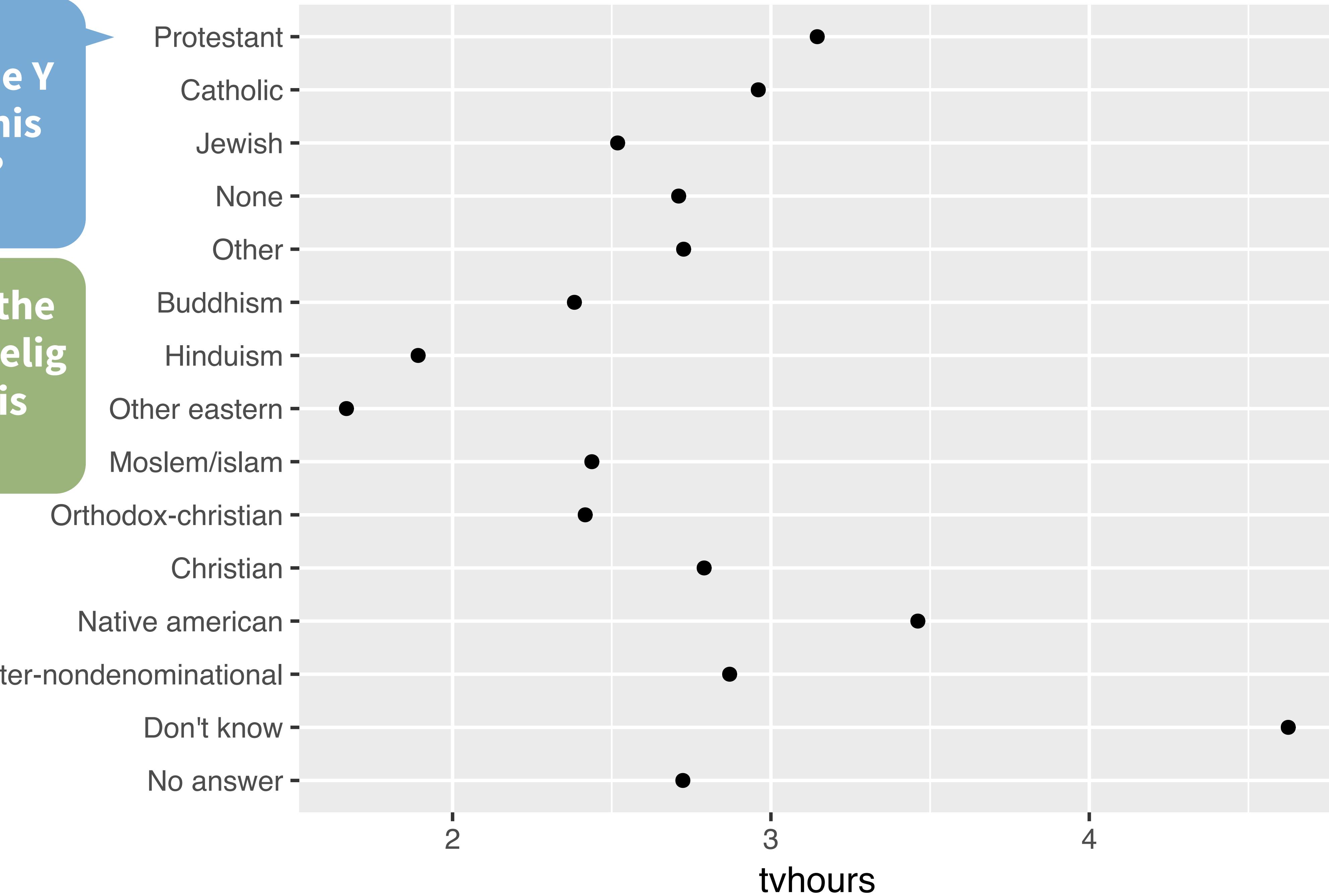
Use **levels()** to access a factor's levels

```
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"
```



Why is the Y axis in this order?

Because the levels of relig have this order



Most useful skills

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



Reordering levels

R

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)

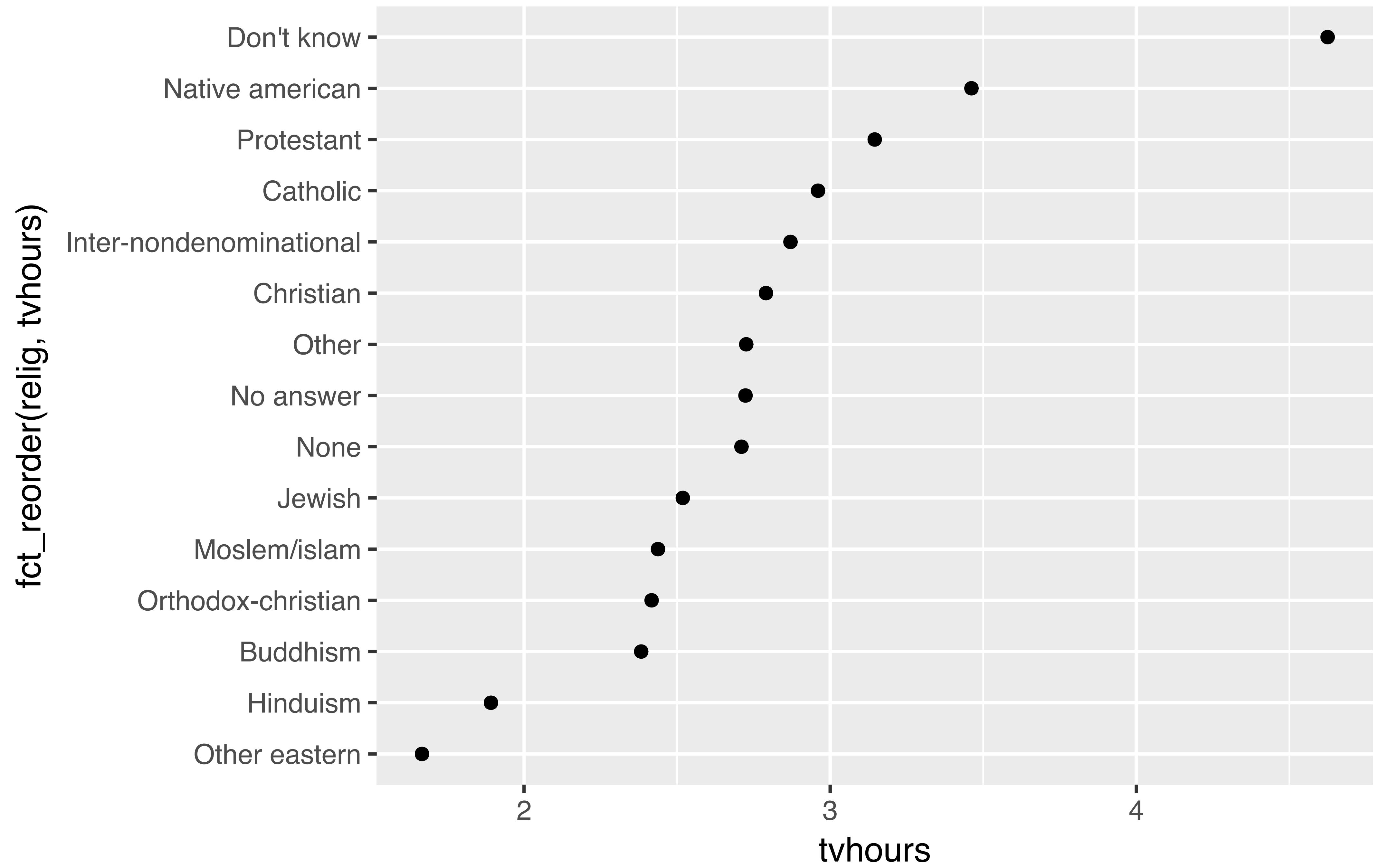
function to
reorder by
(in conjunction
with x)

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()
```





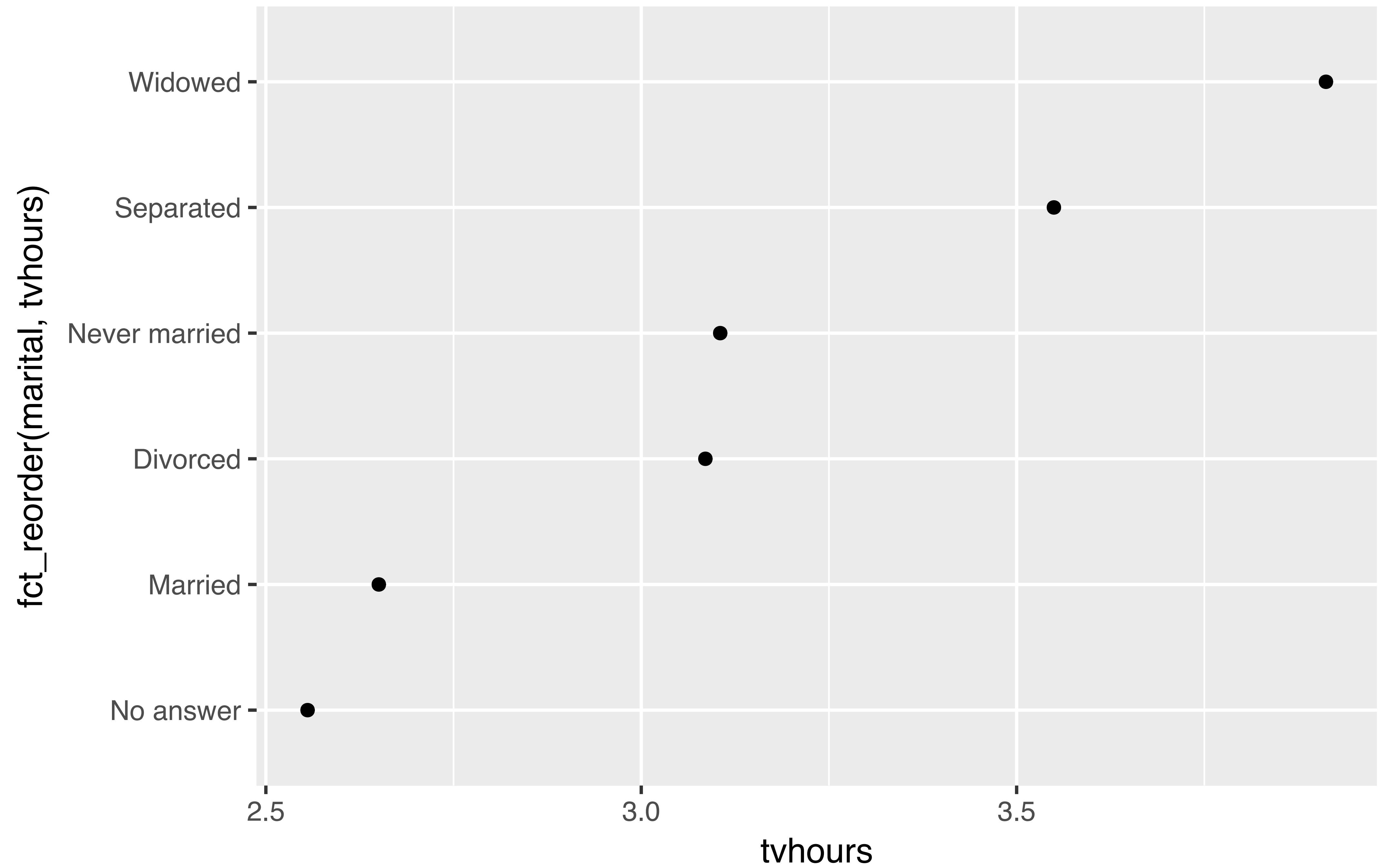
Your Turn 3

Repeat the previous exercise, some of whose code is in your notebook, to make a sensible graph of average TV consumption by marital status.



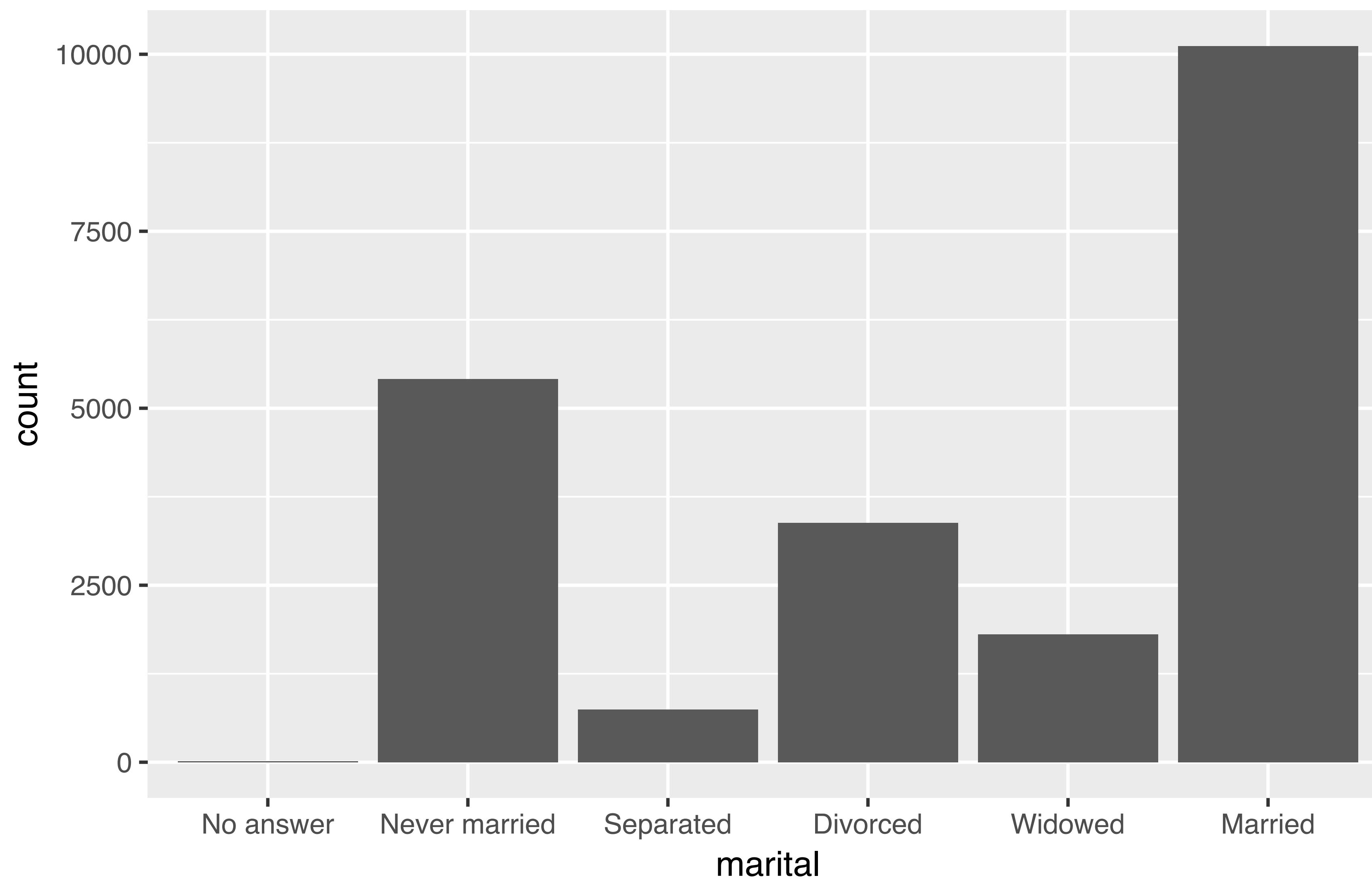
```
gss_cat %>%  
  drop_na(tvhours) %>%  
  group_by(marital) %>%  
  summarise(tvhours = mean(tvhours)) %>%  
  ggplot(aes(tvhours, fct_reorder(marital, tvhours))) +  
  geom_point()
```





Similar reordering functions

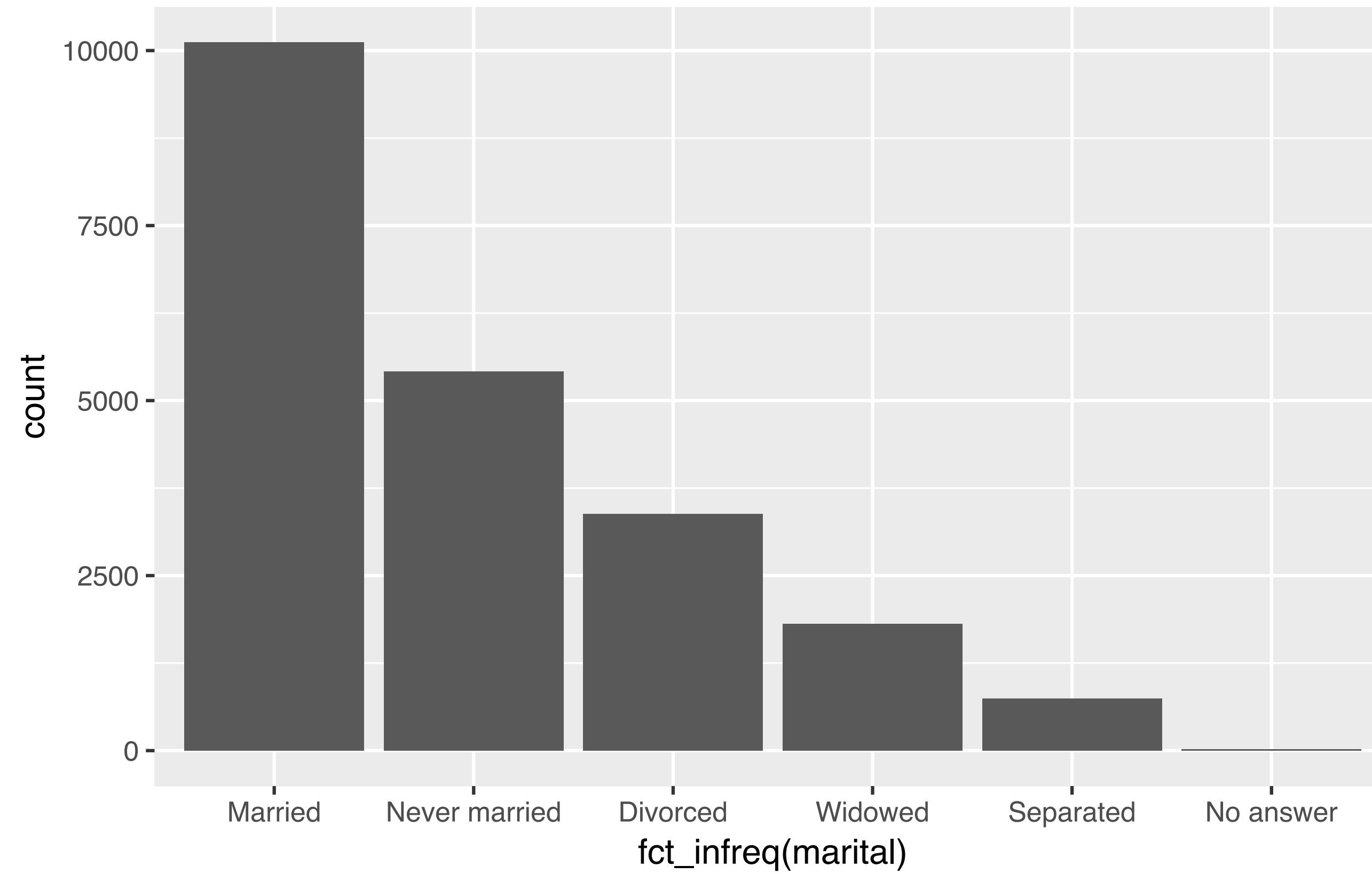




```
gss_cat %>%  
  ggplot(aes(marital)) + geom_bar()
```



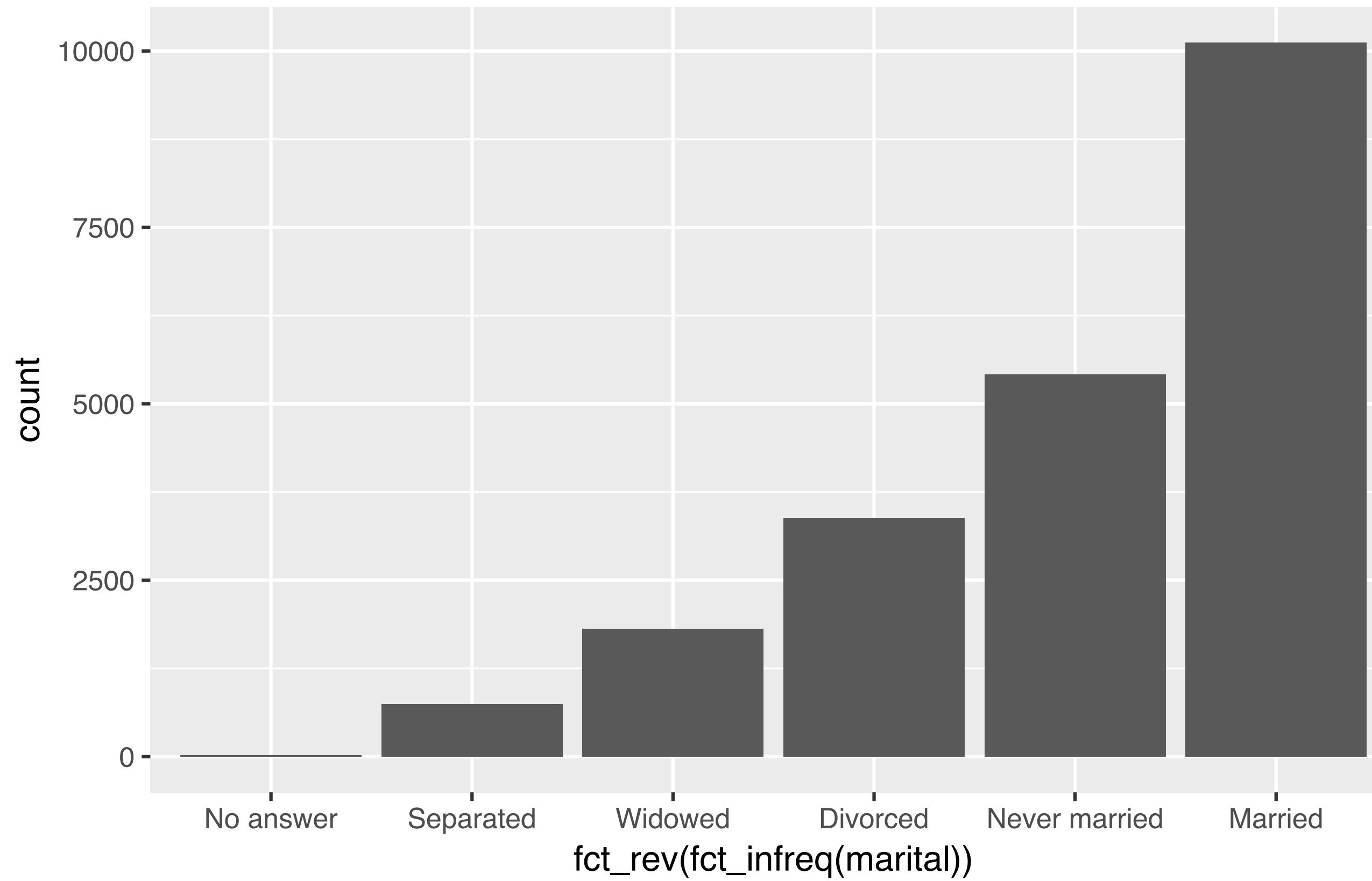
fct_infreq



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



fct_rev



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

Changing level values



Your Turn 4

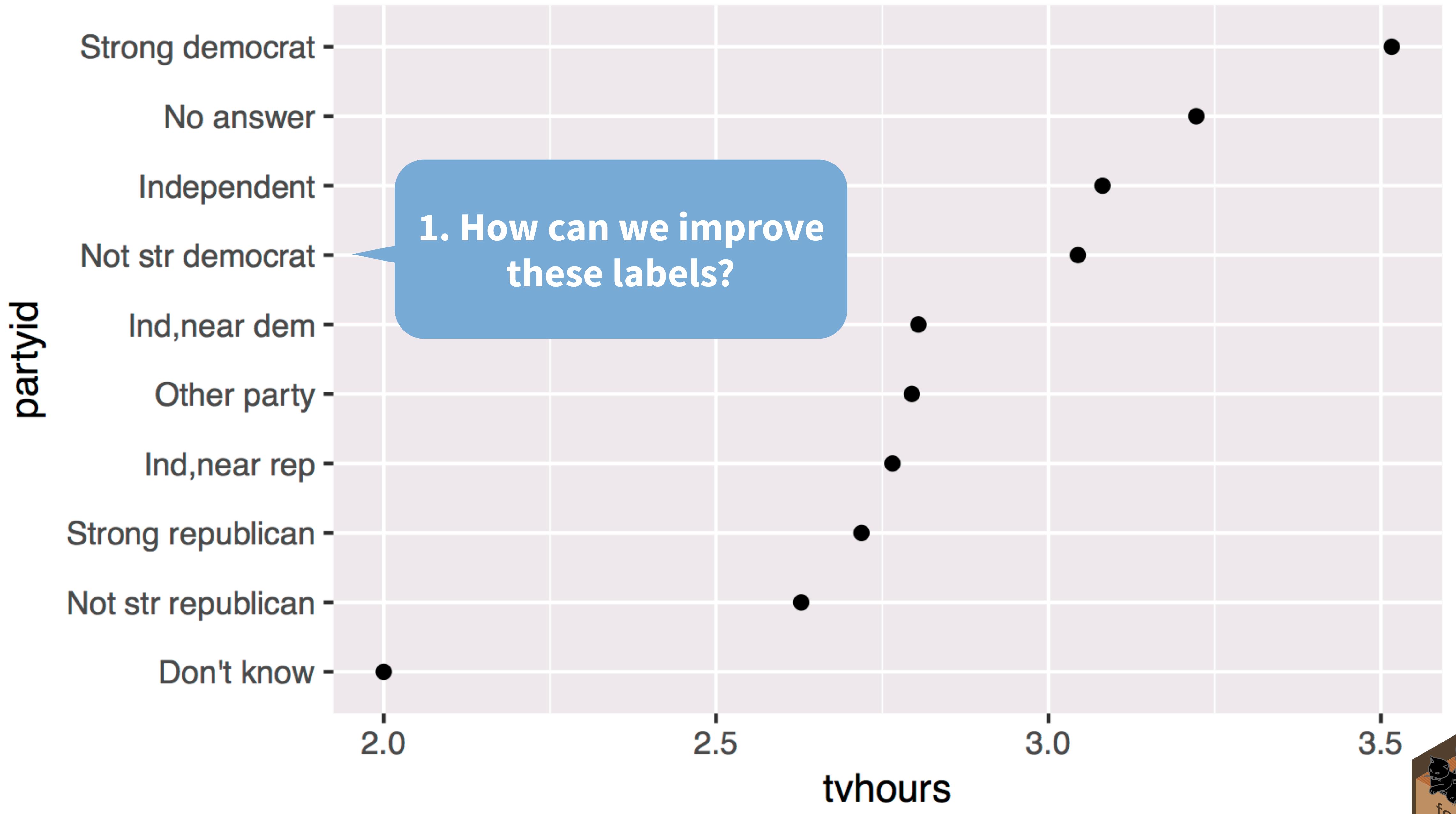
Do you think liberals or conservatives watch more TV?

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, fct_reorder(partyid, tvhours))) +  
  geom_point() +  
  labs(y = "partyid")
```





fct_recode()

Changes values of levels

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

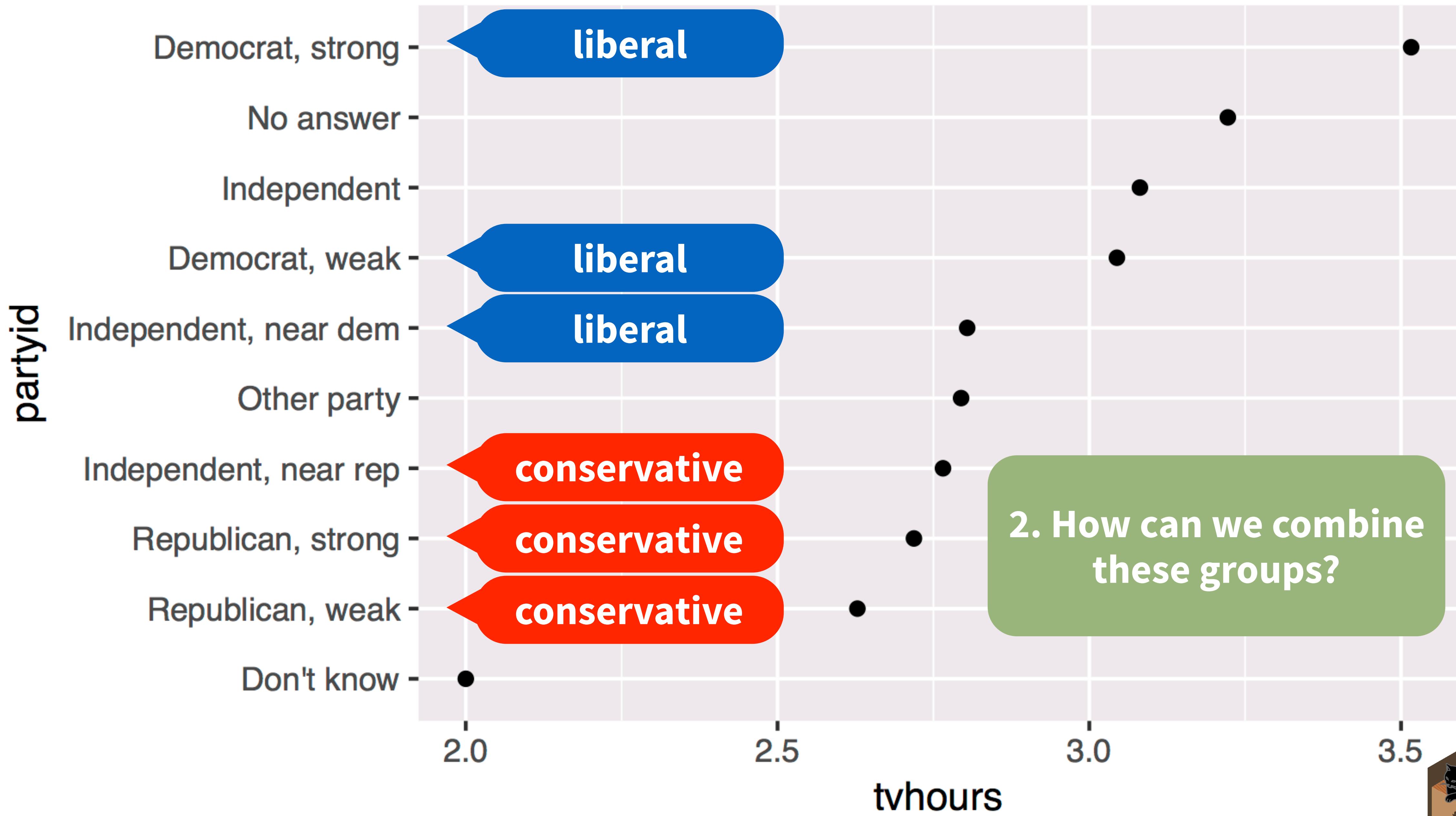
factor with
levels

**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, fct_reorder(partyid, tvhours))) +  
  geom_point() + labs(y = "partyid")
```





Collapsing levels

A large, semi-transparent watermark of the R logo is positioned in the bottom right corner of the slide. The logo consists of a circular arrow with the letters "R" inside.

fct_collapse()

Changes multiple levels into single levels

```
fct_collapse(f, ...)
```

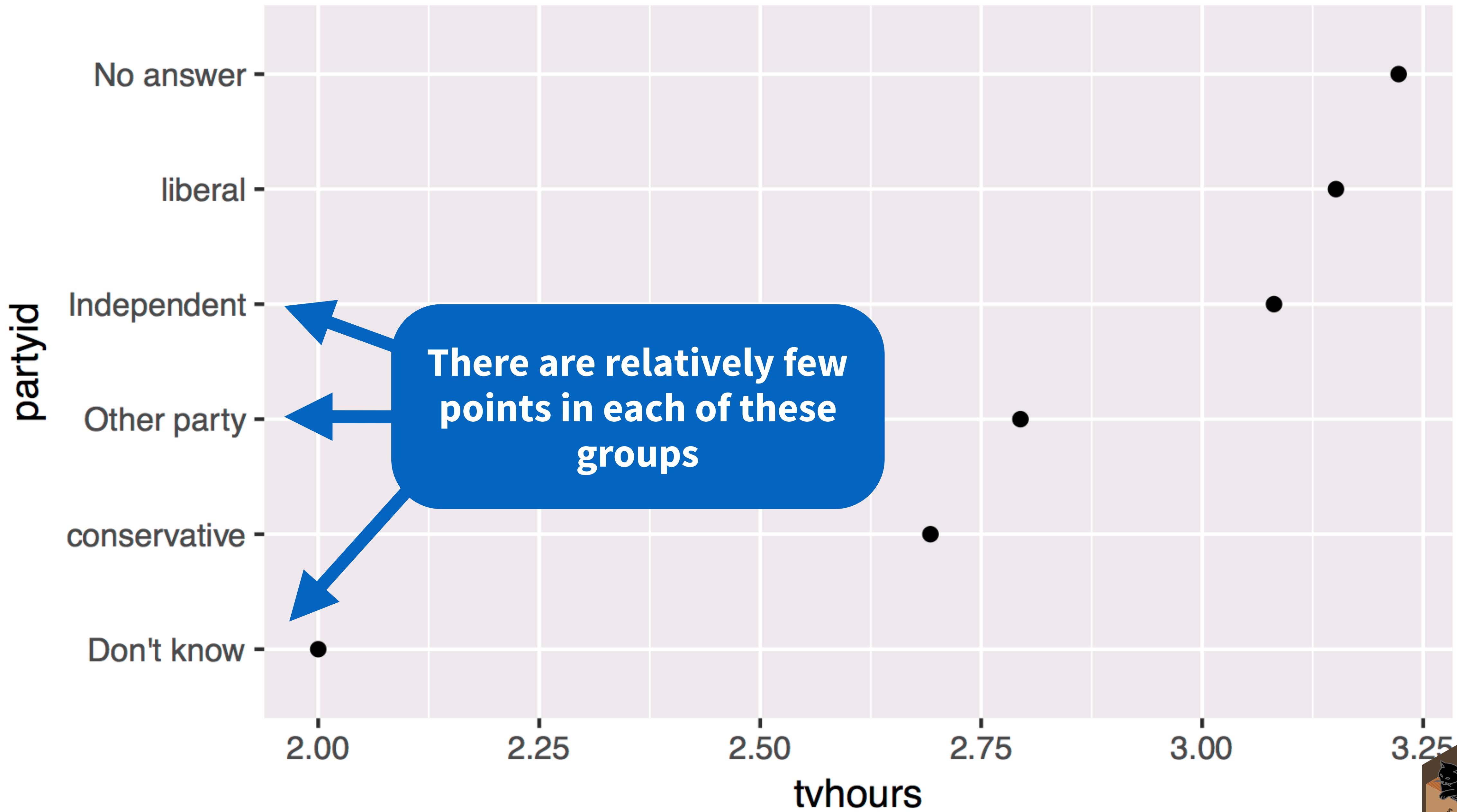
factor with
levels

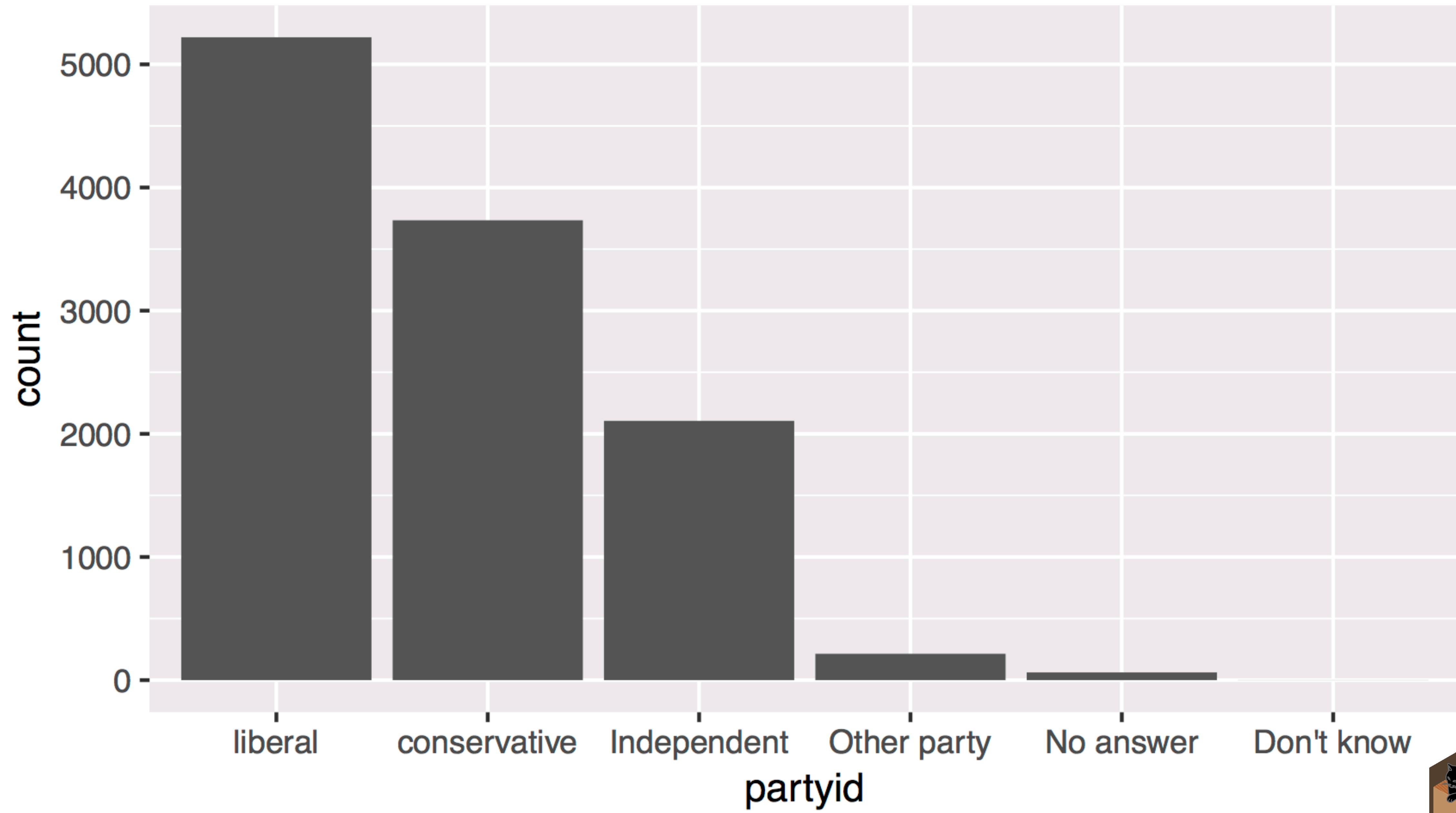
named arguments set to a
character vector (levels in the
vector will be collapsed to the name
of the argument)



```
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, fct_reorder(partyid, tvhours))) +  
  geom_point() + labs(y = "partyid")
```







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", ...)
```

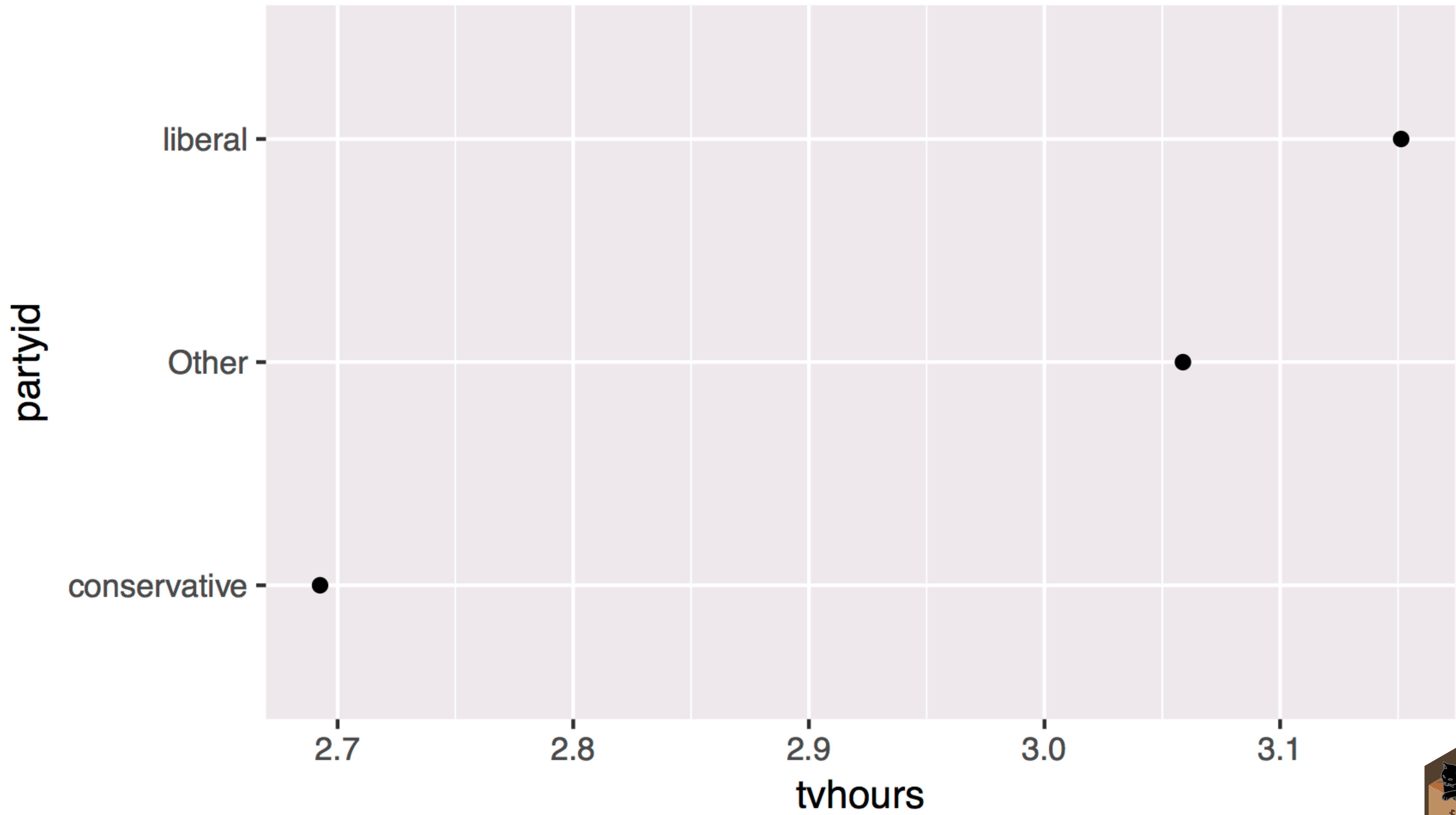
factor with
levels

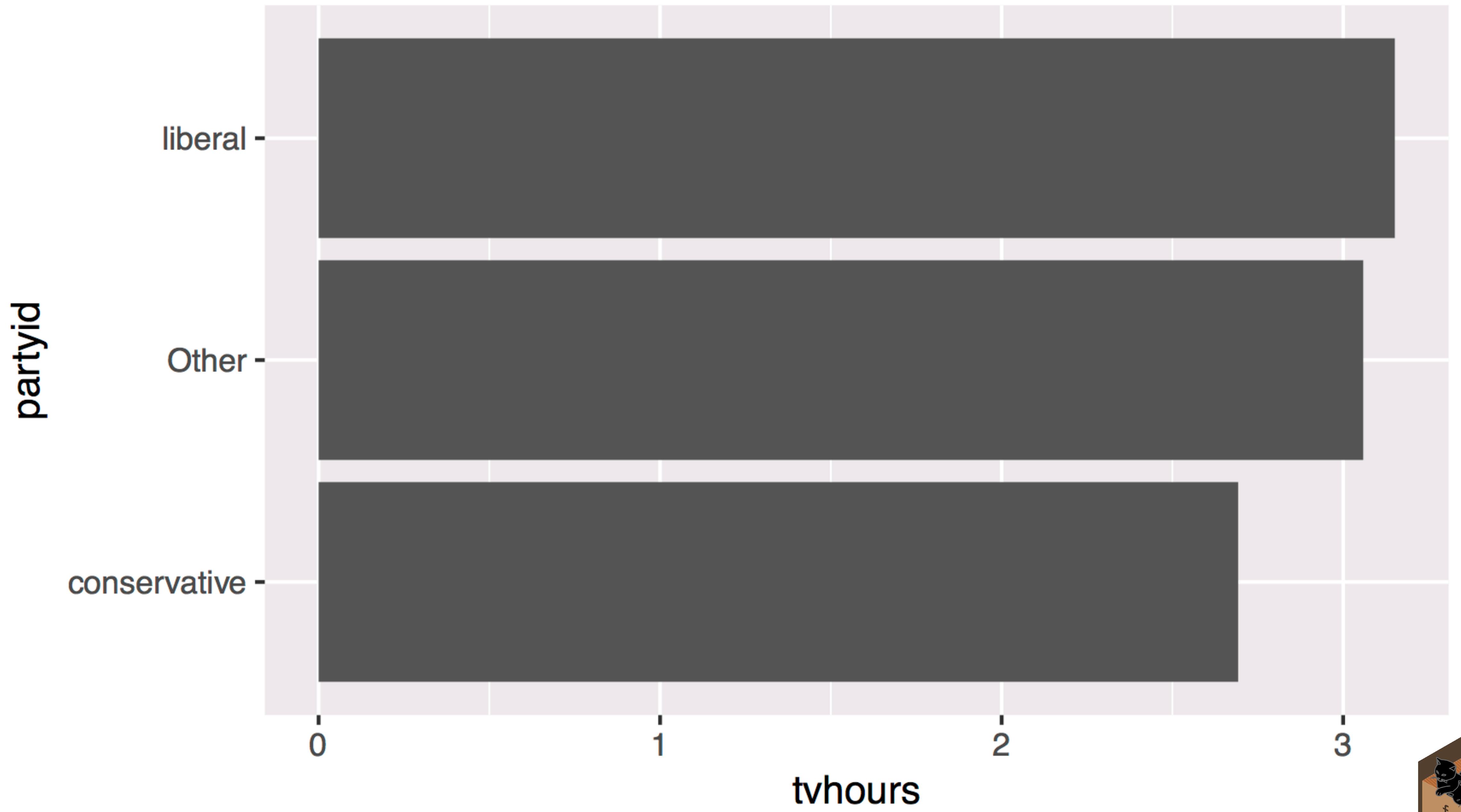
name of new level



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



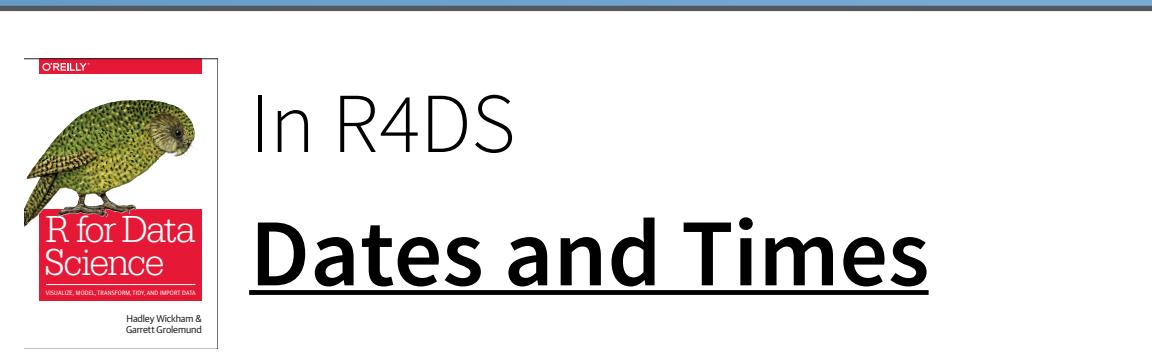




```
gss_cat %>%  
  drop_na(tvhours) %>%  
  mutate(partyid = partyid %>%  
    fct_collapse(  
      conservative = c("Strong republican",  
                        "Not str republican", "Ind,near rep"),  
      liberal = c("Strong democrat", "Not str democrat",  
                  "Ind,near dem")))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



Date times



In R4DS

Dates and 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?

Most useful skills

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



Warm Up

Decide in your group:

- What is the best time of day to fly?
- What is the best day of the week to fly?



```
flights %>% select(c(1, 2, 3, 17, 18, 5, 19))
```

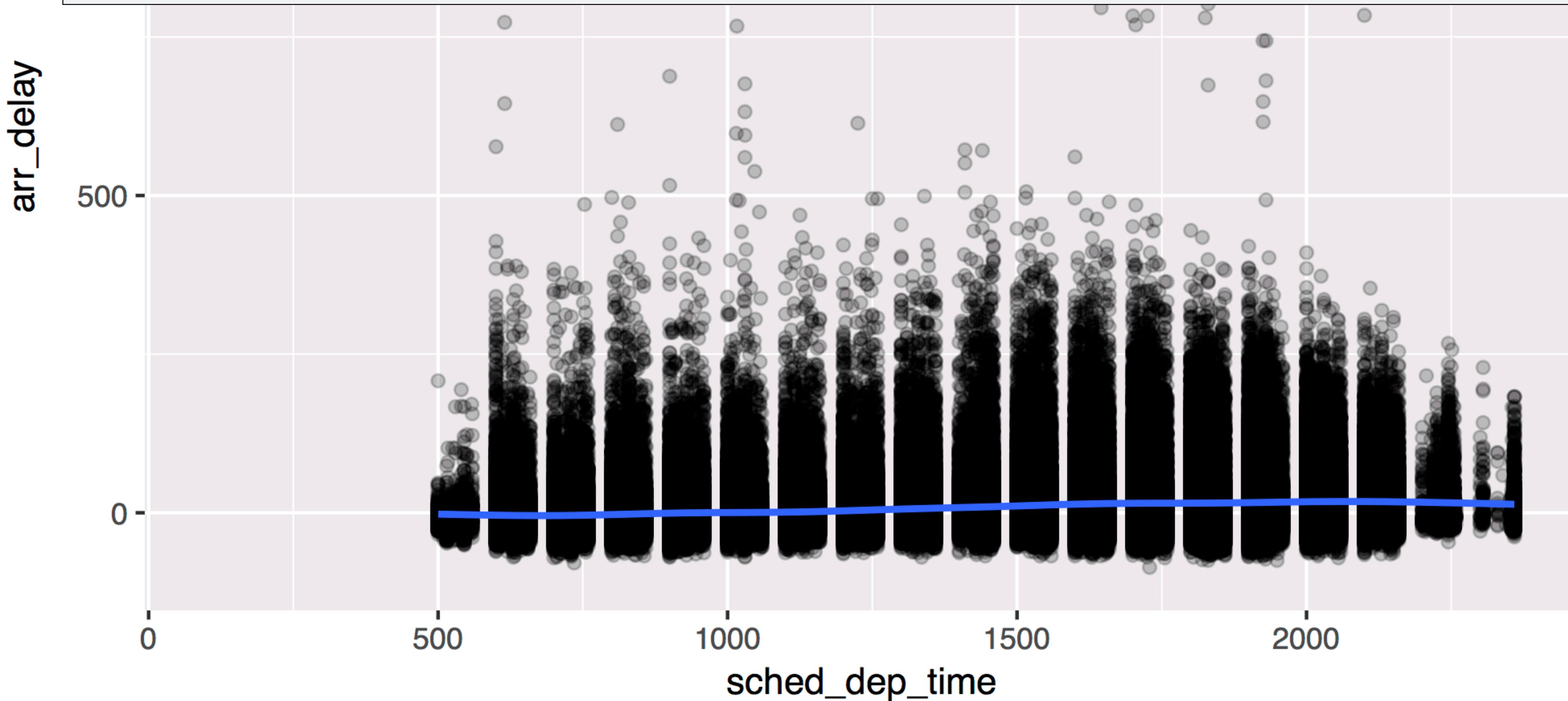
year	month	day	hour	minute	sched_dep_time	time_hour
<int>	<int>	<int>	<dbl>	<dbl>	<int>	<S3: POSIXct>
2013	1	1	5	15	515	2013-01-01 05:00:00
2013	1	1	5	29	529	2013-01-01 05:00:00
2013	1	1	5	40	540	2013-01-01 05:00:00
2013	1	1	5	45	545	2013-01-01 05:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	5	58	558	2013-01-01 05:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00

1-10 of 336,776 rows

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



```
flights %>%  
  ggplot(mapping = aes(x = sched_dep_time, y = arr_delay)) +  
  geom_point(alpha = 0.2) + geom_smooth()
```



```
flights %>% select(c(1, 2, 3, 17, 18, 5, 19))
```

year	month	day	hour	minute	sched_dep_time	time_hour
<int>	<int>	<int>	<dbl>	<dbl>	<int>	<S3: POSIXct>
2013	1	1	5	15	515	2013-01-01 05:00:00
2013	1	1	5	29	529	2013-01-01 05:00:00
2013	1	1	5	40	540	2013-01-01 05:00:00
2013	1	1	5	45	545	2013-01-01 05:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	5	58	558	2013-01-01 05:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00
2013	1	1	6	0	600	2013-01-01 06:00:00

1-10 of 336,776 rows

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



Creating dates and times

R

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)
```

* on a typical day



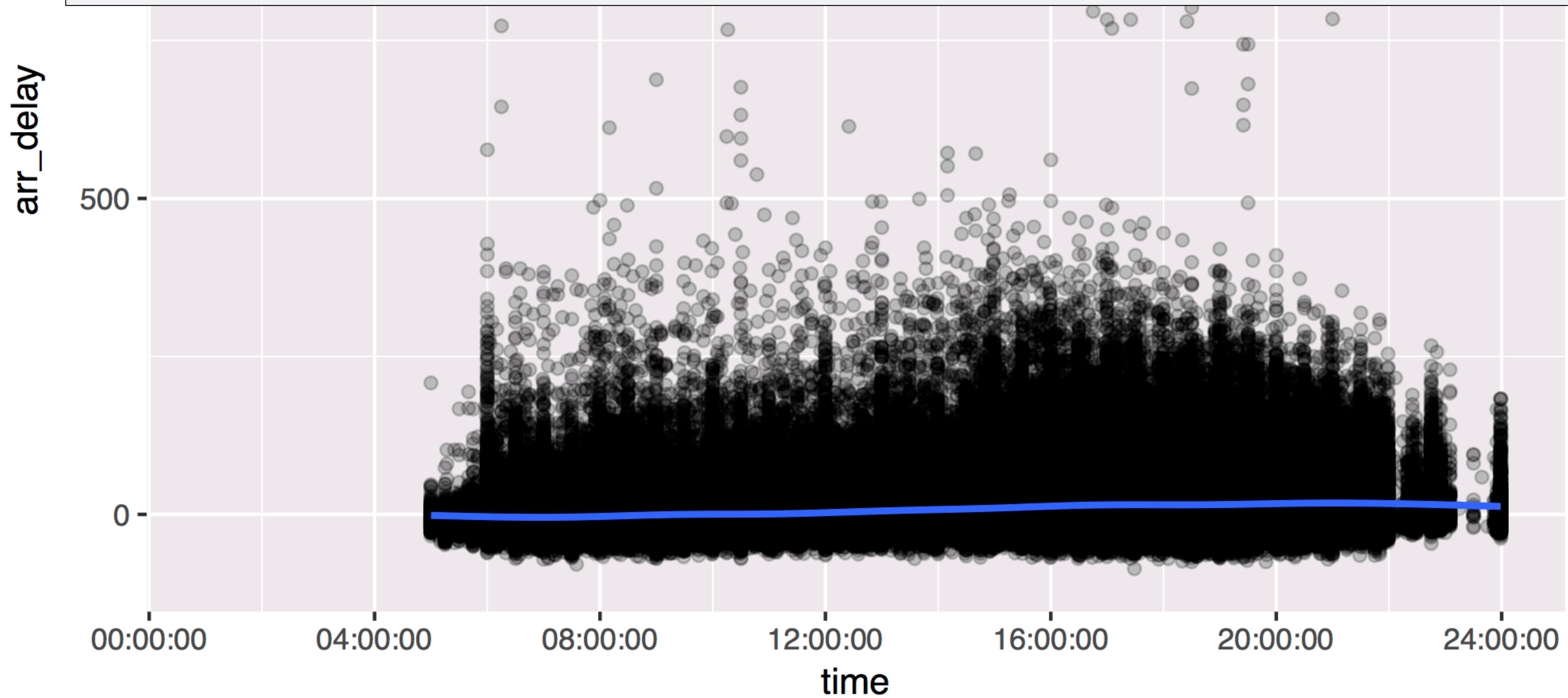
Your Turn 5

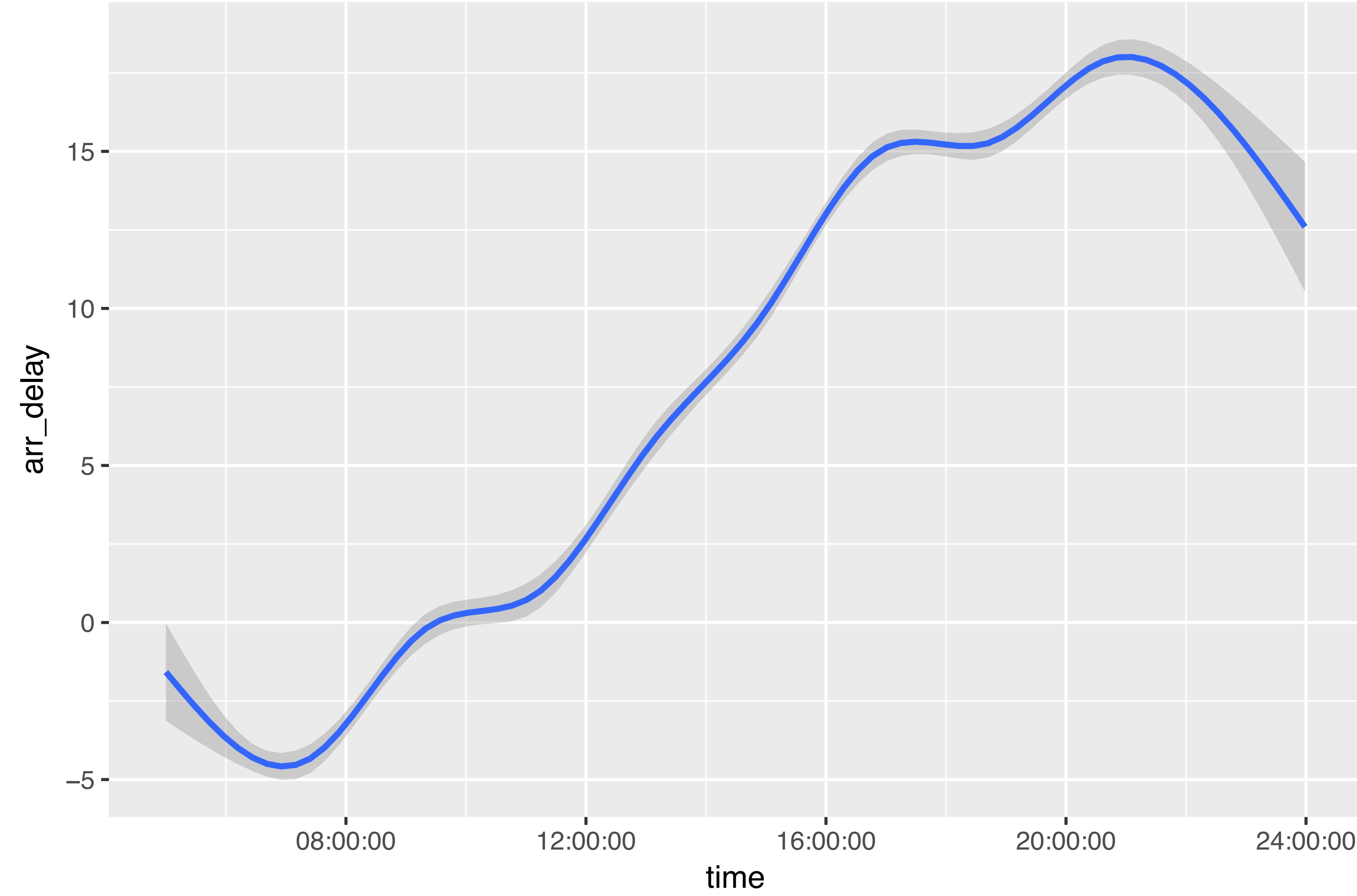
What is the best time of day to fly?

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



```
flights %>%  
  mutate(time = hms(hour = hour, minute = minute)) %>%  
  ggplot(aes(time, arr_delay)) +  
  geom_point(alpha = 0.2) + geom_smooth()
```





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()	POSIXct
dmy_hms(), dmy_hm(), dmy_h()	
mdy_hms(), mdy_hm(), mdy_h()	
ymd(), ydm(), mdy()	
myd(), dmy(), dym(), yq()	Date (POSIXct if tz specified)
hms(), hm(), ms()	Period



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"))

## 4

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] Wednesday

## 7 Levels: Sunday < Monday < Tuesday < ... < Saturday
```



Your Turn 6

Fill in the blanks to:

Extract the day of the week of each flight (as a full name) from `time_hour`.

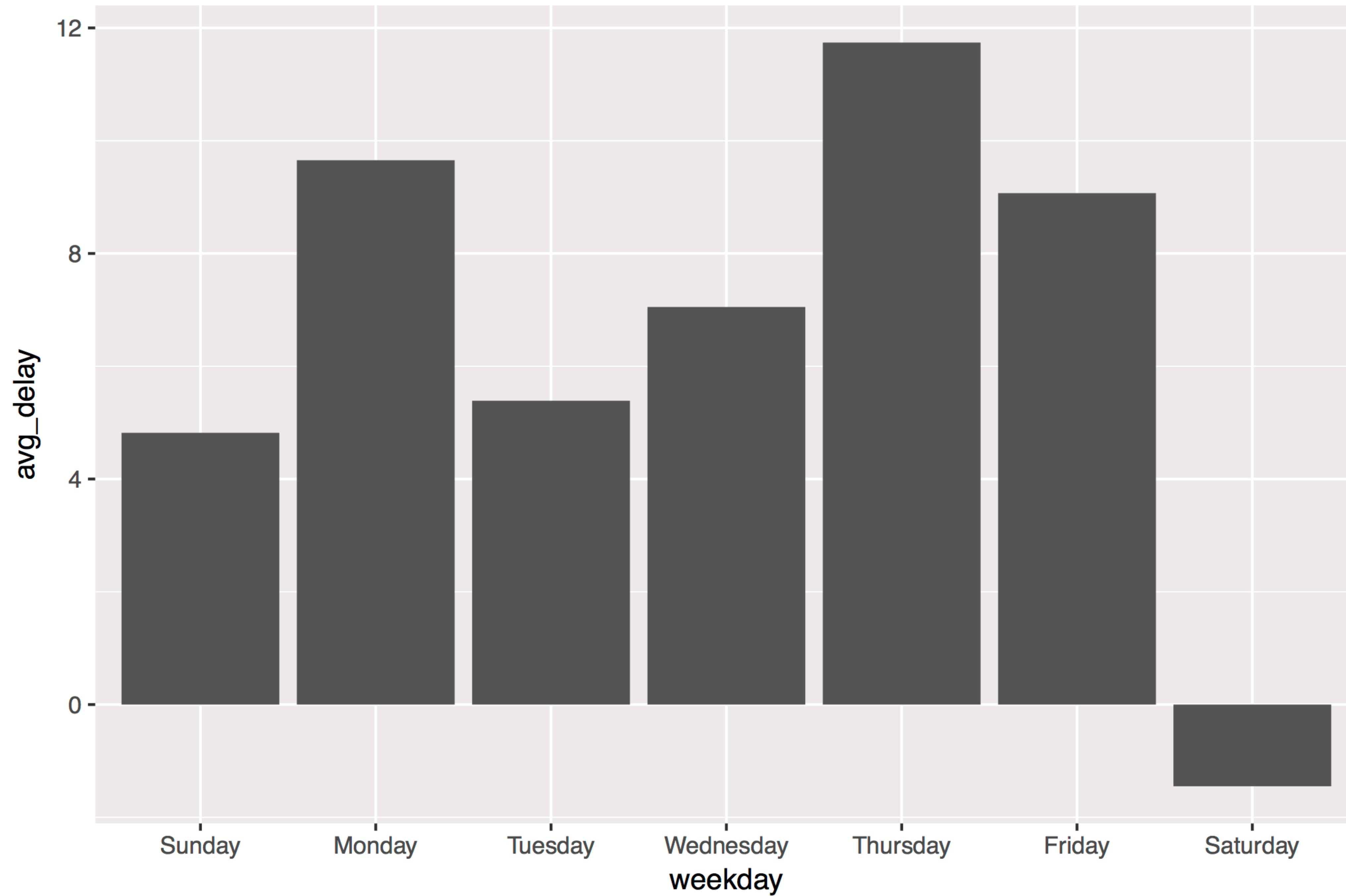
Calculate the average `arr_delay` by day of the week.

Plot the results as a column chart (bar chart) with `geom_col()`.



```
flights %>%  
  mutate(weekday = wday(time_hour, label = TRUE, abbr = FALSE)) %>%  
  group_by(weekday) %>%  
  drop_na(arr_delay) %>%  
  summarise(avg_delay = mean(arr_delay)) %>%  
  ggplot() +  
    geom_col(mapping = aes(x = weekday, y = avg_delay))
```





Parsing functions

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



Parsing functions

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

Same name as
hms() in hms



`hms::hms()`

**package
name**

**function
name**

* on a typical day



`hms::hms()`

`lubridate::hms()`

* on a typical day



`hms()`

```
hms::hms(seconds = 3, hours = 5)
```

Use the
`hms()` function in
the `hms` package



* on a typical day

Data types with

