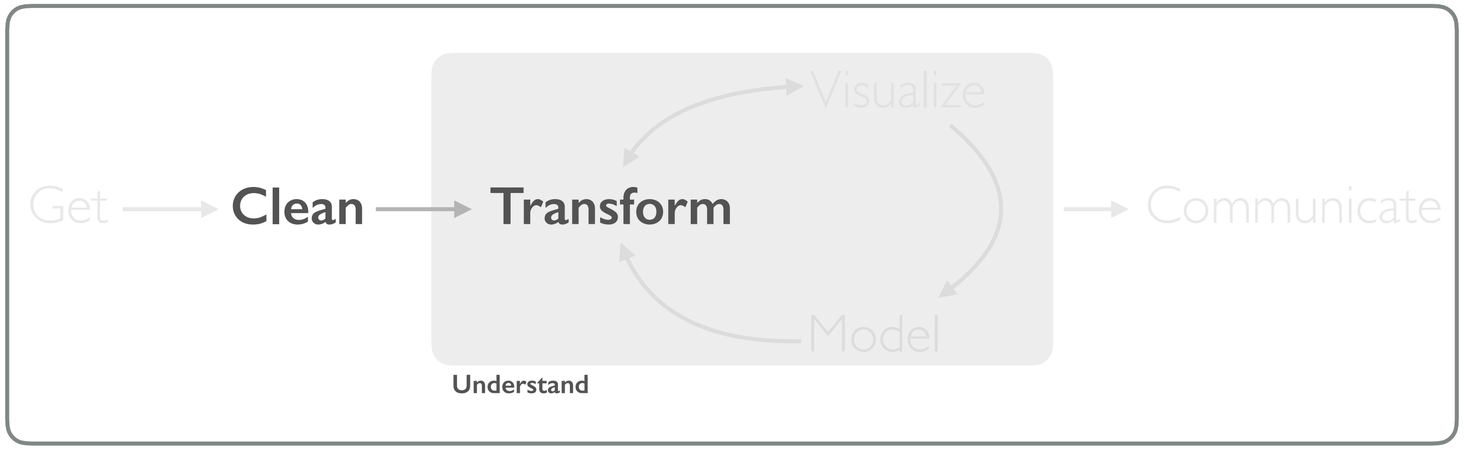
FACTORS



†A modified version of Hadley Wickham's analytic process

factors = categorical variables

Nominal	male, female Hispanic, Asian, African, Caucasian
Ordinal	slow, medium, fast freshman, sophomore, junior, senior
Interval	$0-25, $26-50, $51-75, $76-100 0-10%, 11-20%, 21-30%

Factors have finite categorical levels

FACTORS

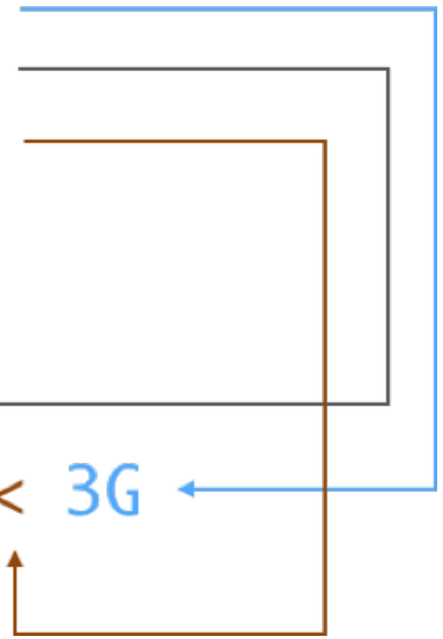
- Factors are a useful data structure; particularly for modeling and visualizations because they control the order of levels
- Working with factors in base R can be a little frustrating because of a handful of missing tools
- The goal of `forcats` is to fill in those missing pieces so you can access the power of factors with minimum pain

FACTOR PRINCIPALS

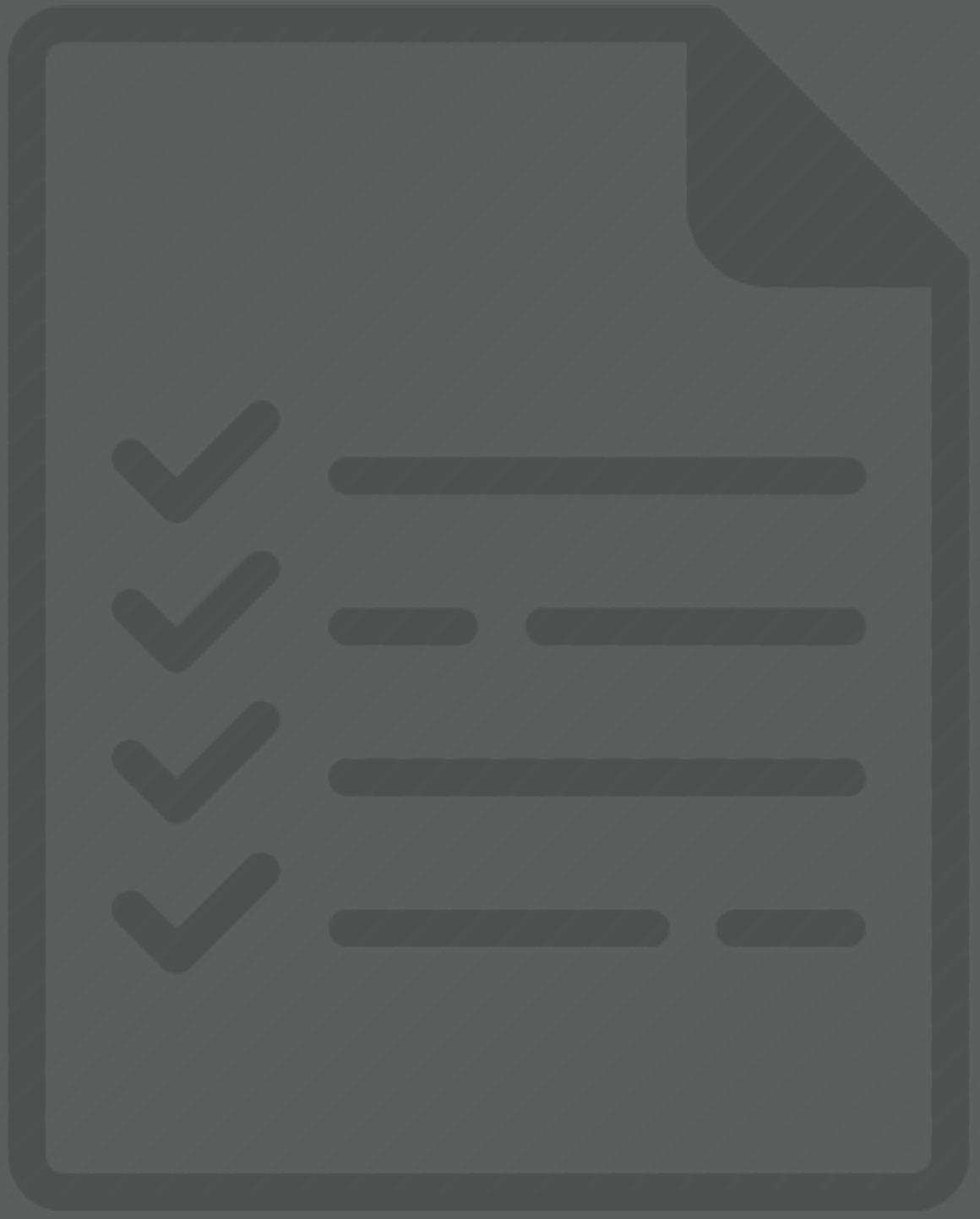
3 Main things to think about:

- level
- label
- order

```
factor(grade)
## [1] pre-k  K  1G  2G  3G
## Levels: pre-k < K < 1G < 2G < 3G
```



PREREQUISITES



PREREQUISITES

- Re-start your R session
  - **Windows:** Ctrl+Shift+F10
  - **Mac:** Command+Shift+F10
- Make sure your working directory is set to the course folder

PACKAGE PREREQUISITE

```
library(forcats)
library(tidyverse)
```

EXAMPLE DATA PREREQUISITE

```
forcats::gss_cat
# A tibble: 21,483 × 9
    year       marital   age   race        rincome            partyid
   <int>        <fctr> <int> <fctr>         <fctr>             <fctr>
1   2000 Never married    26  White  $8000 to 9999       Ind,near rep
2   2000      Divorced    48  White  $8000 to 9999 Not str republican
3   2000       Widowed    67  White Not applicable        Independent
4   2000 Never married    39  White Not applicable       Ind,near rep
5   2000      Divorced    25  White Not applicable   Not str democrat
6   2000       Married    25  White $20000 - 24999    Strong democrat
7   2000 Never married    36  White $25000 or more Not str republican
8   2000      Divorced    44  White  $7000 to 7999       Ind,near dem
9   2000       Married    44  White $25000 or more   Not str democrat
10  2000       Married    47  White $25000 or more  Strong republican
# ... with 21,473 more rows, and 3 more variables: relig <fctr>, denom <fctr>,
#   tvhours <int>
```

PRACTICE DATA PREREQUISITE

```
nycflights13::weather
nycflights13::flights
nycflights13::airlines
```

CREATING FACTORS

CREATING FACTORS

```
grade <- c("pre-K", "1G", "K", "K", "2G", "1G")
```

- Create this character string

CREATING FACTORS

```
grade <- c("pre-K", "1G", "K", "K", "2G", "1G")

factor(grade)
[1] pre-K 1G    K     K    2G    1G
Levels: 1G 2G K pre-K
```

- We can turn this into a factor with `factor()`
- Automatically establishes levels in alpha-numeric order

CREATING FACTORS

```
grade <- c("pre-K", "1G", "K", "K", "2G", "1G")

factor(grade)
[1] pre-K 1G    K     K     2G    1G
Levels: 1G 2G K pre-K

factor(grade, levels = unique(grade))
[1] pre-K 1G    K     K     2G    1G
Levels: pre-K 1G K 2G
```

- We can establish the levels in the order that the data first appears with `levels = unique(x)`
- However, in this case the levels are still out of order

CREATING FACTORS

```
grade <- c("pre-K", "1G", "K", "K", "2G", "1G")

factor(grade)
[1] pre-K 1G    K     K    2G    1G
Levels: 1G 2G K pre-K

factor(grade, levels = unique(grade))
[1] pre-K 1G    K     K    2G    1G
Levels: pre-K 1G K 2G

factor(grade,
       levels = c("pre-K", "K", "1G", "2G"),
       ordered = TRUE)
[1] pre-K 1G    K     K    2G    1G
Levels: pre-K < K < 1G < 2G
```

- Here we define the exact order of the levels and also use `ordered = TRUE` to make this an ordinal factor

YOUR TURN!

1. *Using the* `nycflights13::weather` *data, recode* `month` *from a numeric to a factor variable. Be sure the levels are ordered properly. (Reminder: indexing* `weather$month` *allows you to work with this vector directly)*

2. *Can you figure out how to change the month values from numeric (i.e.* `1, 2, 3,...`*) to names (i.e.* `"Jan"`, `"Feb"`, `"Mar"`, ...*)? Hint: check out the vector* `months.abb`

3. *How would you do this in the middle of a piped operation (hint:* `dplyr::mutate`*)*

SOLUTION

```
# problem 1
factor(weather$month) %>%
  str()
 Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...

# problem 2
factor(weather$month, labels = month.abb) %>%
  str()
Factor w/ 12 levels "Jan","Feb","Mar",..: 1 1 1 1 1 1 1 1 1 1 ...

# problem 3
weather %>%
  mutate(month = factor(month, labels = month.abb)) %>%
  str()
Classes ‘tbl_df’, ‘tbl’ and 'data.frame': 26130 obs. of  15 variables:
 $ origin    : chr  "EWR" "EWR" "EWR" "EWR" ...
 $ year      : num  2013 2013 2013 2013 2013 ...
 $ month     : Factor w/ 12 levels "Jan","Feb","Mar",..: 1 1 1 1 1 1 1 1 1 1 ...
```

CHANGING LEVEL VALUES

CREATING FACTORS

```
gss_cat %>%
  count(partyid)
# A tibble: 10 × 2
              partyid     n
               <fctr> <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
```

- What if you have unnecessary levels?

CREATING FACTORS

```
gss_cat %>%
  mutate(partyid = fct_recode(
                     partyid,
                     Other = "No answer",
                     Other = "Don't know",
                     Other = "Other party")
                   ) %>%
  count(partyid)
# A tibble: 8 × 2
             partyid     n
              <fctr> <int>
1              Other   548
2  Strong republican  2314
3 Not str republican  3032
4       Ind,near rep  1791
5        Independent  4119
6       Ind,near dem  2499
```

- What if you have unnecessary levels?
- We can recode these levels using `fct_recode()`

CREATING FACTORS

```
gss_cat %>%
  count(partyid)
# A tibble: 10 × 2
                partyid     n
                 <fctr> <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
```

- But if we want to collapse a bunch of levels then there is an easier way
- `fct_collapse()`

CREATING FACTORS

```
gss_cat %>%
  mutate(partyid = fct_collapse(
                     partyid,
                     Other = c("No answer", "Don't know", "Other party"),
                     Republican = c("Strong republican", "Not str republican"),
                     Independent = c("Ind,near rep", "Independent", "Ind,near dem"),
                     Democrat = c("Not str democrat", "Strong democrat"))
                   ) %>%
  count(partyid)
# A tibble: 4 × 2
      partyid     n
       <fctr> <int>
1       Other   548
2  Republican  5346
3 Independent  8409
4    Democrat  7180
```

YOUR TURN!

1. *Using the* `nycflights13::weather` *data, can you create a new factor variable (hint* `mutate()`*) that is titled "season"? This variable should align with month data in the following way:*

    1. *Winter: Jan - Mar*
    2. *Spring: Apr - Jun*
    3. *Summer: Jul - Sep*
    4. *Fall: Oct - Dec*

SOLUTION

```
weather %>%
  mutate(season = fct_collapse(
                    factor(month),
                    Winter = c("1", "2", "3"),
                    Spring = c("4", "5", "6"),
                    Summer = c("7", "8", "9"),
                    Fall = c("10", "11", "12")
                  )
  )
```

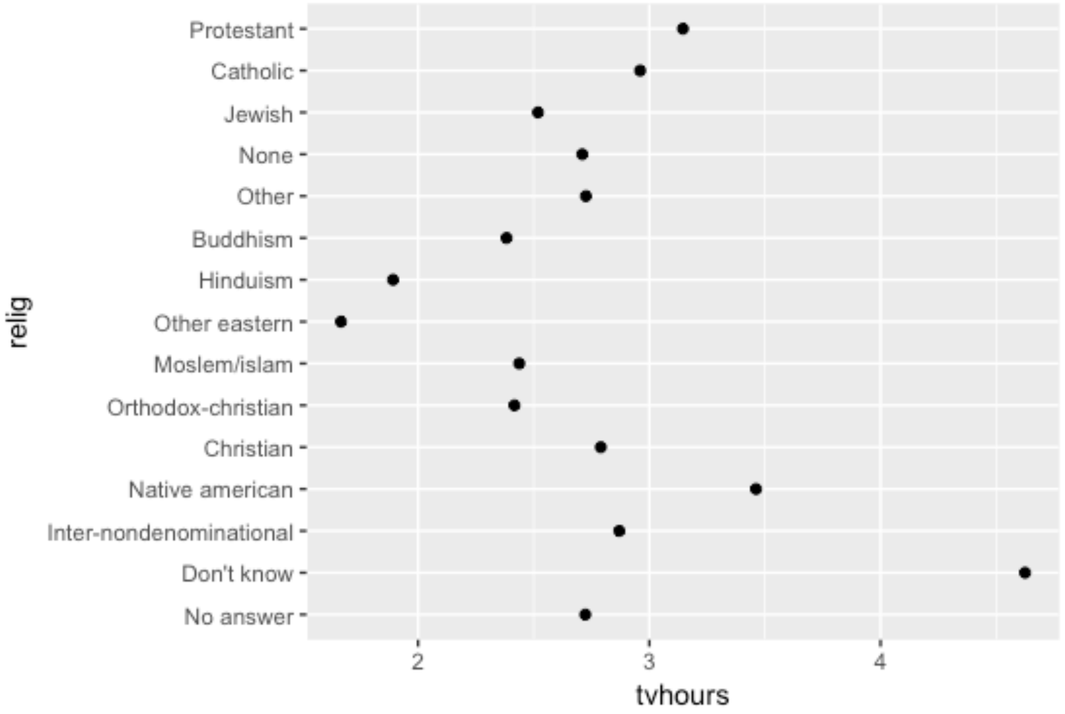
CHANGING LEVEL ORDER

CHANGING LEVEL ORDER

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

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

- Sometimes we want to change the order of our factors as we perform analyses
- This is often for visualization purposes

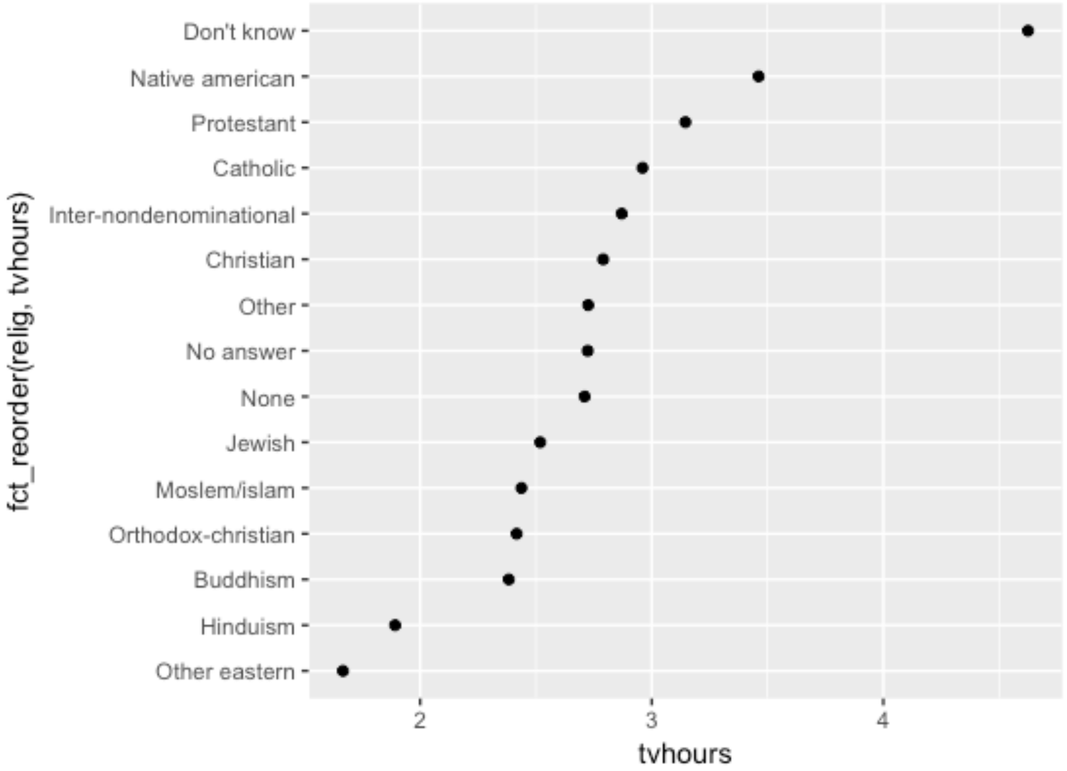


CHANGING LEVEL ORDER

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

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

- using `fct_reorder` allows us to order our variable based on another variables value

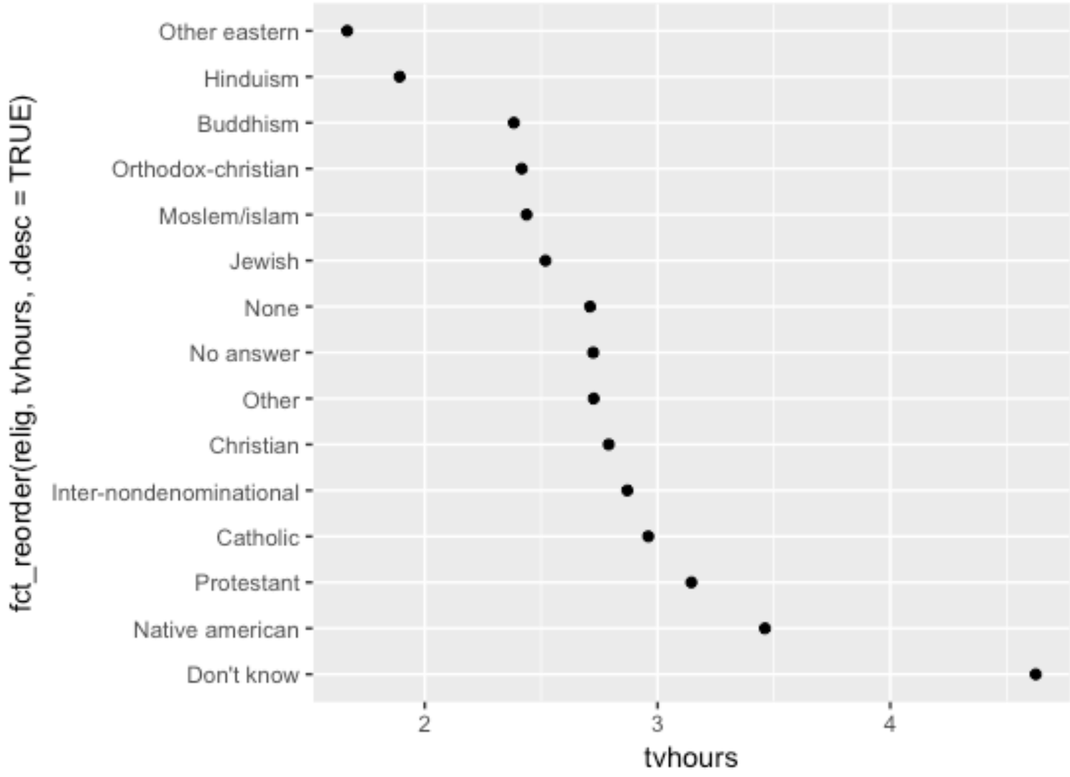


CHANGING LEVEL ORDER

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

```
ggplot(relig, aes(tvhours, fct_reorder(relig,
tvhours, .desc = TRUE))) +
  geom_point()
```

- add `.desc = TRUE` to reverse the order

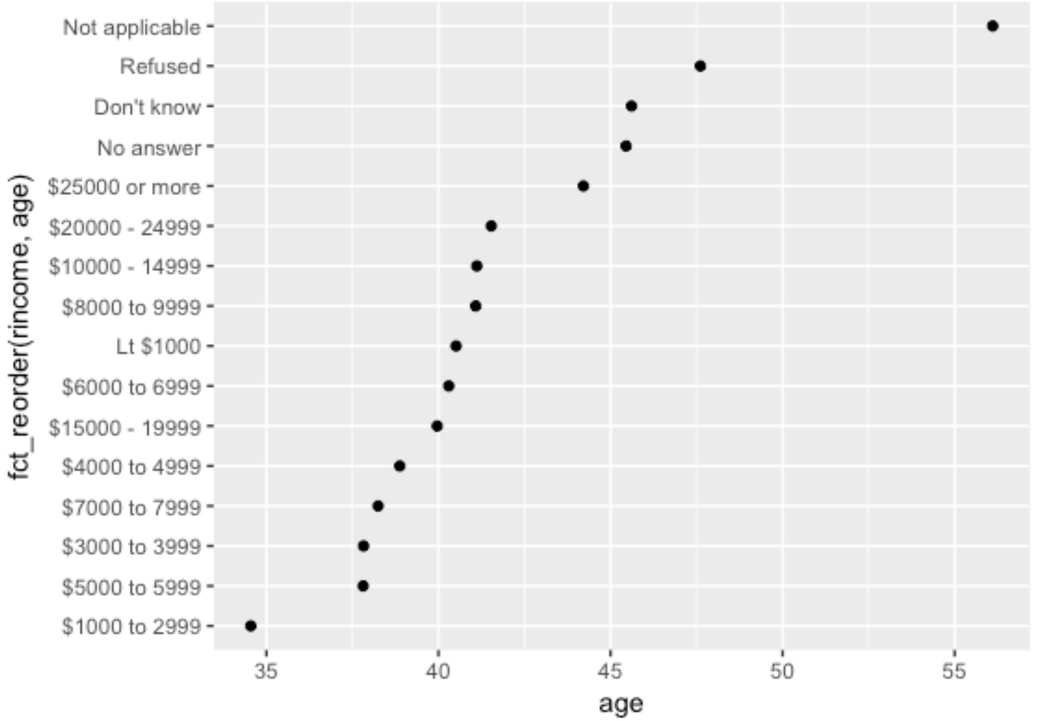


CHANGING LEVEL ORDER

```
rincome <- gss_cat %>%
  group_by(rincome) %>%
  summarise(
    age = mean(age, na.rm = TRUE),
    tvhours = mean(tvhours, na.rm = TRUE)
  )

ggplot(rincome, aes(age, fct_reorder(rincome, age))) +
  geom_point()
```

- Here's another example
- **Should we or should we not re-order the y-axis???**

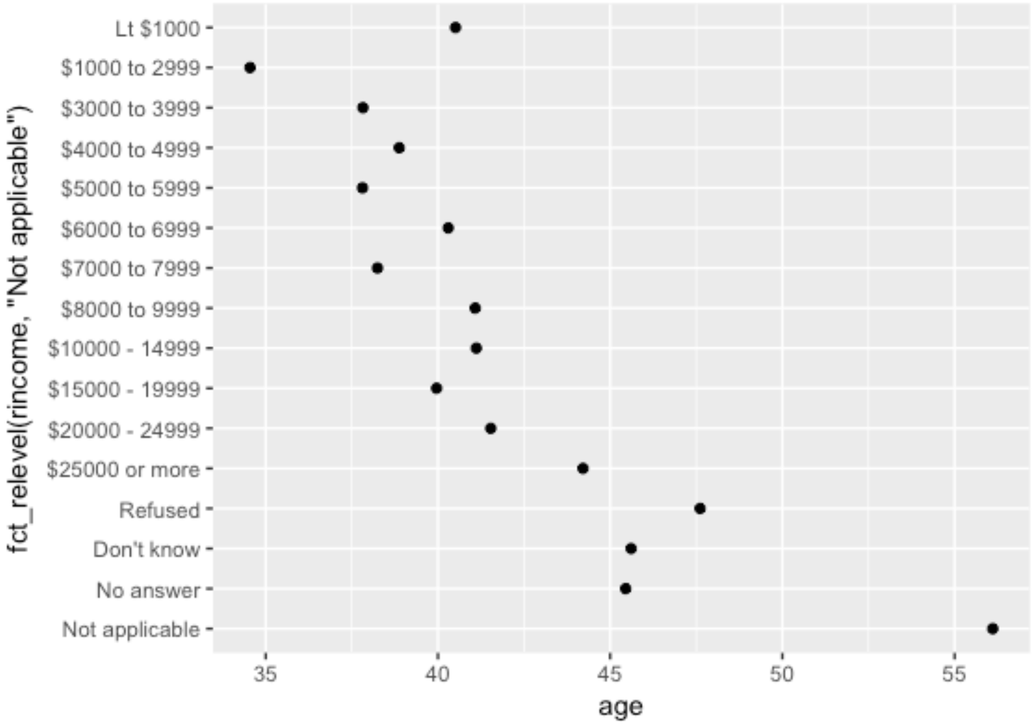


CHANGING LEVEL ORDER

```
rincome <- gss_cat %>%
  group_by(rincome) %>%
  summarise(
    age = mean(age, na.rm = TRUE),
    tvhours = mean(tvhours, na.rm = TRUE)
  )

ggplot(rincome, aes(age, fct_relevel(rincome, “Not Applicable"))) +
  geom_point()
```

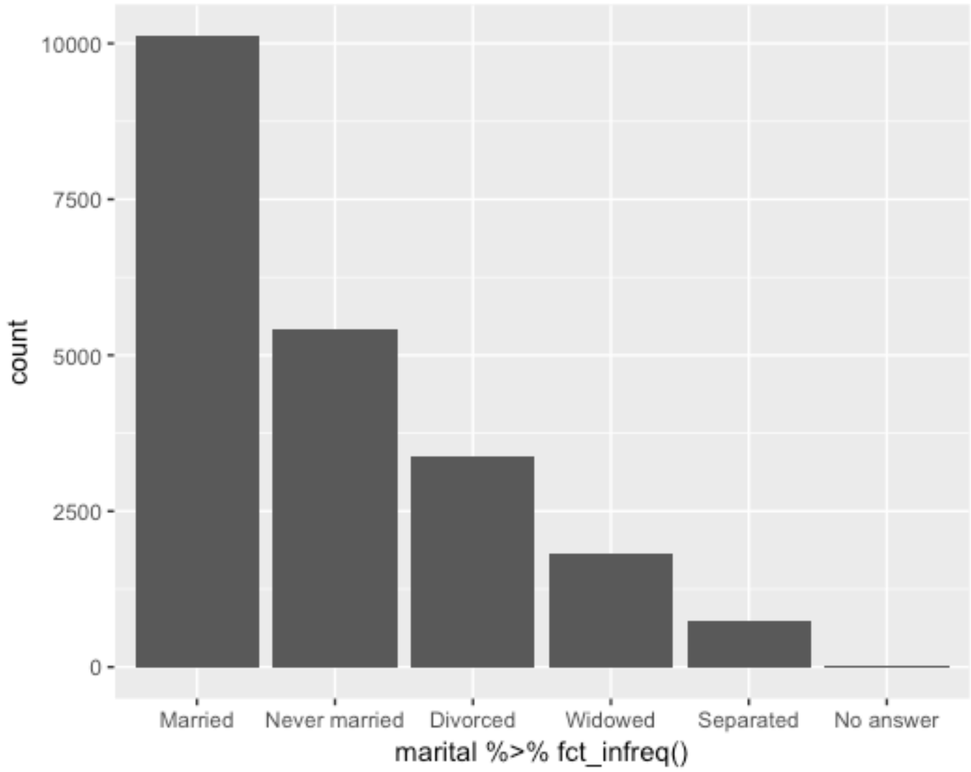
- A better idea is to move the “not applicable”, “no answer”, etc. to the bottom of the chart.
- We can do that with `fct_relevel`



CHANGING LEVEL ORDER

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

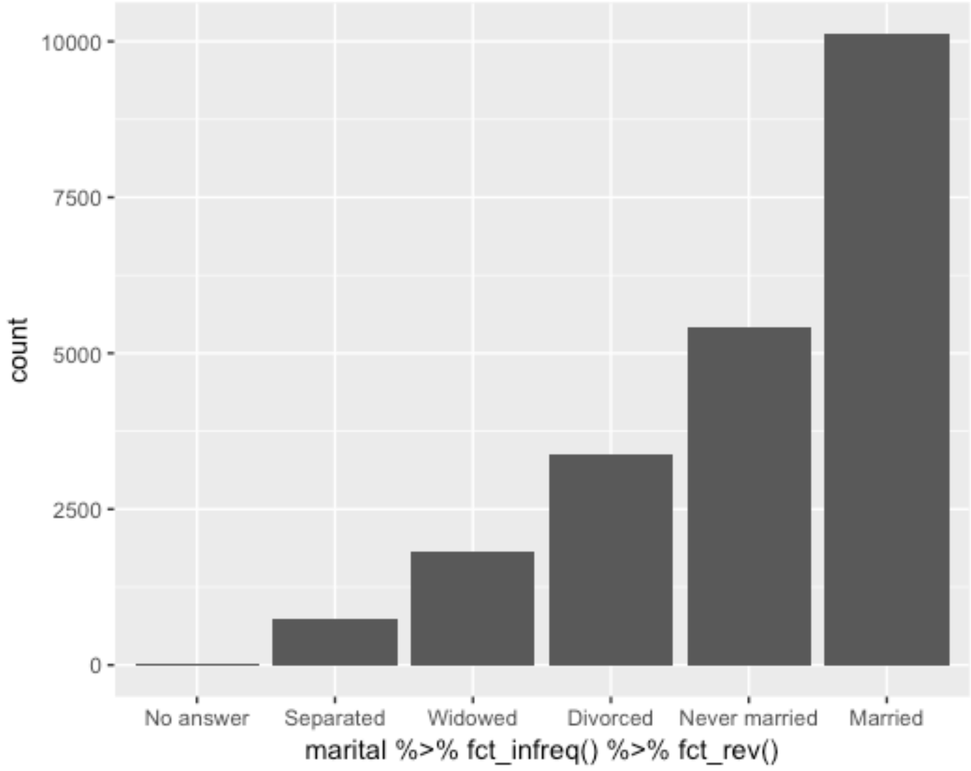
- Lastly, for bar charts you can use fct_infreq to plot the bars in order of frequency



CHANGING LEVEL ORDER

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

- Lastly, for bar charts you can use fct_infreq to plot the bars in order of frequency
- You can add fct_rev to reverse the order



YOUR TURN!

1. *Using the* `flights` *and* `airlines` *data perform the following:*

    1. *join the two data sets*

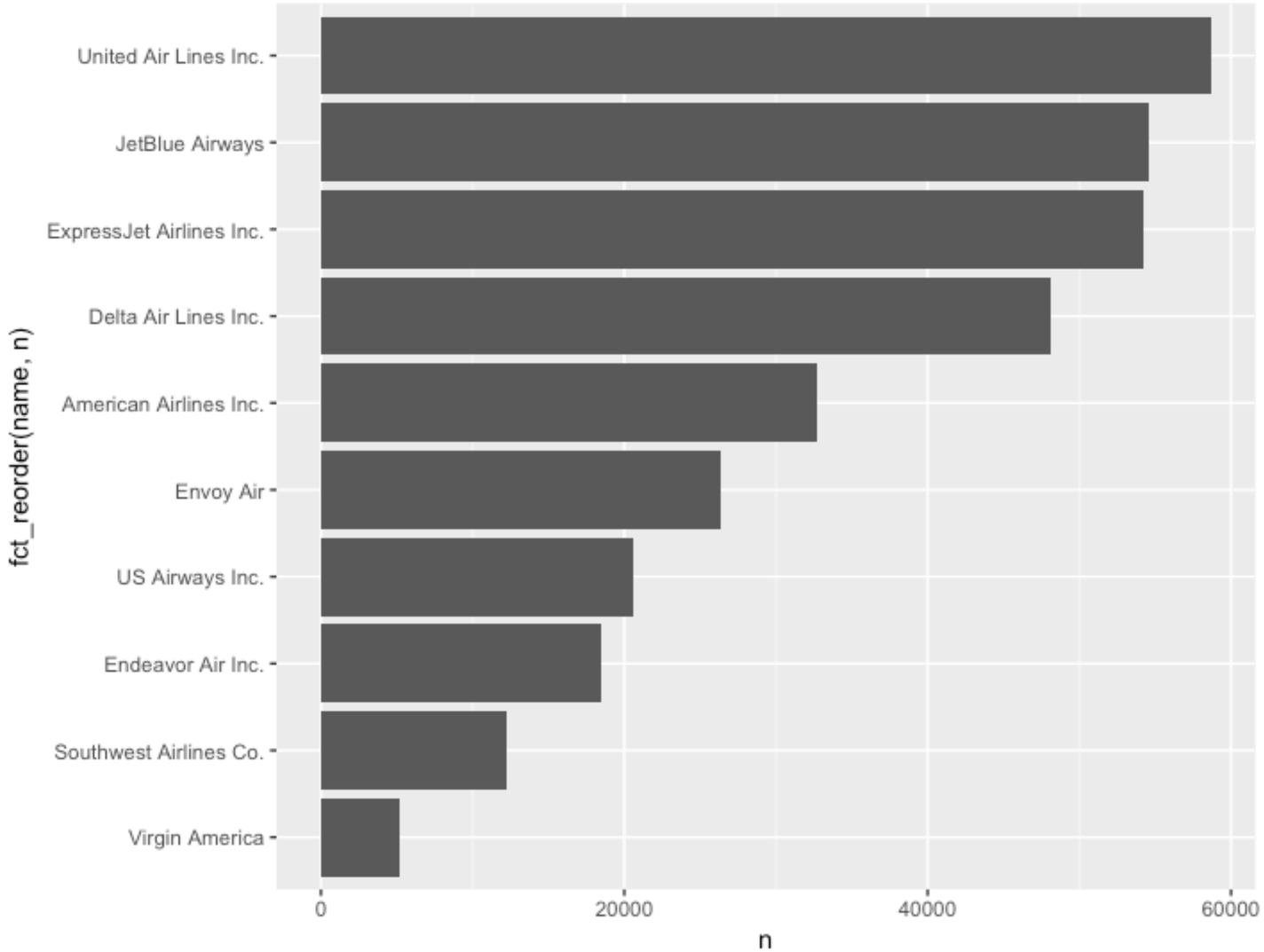
    2. *count the number of observations for each airline (i.e. Delta appears 48,110 times)*

    3. *identify the top 10 airlines (by name)*

    4. *create a bar chart that plots the number of observations for each airline and make sure the bar chart is ordered*

SOLUTION

```
flights %>%
  inner_join(airlines) %>%
  count(name, sort = TRUE) %>%
  top_n(10) %>%
  mutate(name = factor(name)) %>%
  ggplot(aes(fct_reorder(name, n), n)) +
  geom_bar(stat = "identity") +
  coord_flip()
```



WHAT TO REMEMBER

FUNCTIONS TO REMEMBER

Operator/Function	Description
`factor`	create a factor or coerce an existing vector into a factor vector
`fct_recode`	change the level of a factor
`fct_collapse`	collapse factors into groups
`fct_reorder`	reorder the levels of a function according to another variable
`fct_relevel`	change the order of levels in a factor by moving any number of levels to the front
`fct_infreq, fct_rev`	Reorder levels in order of frequency, reverse the levels