

Data Imports

STAT 220

Bastola

January 26 2022

Working Directories

The working directory is where R looks for files and saves files by default.

```
# see working directory  
getwd()  
# change your working directory  
setwd()
```

To set working directory to your STAT 220 course folder

```
setwd("path/to/stat220-folder/") # set  
getwd() # check
```

Useful Terminal Commands:

```
$ cd  
$ ls  
$ pwd
```

Web Imports

To your working folder:

```
url <- "https://raw.githubusercontent.com/deepbas/statdatasets/main/murders.csv"
dat <- read_csv(url)
```

To temporary folder:

```
download.file(url, "murders.csv")
tempfile()
[1] "/tmp/RtmpB1MBDh/file1d40843bf21128"
```

Download, store, and then remove:

```
tmp_filename <- tempfile()
download.file(url, tmp_filename)
mydat <- read_csv(tmp_filename)
file.remove(tmp_filename)
[1] TRUE
```

readr

- `readr` is a part of tidyverse library
- Includes functions for reading data stored in text file spreadsheets into R.
- Functions in the package include `read_csv()`, `read_tsv()`, `read_delim()` and more.
- These differ by the delimiter they use to split columns.



readr functions

function	reads
read_csv()	Comma separated values
read_csv2()	Semi-colon separated values
read_delim()	General delimited files
read_fwf()	Fixed width files
read_log()	Apache log files
read_table()	Space separated
read_tsv()	Tab delimited values

Basic syntax

All `readr` functions share a common syntax

```
df <- read_csv(file = "path/to/file.csv", ...)
```

Base R Imports

- R-base import functions
 - `read.csv()`
 - `read.table()`
 - `read.delim()`
- Generate data frames rather than tibbles
- Character variables are converted to factors
 - Can be avoided by setting the argument `stringsAsFactors=FALSE`

Advantages of readr

- readr functions are:
 - ~ 10 times faster
 - Return tibbles
 - Have more intuitive defaults.
 - No row names, no strings as factors.

Data frames and tibbles Conversion



- `as_tibble()` - convert a data frame to a tibble
- `as.data.frame()` - convert a tibble to a data frame

Your turn 1

Please git clone the repository on [Data Imports](#) to your local folder.

Task: Use `read_csv()` to import the **desserts** data set from <https://raw.githubusercontent.com/deepbas/statdatasets/main/desserts.csv>

Store the data in the **desserts** object

Overview of data

- Contains results from series 1-8 of *The Great British Bake Off*
- Case defined by series, episode and baker
- Data set created by Alison Hill

03:00

Did it work as expected?

```
Rows: 549
Columns: 16
$ series      <dbl> 1, 1, 1, 1, 1, 1, 1, 1
$ episode    <dbl> 1, 1, 1, 1, 1, 1, 1, 1
$ baker      <chr> "Annetha", "David", "E
$ technical  <chr> "2nd", "3rd", "1st", "
$ result     <chr> "IN", "IN", "IN", "IN"
$ uk_airstate <chr> "17 August 2010", "17
$ us_season  <dbl> NA, NA, NA, NA, NA, NA
$ us_airstate <date> NA, NA, NA, NA, NA, N
$ showstopper_chocolate <chr> "chocolate", "chocolat
$ showstopper_dessert   <chr> "other", "other", "oth
$ showstopper_fruit     <chr> "no fruit", "no fruit"
$ showstopper_nut       <chr> "no nut", "no nut", "n
$ signature_chocolate  <chr> "no chocolate", "choco
$ signature_dessert     <chr> "cake", "cake", "cake"
$ signature_fruit       <chr> "no fruit", "fruit", "
$ signature_nut         <chr> "no nut", "no nut", "n
```

Something to note ..

- `technical` is character, not numeric
- `uk_airstate` is character, not date

The col_types argument

By default, looks at first 1000 rows to guess variable data types (`guess_max`)

```
desserts <- read_csv(  
  "https://raw.githubusercontent.com/deepbas/statdatasets/main/desserts.csv",  
  col_types = list(  
    technical = col_number(),  
    uk_airstate = col_date()  
  )  
)
```

Looking for problems

List of potential problems parsing the file

```
problems(desserts)
# A tibble: 556 × 5
```

	row <int>	col <int>	expected <chr>	actual <chr>	file <chr>
1	2	6	date in ISO8601	17 August 2010	""
2	3	6	date in ISO8601	17 August 2010	""
3	4	6	date in ISO8601	17 August 2010	""
4	5	4	a number	N/A	""
5	5	6	date in ISO8601	17 August 2010	""
6	6	6	date in ISO8601	17 August 2010	""
7	7	4	a number	N/A	""
8	7	6	date in ISO8601	17 August 2010	""
9	8	6	date in ISO8601	17 August 2010	""
10	9	4	a number	N/A	""

```
# ... with 546 more rows
```

Date formatting

```
# A tibble: 556 × 5
  row   col expected      actual      file
<int> <int> <chr>      <chr>      <chr>
1     2     6 date in ISO8601 17 August 2010 ""
2     3     6 date in ISO8601 17 August 2010 ""
3     4     6 date in ISO8601 17 August 2010 ""
4     5     4 a number      N/A      ""
5     5     6 date in ISO8601 17 August 2010 ""
# ... with 551 more rows
```

ISO8601 format: 2021-10-04

What we have: 17 August 2010

Adding format instructions

```
desserts <- read_csv(  
  "https://raw.githubusercontent.com/deepbas/statdatasets/main/desserts.csv",  
  col_types = list(  
    technical = col_number(),  
    uk_airdate = col_date(format = "%d %B %Y")  
  )  
)
```

- Year: "%Y" (4 digits), "%y" (2 digits)
- Month: "%m" (2 digits), "%b" (abbreviated name in current locale), "%B" (full name in current locale).
- Day: "%d" (2 digits), "%e" (optional leading space)

Looking for more problems

List of potential problems parsing the file

```
problems(desserts)
# A tibble: 7 × 5
  row    col expected actual file
  <int> <int> <chr>    <chr> <chr>
1     5     4 a number N/A    ""
2     7     4 a number N/A    ""
3     9     4 a number N/A    ""
4    11     4 a number N/A    ""
5    35     4 a number N/A    ""
6    36     4 a number N/A    ""
7    37     4 a number N/A    ""
```


Addressing missing values

By default `na = c("", "NA")` are the recognized missing values

```
desserts <- read_csv(  
  "https://raw.githubusercontent.com/deepbas/statdatasets/main/desserts.csv",  
  col_types = list(  
    technical = col_number(),  
    uk_airdate = col_date(format = "%d %B %Y")  
  ),  
  na = c("", "NA", "N/A")  
)
```

No more problems

```
problems(desserts)  
# A tibble: 0 × 5  
# ... with 5 variables: row <int>, col <int>, expected <chr>, actual <chr>,  
#   file <chr>
```

The Dataset

```
# A tibble: 549 × 16
  series episode baker      technical result uk_airstate us_season us_airstate
  <dbl>   <dbl> <chr>      <dbl> <chr>   <date>      <dbl> <date>
1     1     1     1 Annetha      2 IN    2010-08-17      NA NA
2     1     1     1 David       3 IN    2010-08-17      NA NA
3     1     1     1 Edd         1 IN    2010-08-17      NA NA
4     1     1     1 Jasminder  NA IN    2010-08-17      NA NA
5     1     1     1 Jonathan   9 IN    2010-08-17      NA NA
6     1     1     1 Louise    NA IN    2010-08-17      NA NA
7     1     1     1 Miranda    8 IN    2010-08-17      NA NA
8     1     1     1 Ruth      NA IN    2010-08-17      NA NA
9     1     1     1 Lea      10 OUT   2010-08-17      NA NA
10    1     1     1 Mark      NA OUT   2010-08-17      NA NA
# ... with 539 more rows, and 8 more variables: showstopper_chocolate <chr>,
# showstopper_dessert <chr>, showstopper_fruit <chr>, showstopper_nut <chr>,
# signature_chocolate <chr>, signature_dessert <chr>, signature_fruit <chr>,
# signature_nut <chr>
```

Column casting functions

Type	dplyr::glimpse()	readr::col_*
logical	<lgl>	col_logical
numeric	<int> or <dbl>	col_number
character	<chr>	col_character
factor	<fct>	col_factor
date	<date>	col_date

?read_csv

```
read_csv(file,  
         col_names = TRUE,  
         col_types = NULL,  
         locale = default_locale(),  
         na = c("", "NA"),  
         quoted_na = TRUE,  
         quote = "\"",  
         comment = "",  
         trim_ws = TRUE,  
         skip = 0,  
         n_max = Inf,  
         guess_max = min(1000, n_max),  
         progress = show_progress())
```

Your turn 2

Use the appropriate `read_<type>()` function to import the following data sets:

- `simple-1.dat`
- `mild-1.csv`
- `tricky-1.csv`

If you hit any errors/problems, be sure to explore them and identify the issue, even if you can't "fix" it.

Remember, it might help to look at the data first!

03:00

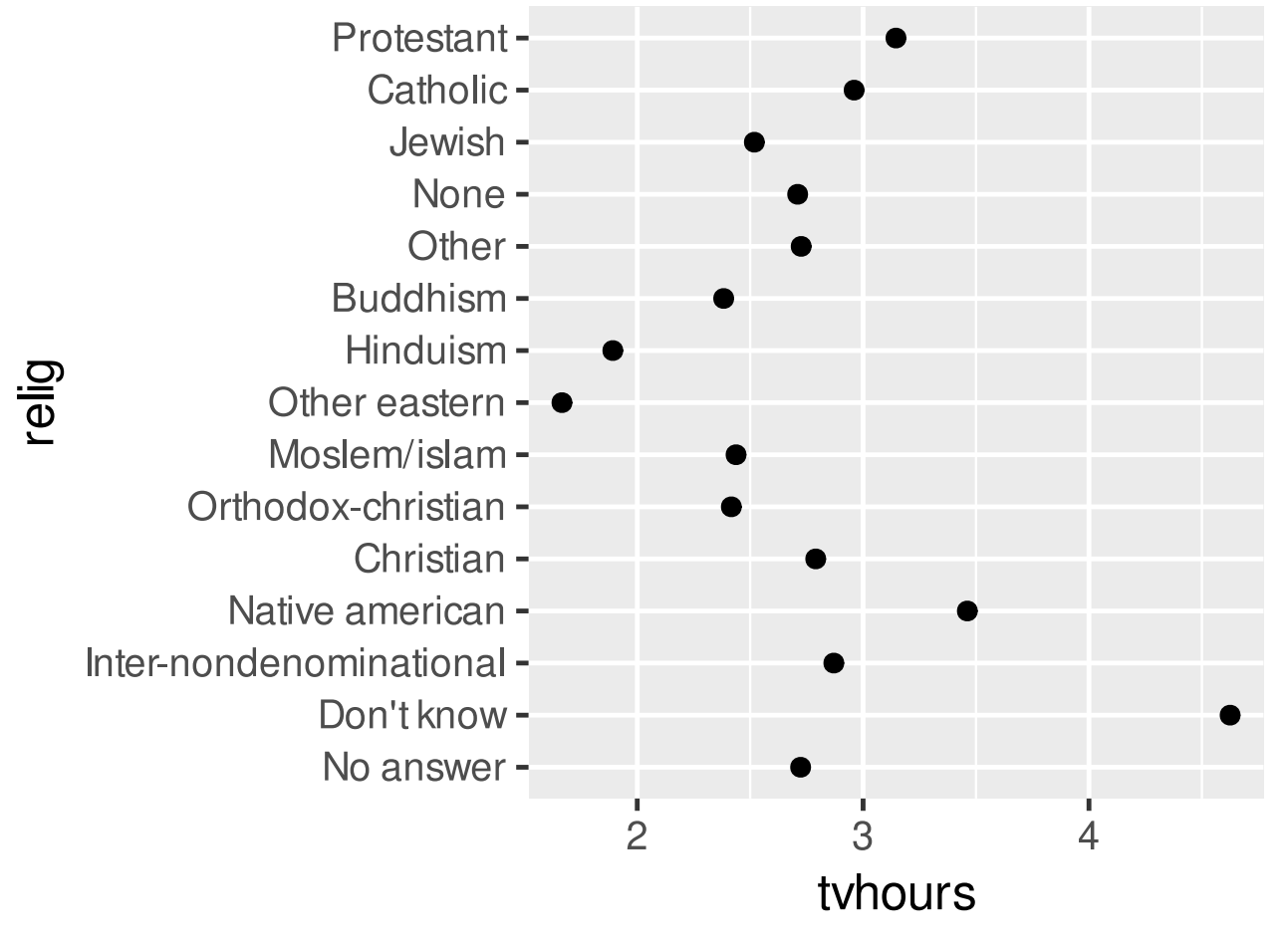
More with forcats: gss_cat

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

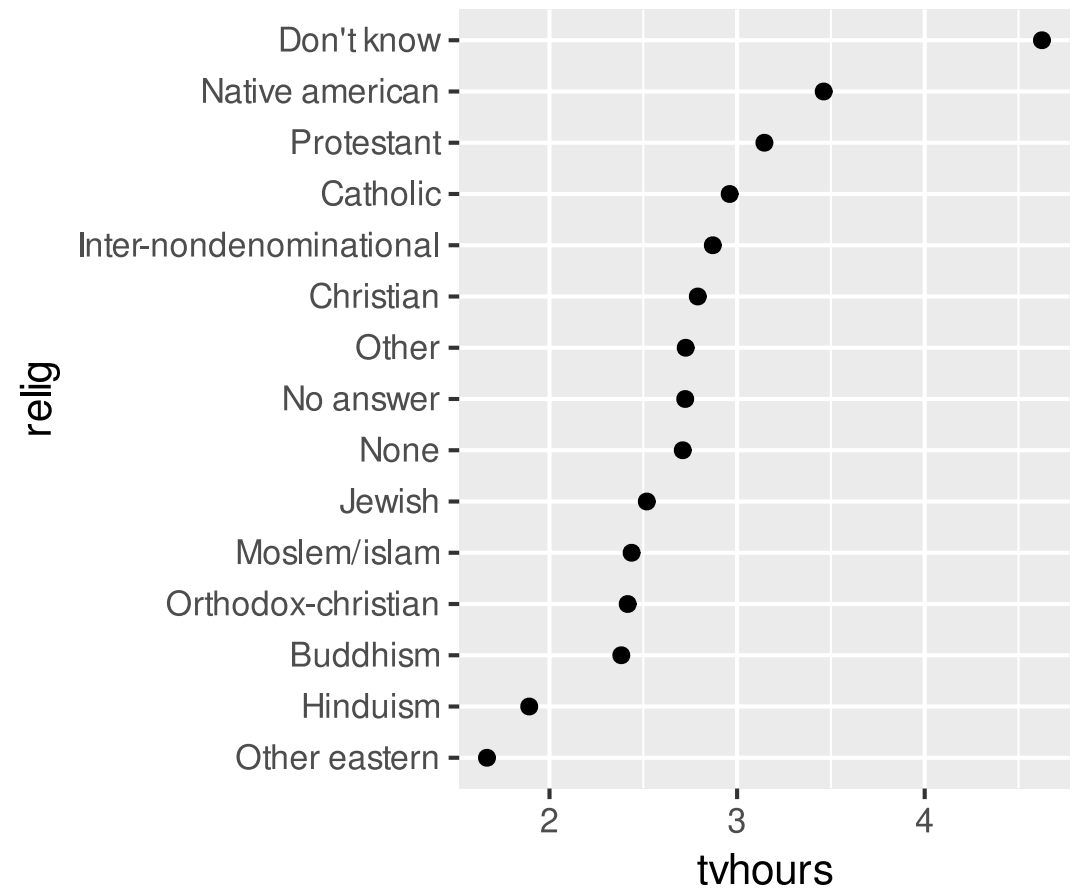
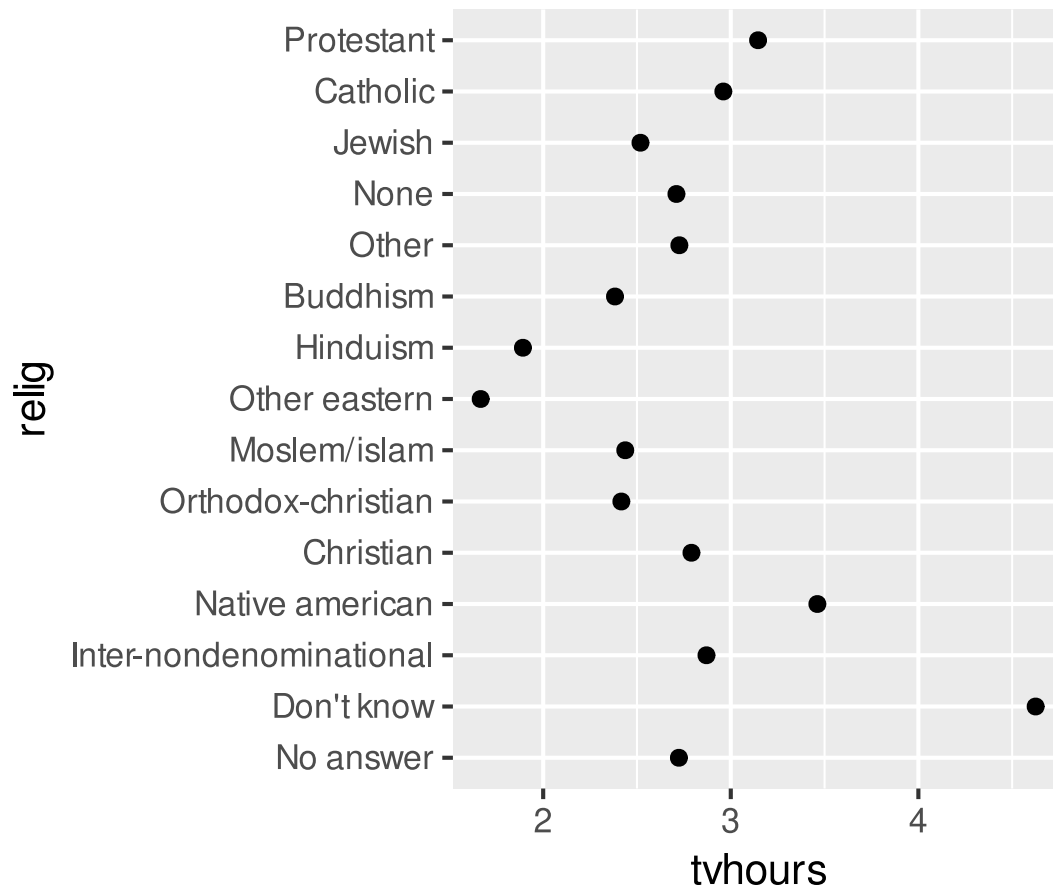
```
# A tibble: 21,483 × 9
  year marital      age race  rincome      partyid  relig  denom  tvhours
  <int> <fct>      <int> <fct> <fct>      <fct>    <fct> <fct>    <int>
1  2000 Never married    26 White $8000 to 9999 Ind,nea... Prote... South...    12
2  2000 Divorced        48 White $8000 to 9999 Not str... Prote... Bapti...   NA
3  2000 Widowed        67 White Not applicable Indepen... Prote... No de...     2
4  2000 Never married    39 White Not applicable Ind,nea... Ortho... Not a...     4
5  2000 Divorced        25 White Not applicable Not str... None    Not a...     1
6  2000 Married        25 White $20000 - 24999 Strong ... Prote... South...   NA
7  2000 Never married    36 White $25000 or more Not str... Chris... Not a...     3
8  2000 Divorced        44 White $7000 to 7999 Ind,nea... Prote... Luthe...   NA
9  2000 Married        44 White $25000 or more Not str... Prote... Other      0
10 2000 Married        47 White $25000 or more Strong ... Prote... South...     3
# ... with 21,473 more rows
```

Which religions watch the least TV?

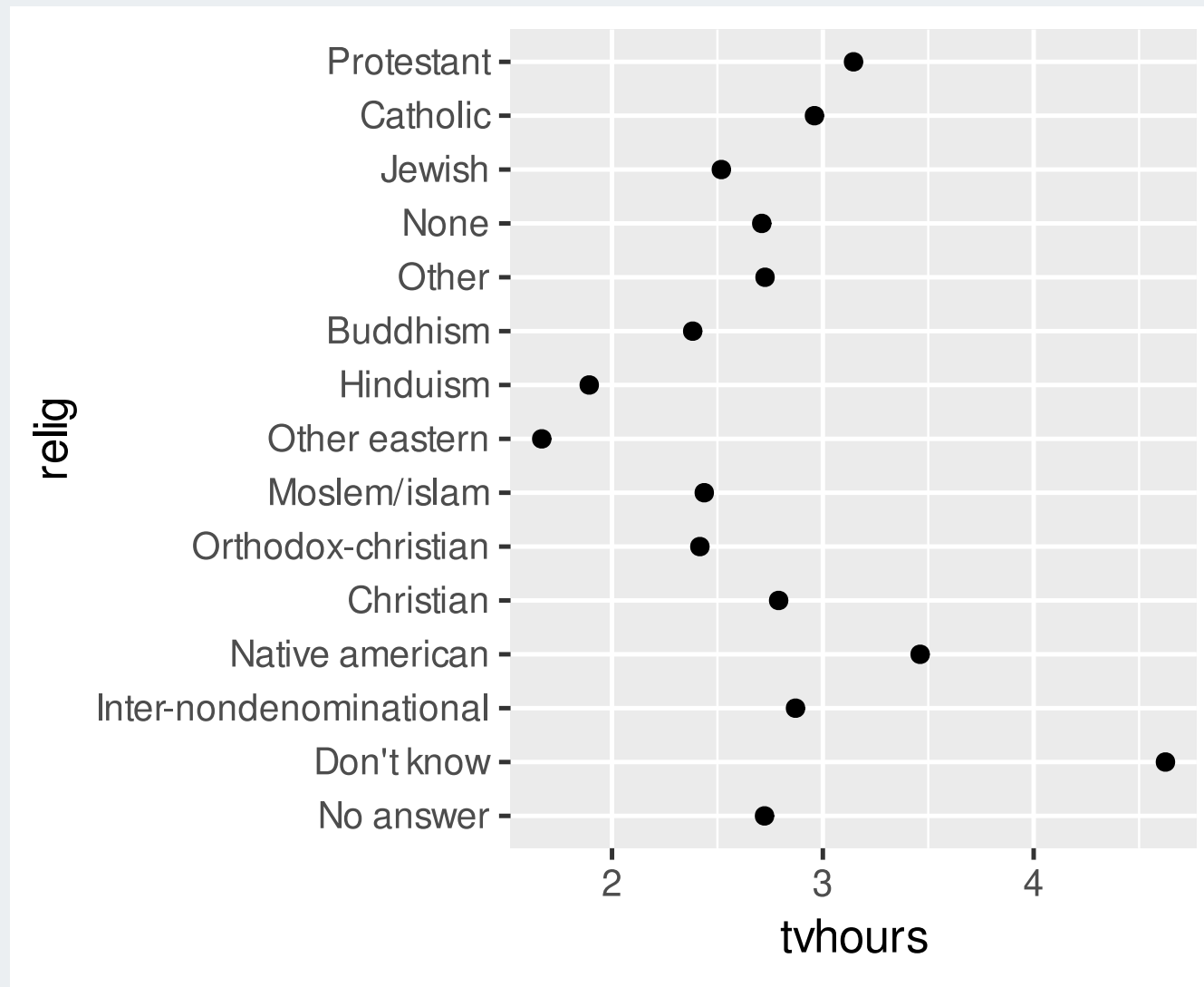
```
gss_cat %>%  
  drop_na(tvhours) %>%  
  group_by(relig) %>%  
  summarize(tvhours = mean(tvhours))  
ggplot(aes(tvhours, relig)) +  
  geom_point()
```



Which one do you prefer?



Why is the y-axis in this order?



levels()

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

```
gss_cat %>% pull(relig) %>% levels()
[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"
```

Most useful factor skills

1. **Reorder** the levels
2. **Recode** the levels
3. **Collapse** levels
4. **Lump** levels

fct_reorder

- .f factor vector

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

fct_reorder

- `.f` factor vector
- `.x` variable to reorder by (in conjunction with `.fun`)

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

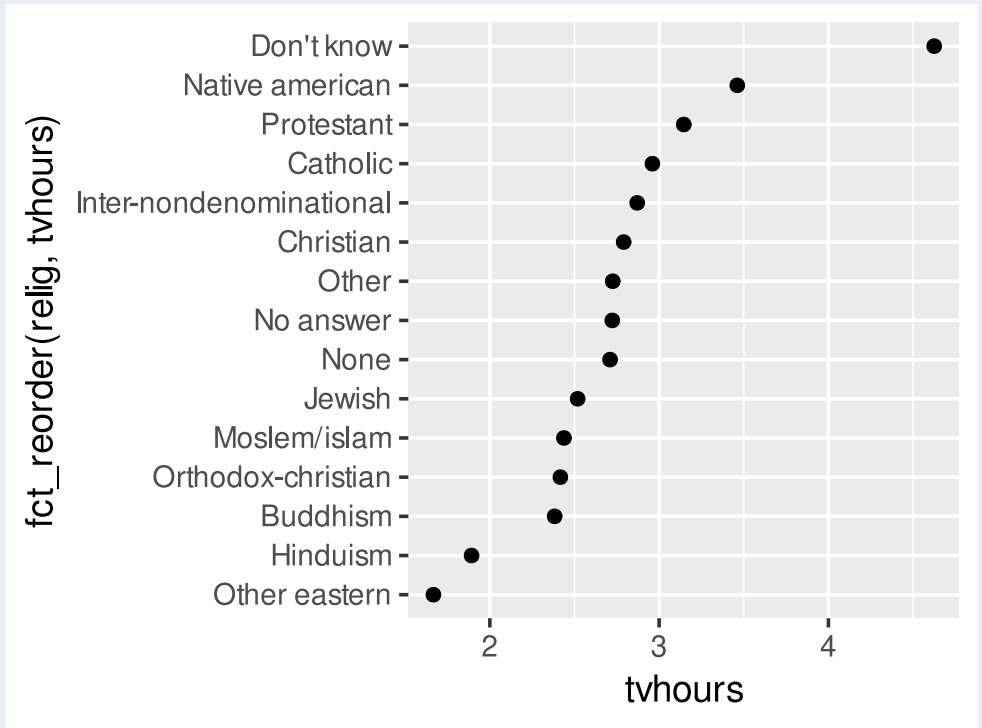
fct_reorder

- `.f` factor vector
- `.x` variable to reorder by (in conjunction with `.fun`)
- `.fun` function to reorder by

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

Reorder relig by tvhours

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



Your turn 4

Use `rincome_summary` to construct a dotplot of `rincome` against `age`.

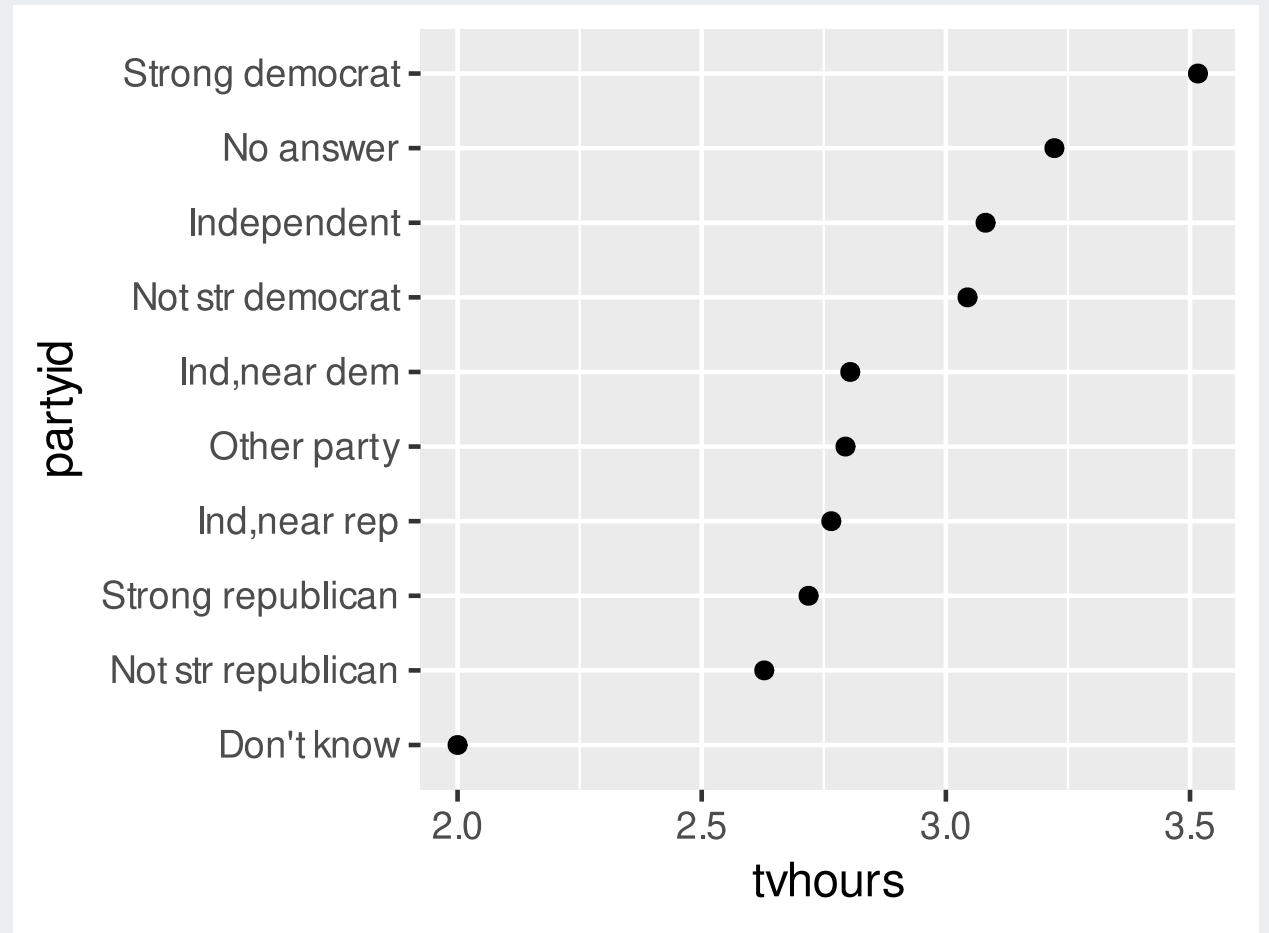
Reorder `rincome` by `age`

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

01:30

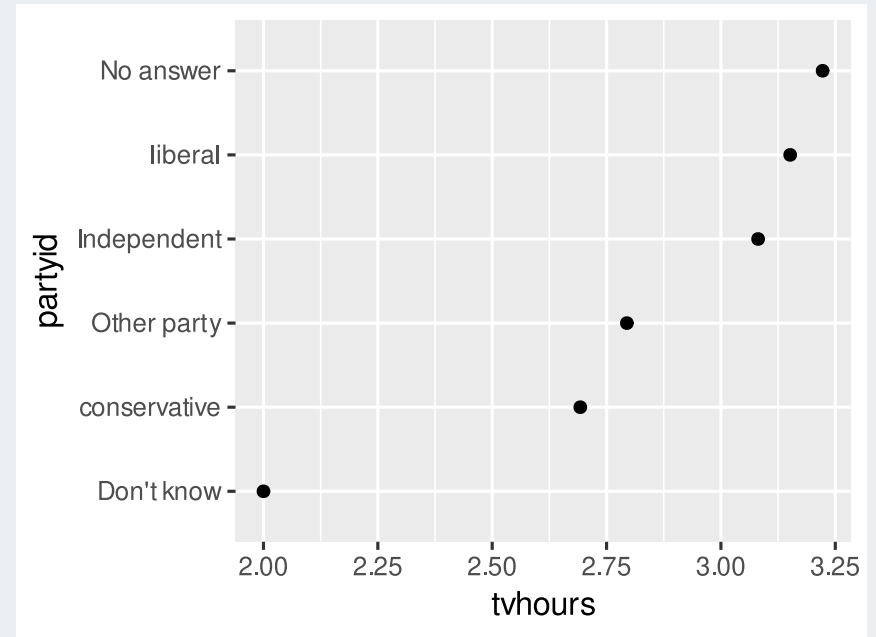
Which political leaning watches more TV?

How could we improve the `partyid` labels?



Collapsing partyid: fct_collapse()

```
gss_cat %>%  
  drop_na(tvhours) %>%  
  select(partyid, 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) %>%  
  summarize(tvhours = mean(tvhours)) %>%  
  ggplot(aes(tvhours, fct_reorder(partyid, tvhours)))  
  geom_point() +  
  labs(y = "partyid")
```

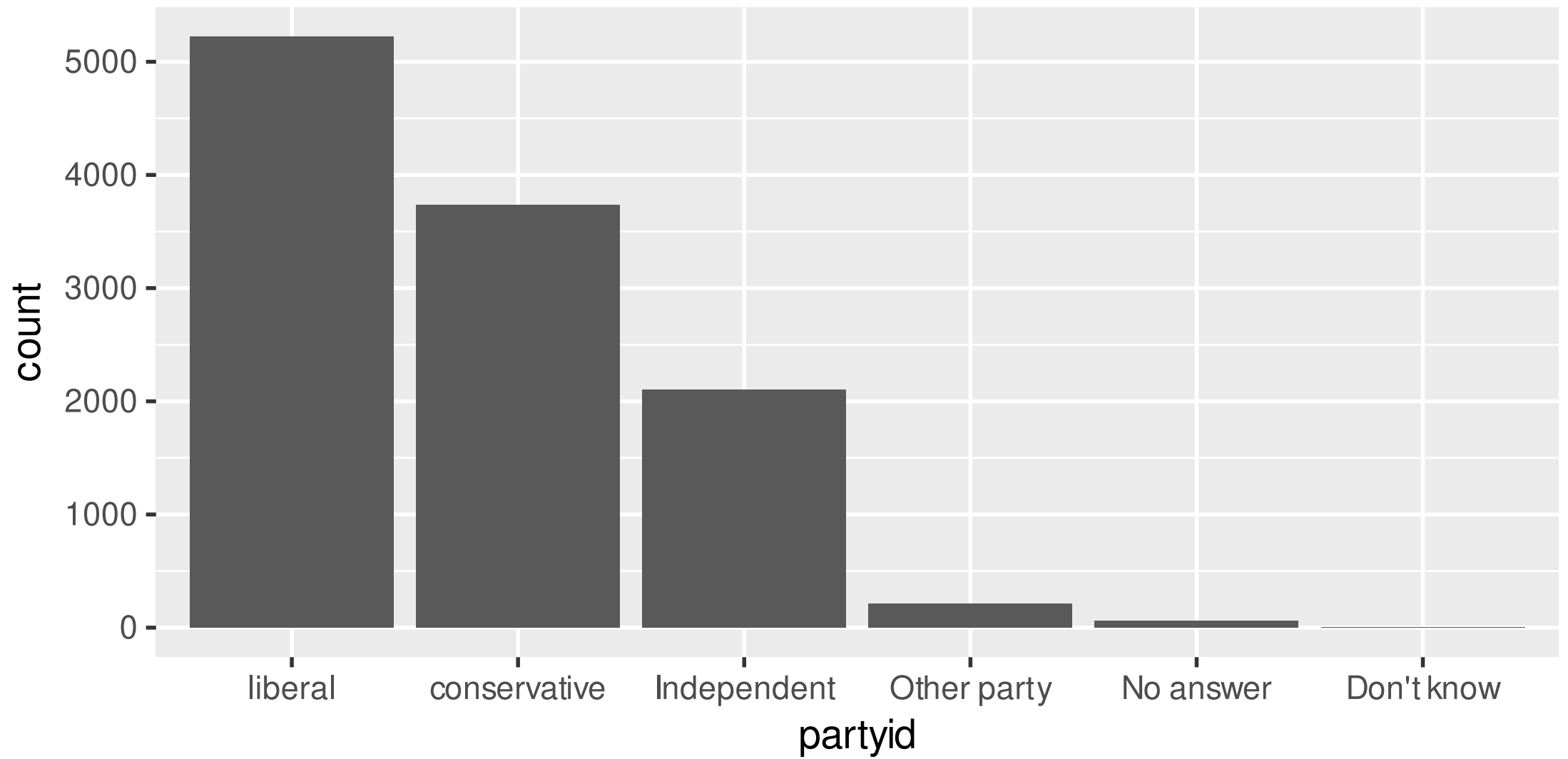


Your turn 5

Collapse the `marital` variable to have levels `Married`, `not_married`, and `No answer`

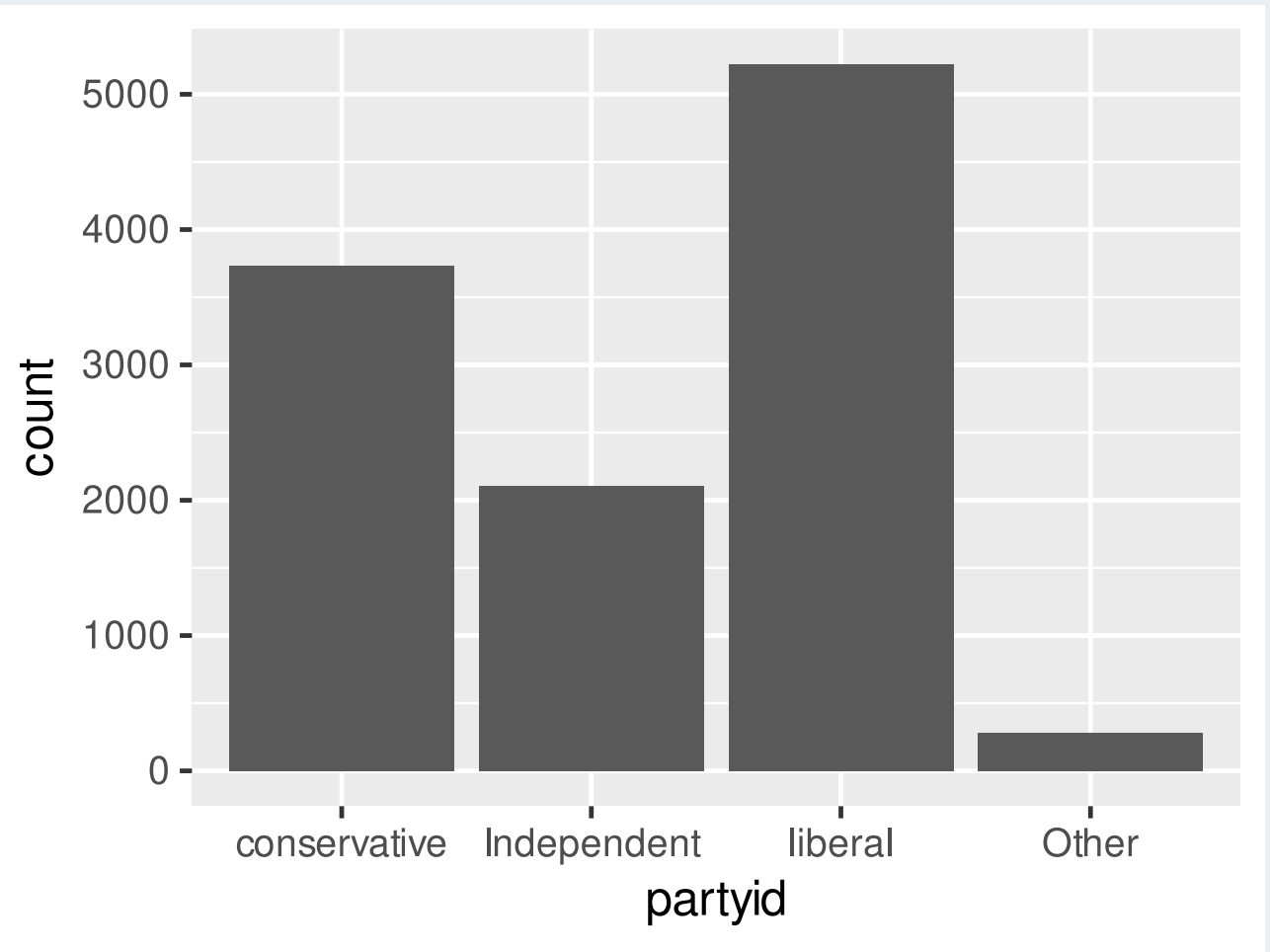
Include `"Never married"`, `"Divorced"`, and `"Widowed"` in `not_married`

02:00



Lumping partyid

```
gss_cat %>%  
  mutate(partyid = fct_lump(partyid,  
    ggplot(aes(partyid)) +  
    geom_bar() +  
    labs(x = "partyid")
```



Acknowledgements

These slides were adapted from Adam Loy's instruction materials on data imports and manipulating factors and are licensed under the CC BY 4.0 Creative Commons.