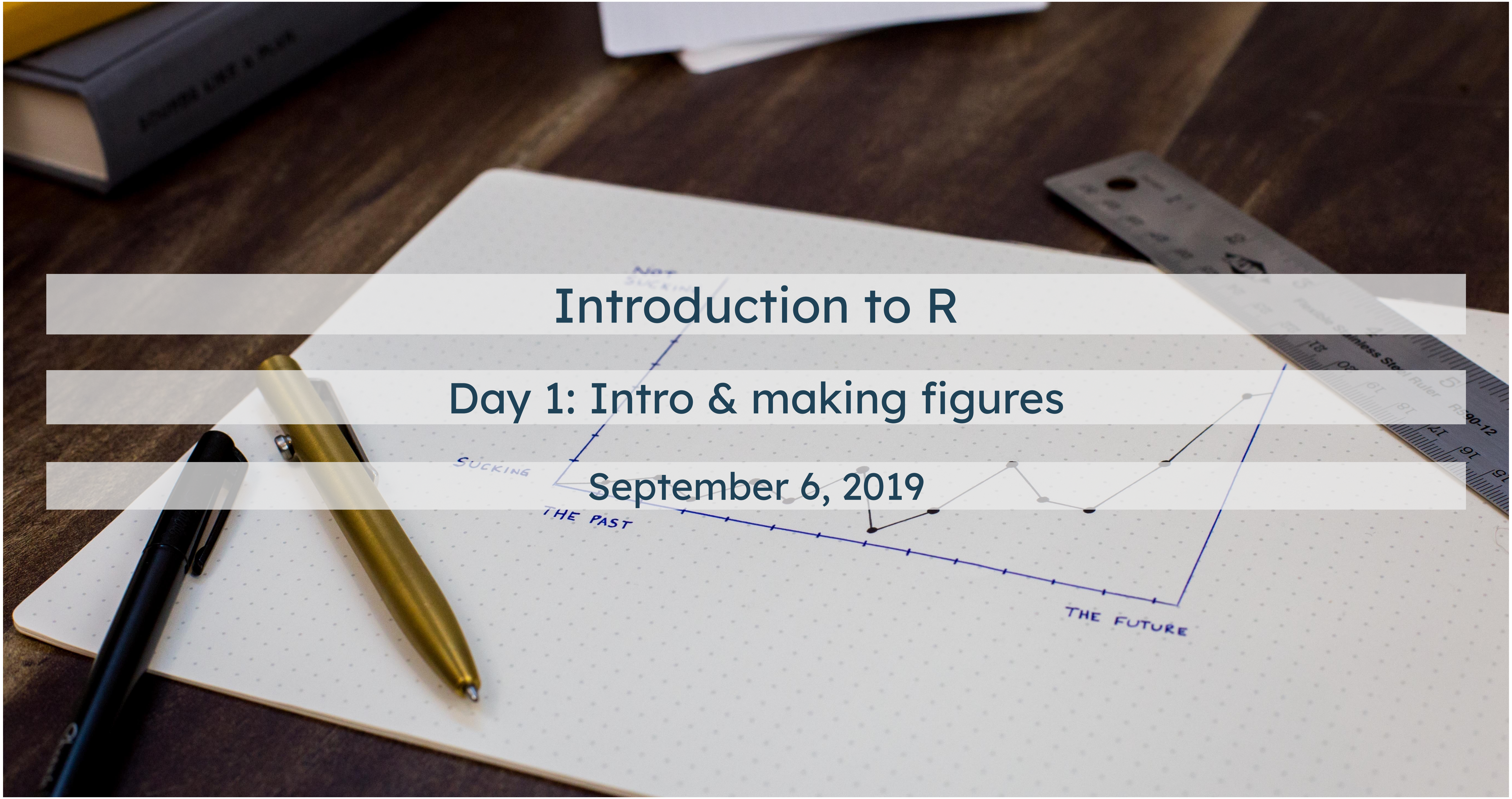
Introduction to R

Day 1: Intro & making figures

September 6, 2019



About this class

- Non-credit
- 5 sessions
- "Challenges" but no homework

Work hard with each other during class

Try to figure it out on your own before you ask for help

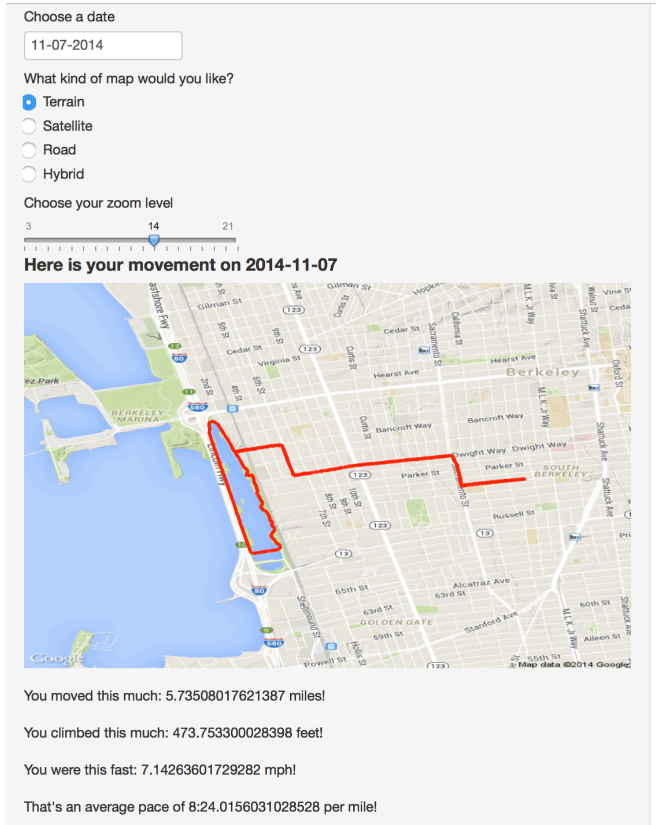
Practice by yourself in between classes

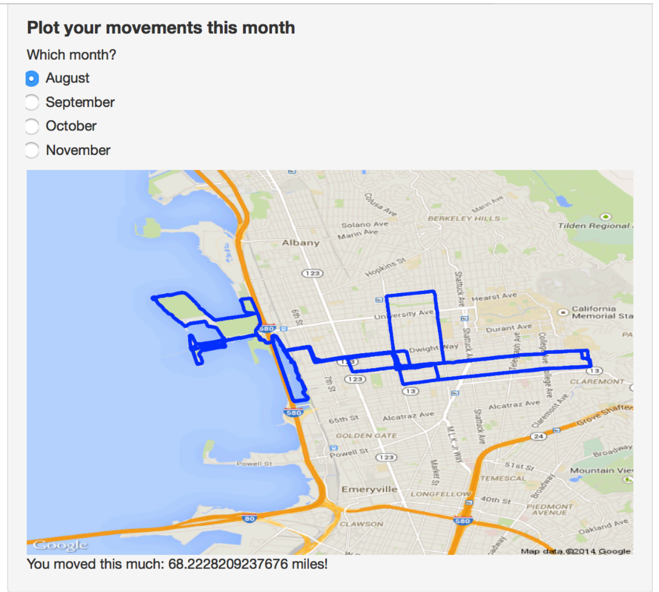
You are not going to break anything!

Anyone can learn to use R, it's just a matter of sitting down and doing it. Now's your chance!

About me

- 4th-year PhD candidate in Epidemiology
- Started using R during my masters (so 5 years of experience); learned mostly by doing
- Problem sets, manuscripts, slides, website all in R (www.louisahsmith.com)
- Almost 100 R projects on my computer, including over 1000 R scripts





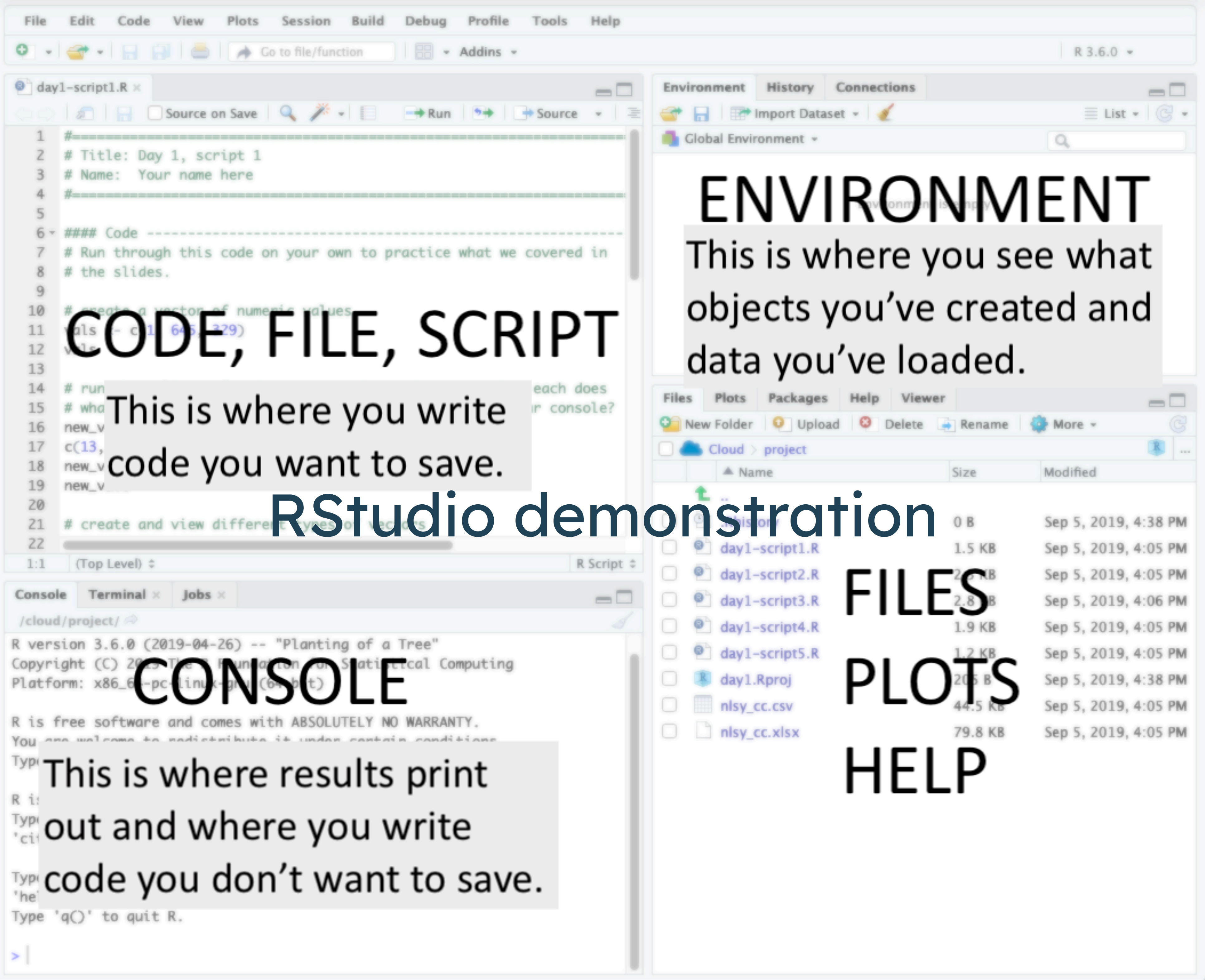
I have to Google things literally every time I use R!

An IDE for R

An *integrated development environment* is software that makes coding easier

- see objects you've imported and created
- autocomplete
- syntax highlighting
- run part or all of your code

RStudio demonstration



R uses <- for assignment

Create an object `vals` that contains and sequence of numbers:

```
# create values
vals <- c(1, 645, 329)
```

Put your cursor at the end of the line and hit ctrl/cmd + enter.

Now `vals` holds those values.

We can see them again by running just the name (put your cursor after the name and press ctrl/cmd + enter again).

```
vals
```

```
## [1]   1 645 329
```

No assignment arrow means that the object will be printed to the console.

Types of data (*classes*)

We could also create a character *vector*:

```
chars <- c("dog", "cat", "rhino")
chars
```

```
## [1] "dog"   "cat"   "rhino"
```

Or a *logical* vetor:

```
logs <- c(TRUE, FALSE, FALSE)
logs
```

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

We'll see more options as we go along!

Types of objects

We created *vectors* with the `c()` function (`c` stands for concatenate)

We could also create a *matrix* of values with the `matrix()` function:

```
# turn the vector of numbers into a 2-row matrix
mat <- matrix(c(234, 7456, 12, 654, 183, 753), nrow = 2)
mat
```

```
##      [,1] [,2] [,3]
## [1,]  234   12  183
## [2,] 7456  654  753
```

The numbers in square brackets are *indices*, which we can use to pull out values:

```
# extract second row
mat[2, ]
```

```
## [1] 7456  654  753
```

Exercises 1

1. Extract `645` from `vals` using square brackets
2. Extract `"rhino"` from `chars` using square brackets
3. You saw how to extract the second row of `mat`. Figure out how to extract the second column.
4. Extract `183` from `mat` using square brackets
5. Figure out how to get the following errors:

```
## [1] "incorrect number of dimensions"
```

```
## [1] "subscript out of bounds"
```

Dataframes

We usually do analysis in R with dataframes (or some variant).

Dataframes are basically like spreadsheets: columns are variables, and rows are observations.

```
gss_cat
```

```
## # A tibble: 21,483 x 9
##     year marital         age race  rincome        partyid        relig          denom          tvhours
##    <int> <fct>         <int> <fct> <fct>          <fct>          <fct>          <fct>            <int>
##  1  2000 Never marr…      26 White $8000 to 99… Ind,near rep   Protestant     Southern ba…        12
##  2  2000 Divorced         48 White $8000 to 99… Not str repu… Protestant     Baptist-dk …        NA
##  3  2000 Widowed          67 White Not applica… Independent    Protestant     No denomina…         2
##  4  2000 Never marr…      39 White Not applica… Ind,near rep   Orthodox-ch… Not applica…         4
##  5  2000 Divorced         25 White Not applica… Not str demo… None           Not applica…         1
##  6  2000 Married          25 White $20000 - 24… Strong democ… Protestant     Southern ba…        NA
##  7  2000 Never marr…      36 White $25000 or m… Not str repu… Christian      Not applica…         3
##  8  2000 Divorced         44 White $7000 to 79… Ind,near dem   Protestant     Lutheran-mo…        NA
##  9  2000 Married          44 White $25000 or m… Not str demo… Protestant     Other                0
## 10  2000 Married          47 White $25000 or m… Strong repub… Protestant     Southern ba…         3
## # … with 21,473 more rows
```

tibble???

Packages in R

Although R comes with a number of functions (and datasets! try running `data()`), you can also add on lots of **packages**.

Many packages can be found on CRAN, which is what R goes to automatically when you run `install.packages("packagename")`.

Other packages live only on GitHub, or in other repositories. To download these, you will have to use something like `remotes::install_github("developer/package")` or similar.

You only need to install a package once (until it needs to be updated, or you update R). But every time you want to use a package, you need to include `library(packagename)` at the top of your script, and run that before you run any functions.

tidyverse

The tidyverse is a collection of packages for R that are designed to make working with data easy and intuitive.

You might hear it contrasted with "base R" or the package `data.table`. You can (and should!) learn as many coding techniques and strategies as possible, then choose the best option (in terms of speed, readability, etc.) for you.

I find tidyverse the quickest and most intuitive way to get up and running with R.

```
install.packages("tidyverse")
library(tidyverse)
# installs and loads ggplot2, dplyr, tidyr, readr,
# purrr, tibble, stringr, forcats
```

and tibbles are the quickest and most intuitive way to make and read a dataset

```
dat1 <- tibble(
  age = c(24, 76, 38),
  height_in = c(70, 64, 68),
  height_cm = height_in * 2.54
)
dat1
```

```
## # A tibble: 3 x 3
##     age height_in height_cm
##   <dbl>     <dbl>     <dbl>
## 1    24        70      178.
## 2    76        64      163.
## 3    38        68      173.
```

```
dat2 <- tribble(
  ~n, ~food, ~animal,
  39, "banana", "monkey",
  21, "milk", "cat",
  18, "bone", "dog"
)
dat2
```

```
## # A tibble: 3 x 3
##       n food   animal
##   <dbl> <chr>  <chr>
## 1    39 banana monkey
## 2    21 milk   cat
## 3    18 bone   dog
```

tibbles are basically just pretty dataframes

```r
as_tibble(gss_cat)[, 1:4]
```

```
# A tibble: 21,483 x 4
    year marital         age race 
   <int> <fct>         <int> <fct>
 1  2000 Never married    26 White
 2  2000 Divorced         48 White
 3  2000 Widowed          67 White
 4  2000 Never married    39 White
 5  2000 Divorced         25 White
 6  2000 Married          25 White
 7  2000 Never married    36 White
 8  2000 Divorced         44 White
 9  2000 Married          44 White
10  2000 Married          47 White
# … with 21,473 more rows
```

```r
as.data.frame(gss_cat)[, 1:4]
```

```
   year       marital age  race
1  2000 Never married  26 White
2  2000      Divorced  48 White
3  2000       Widowed  67 White
4  2000 Never married  39 White
5  2000      Divorced  25 White
6  2000       Married  25 White
7  2000 Never married  36 White
8  2000      Divorced  44 White
9  2000       Married  44 White
10 2000       Married  47 White
11 2000       Married  53 White
12 2000       Married  52 White
13 2000       Married  52 White
14 2000       Married  51 White
15 2000      Divorced  52 White
16 2000       Married  40 Black
17 2000       Widowed  77 White
18 2000 Never married  44 White
19 2000       Married  40 White
```

National Longitudinal Survey of Youth | 1979 NLSY79

We'll use some data from the National Longitudinal Survey of Youth 1979, a cohort of American young adults aged 14-22 at enrollment in 1979. They continue to be followed to this day, and there is a wealth of publicly available data online. I've downloaded the answers to a survey question about whether respondents wear glasses, a scale about their eyesight with glasses, whether they are black or white/hispanic, their sex, their family's income in 1979, and their age at the birth of their first child.

Read in data

```
nlsy <- read_csv("nlsy_cc.csv")
nlsy
```

```
## # A tibble: 1,205 x 14
##   H0012400 H0012500 H0022300 H0022500 R0000100 R0009100 R0173600 R0214700 R0214800 R0216400
##      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1        0        1        5        7        3        3        5        3        2        1
## 2        1        2        6        7        6        1        1        3        1        1
## # … with 1,203 more rows, and 4 more variables: R0217900 <dbl>, R0402800 <dbl>,
## #   R7090700 <dbl>, T4120500 <dbl>
```

Ugh...

```
colnames(nlsy)
```

```
##  [1] "H0012400" "H0012500" "H0022300" "H0022500" "R0000100" "R0009100" "R0173600" "R0214700"
##  [9] "R0214800" "R0216400" "R0217900" "R0402800" "R7090700" "T4120500"
```

```
colnames(nlsy) <- c("glasses", "eyesight", "sleep_wkdy", "sleep_wknd",
                    "id", "nsibs", "samp", "race_eth", "sex", "region",
                    "income", "res_1980", "res_2002", "age_bir")
```

Explore your data

```
glimpse(nlsy)
```

```
## Observations: 1,205
## Variables: 14
## $ glasses     <dbl> 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,…
## $ eyesight    <dbl> 1, 2, 2, 3, 3, 2, 1, 1, 2, 1, 3, 5, 1, 1, 1, 1, 3, 2, 3, 3, 4, 2, 2, 5,…
## $ sleep_wkdy  <dbl> 5, 6, 7, 6, 10, 7, 8, 8, 7, 8, 8, 7, 7, 7, 8, 7, 7, 8, 8, 8, 7, 6, 8, 7…
## $ sleep_wknd  <dbl> 7, 7, 9, 7, 10, 8, 8, 8, 8, 8, 8, 7, 8, 7, 8, 7, 4, 8, 8, 9, 7, 10, 8, …
## $ id          <dbl> 3, 6, 8, 16, 18, 20, 27, 49, 57, 67, 86, 96, 97, 98, 117, 137, 172, 179…
## $ nsibs       <dbl> 3, 1, 7, 3, 2, 2, 1, 6, 1, 1, 7, 2, 7, 2, 2, 4, 9, 2, 2, 2, 4, 2, 4, 4,…
## $ samp        <dbl> 5, 1, 6, 5, 1, 5, 5, 5, 5, 1, 7, 6, 5, 6, 1, 5, 6, 5, 5, 5, 8, 1, 7, 5,…
## $ race_eth    <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 2, 3,…
## $ sex         <dbl> 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2,…
## $ region      <dbl> 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3,…
## $ income      <dbl> 22390, 35000, 7227, 48000, 4510, 50000, 20000, 23900, 23289, 35000, 168…
## $ res_1980    <dbl> 11, 3, 11, 11, 11, 3, 11, 11, 11, 3, 11, 11, 11, 11, 6, 3, 11, 11, 3, 1…
## $ res_2002    <dbl> 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 19, 11, 11, 11, 11, 11, 11, 11,…
## $ age_bir     <dbl> 19, 30, 17, 31, 19, 30, 27, 24, 21, 36, 17, 19, 29, 30, 26, 26, 35, 22,…
```

Explore your data

```
summary(nlsy)
```

```
##     glasses          eyesight       sleep_wkdy       sleep_wknd             id
##  Min.   :0.0000   Min.   :1.00   Min.   : 0.000   Min.   : 0.000   Min.   :    3
##  1st Qu.:0.0000   1st Qu.:1.00   1st Qu.: 6.000   1st Qu.: 6.000   1st Qu.: 2317
##  Median :1.0000   Median :2.00   Median : 7.000   Median : 7.000   Median : 4744
##  Mean   :0.5178   Mean   :1.99   Mean   : 6.643   Mean   : 7.267   Mean   : 5229
##  3rd Qu.:1.0000   3rd Qu.:3.00   3rd Qu.: 8.000   3rd Qu.: 8.000   3rd Qu.: 7937
##  Max.   :1.0000   Max.   :5.00   Max.   :13.000   Max.   :14.000   Max.   :12667
##      nsibs             samp           race_eth          sex            region
##  Min.   : 0.000   Min.   : 1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.: 2.000   1st Qu.: 4.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:2.000
##  Median : 3.000   Median : 5.000   Median :3.000   Median :2.000   Median :3.000
##  Mean   : 3.937   Mean   : 7.002   Mean   :2.395   Mean   :1.584   Mean   :2.593
##  3rd Qu.: 5.000   3rd Qu.:11.000   3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:3.000
##  Max.   :16.000   Max.   :20.000   Max.   :3.000   Max.   :2.000   Max.   :4.000
##      income          res_1980        res_2002        age_bir
##  Min.   :    0   Min.   : 1.00   Min.   : 5.00   Min.   :13.00
##  1st Qu.: 6000   1st Qu.:11.00   1st Qu.:11.00   1st Qu.:19.00
##  Median :11155   Median :11.00   Median :11.00   Median :22.00
##  Mean   :15289   Mean   : 9.14   Mean   :11.05   Mean   :23.45
##  3rd Qu.:20000   3rd Qu.:11.00   3rd Qu.:11.00   3rd Qu.:27.00
```

Explore your data

```
summary(nlsy$glasses)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  0.0000  0.0000  1.0000  0.5178  1.0000  1.0000
```

```
mean(nlsy$age_bir)
```

```
## [1] 23.44813
```

```
?cor
```

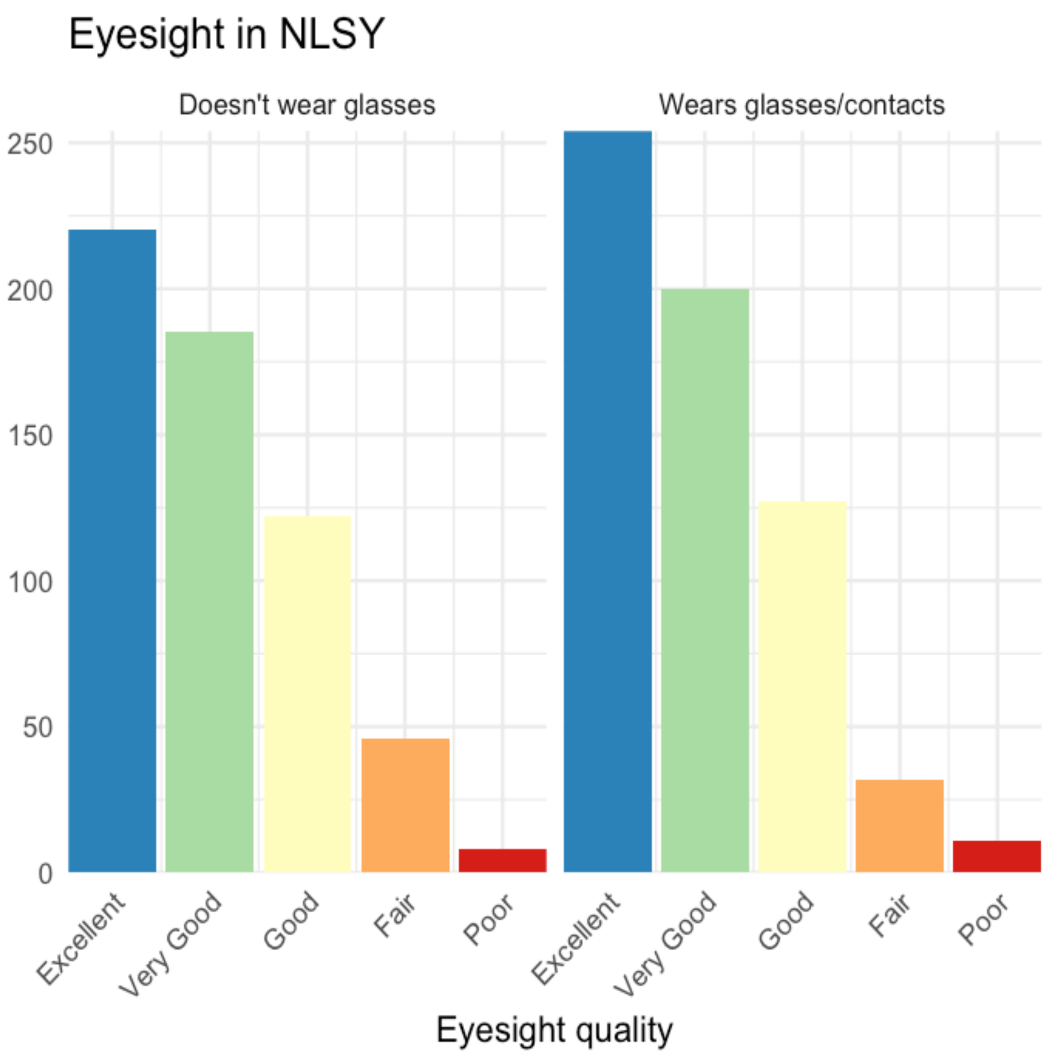
Get help!

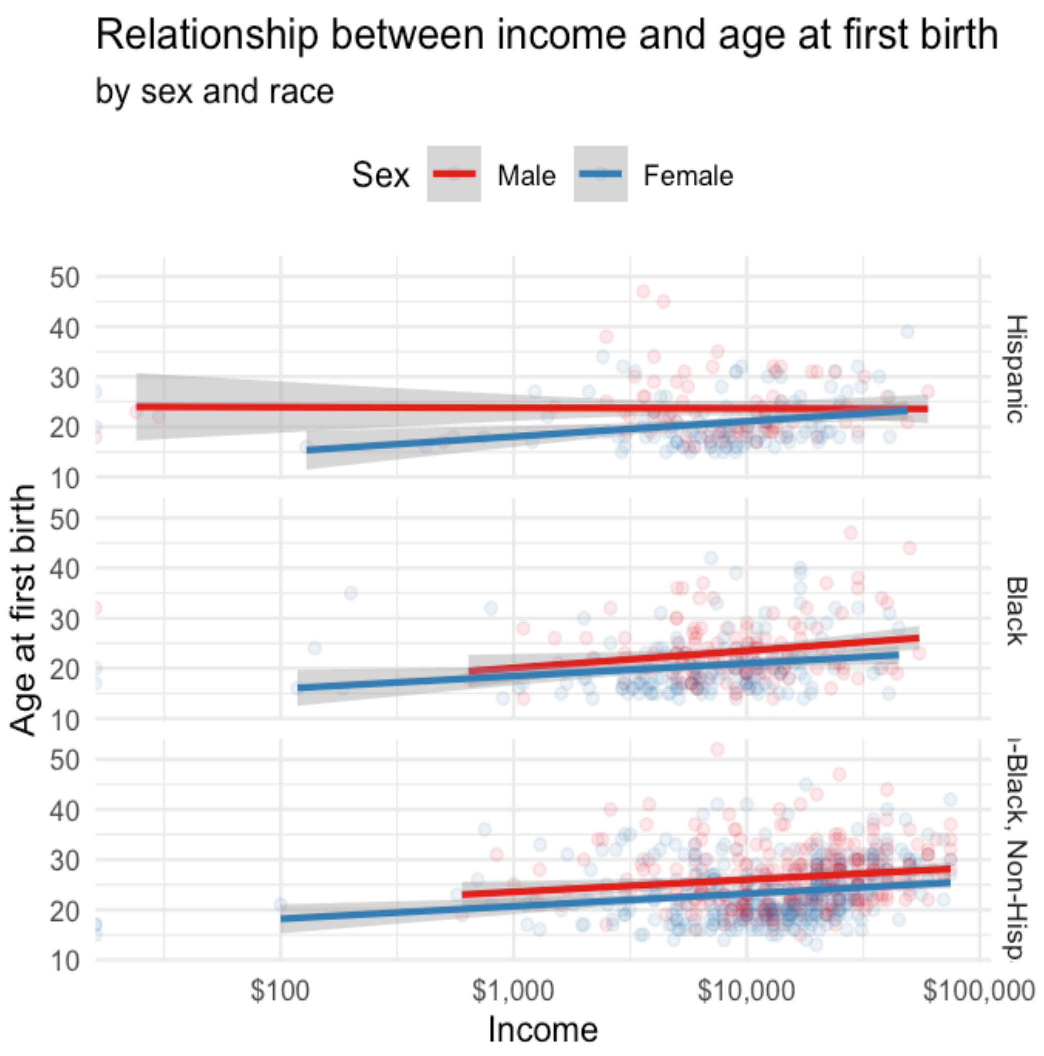
- `help(cor)`
- https://www.rdocumentation.org
- https://rdrr.io
- https://www.r-project.org/help.html
- SO. MUCH. MORE.

Exercises 2

1. How many people are in the NLSY? How many variables are in this dataset? What are two ways you can answer these questions?
2. Can you find an R function(s) we haven't discussed that answers q2? (Hint: Google)
3. What's the Spearman correlation between hours of sleep on weekends and weekdays in this data?
4. I've also provided you with the same dataset as an Excel document, but it's not on the first sheet, and there's an annoying header. Load the `readxl` package (you already installed with with `tidyverse`, but it doesn't load automatically). Figure out how to read in the data. This may help: https://readxl.tidyverse.org.

#goals

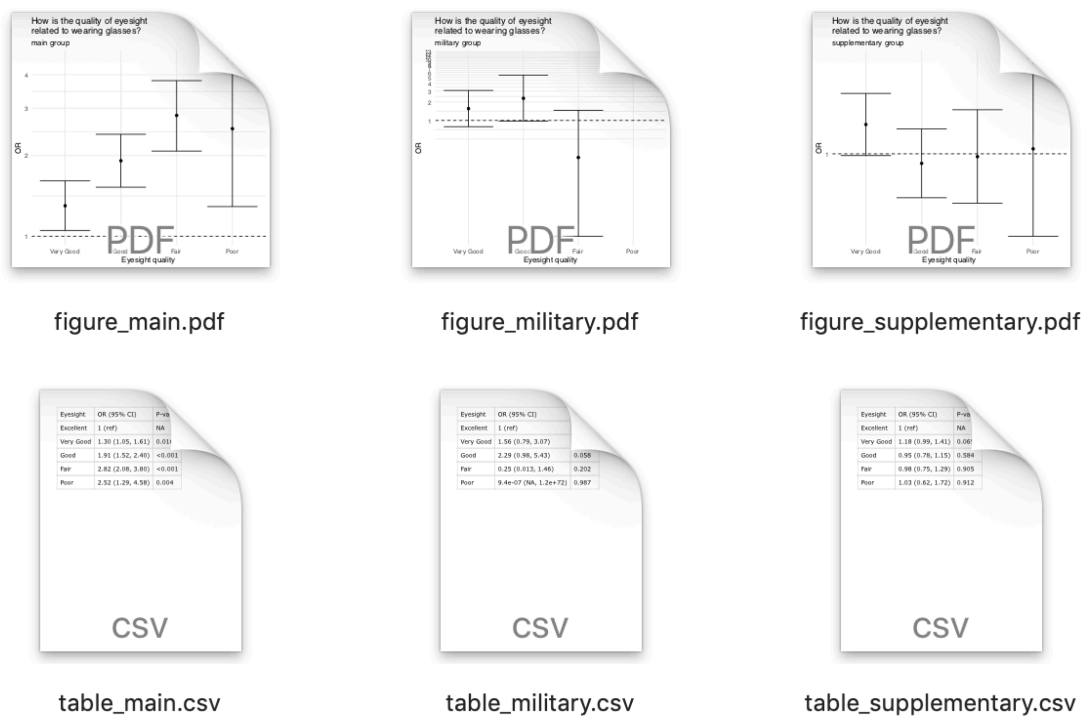

Eyesight in NLSY
Doesn't wear glasses
Wears glasses/contacts
250
200
150
100
50
0
Excellent
Very Good
Good
Fair
Poor
Excellent
Very Good
Good
Fair
Poor
Eyesight quality


Relationship between income and age at first birth
by sex and race
Sex
Male
Female
Age at first birth
50
40
30
20
10
Hispanic
Black
-Black, Non-Hisp
$100
$1,000
$10,000
$100,000
Income

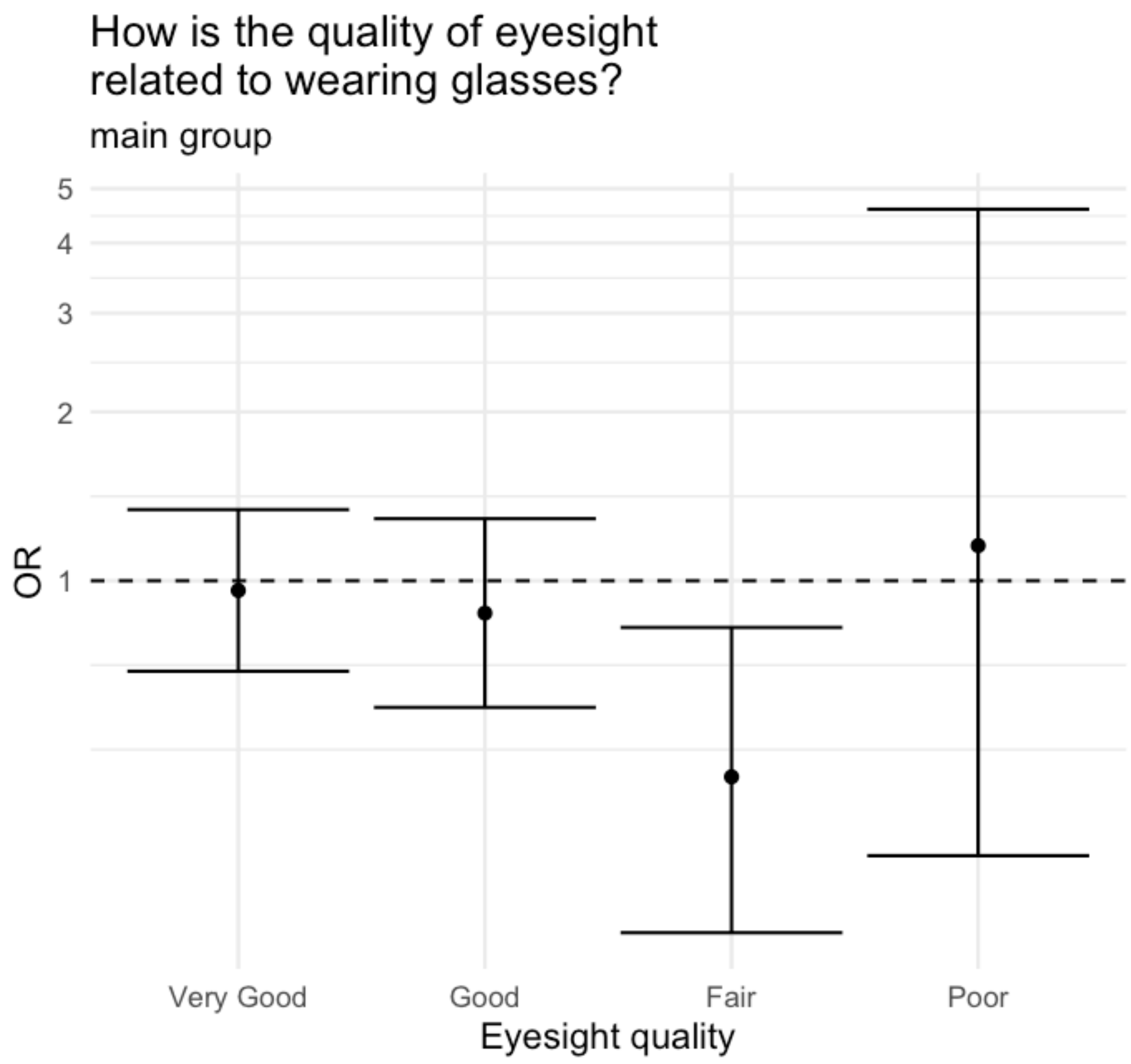
#goals

```r
run_analysis(group = "main")
run_analysis(group = "supplementary")
run_analysis(group = "military")
```

Ta da!



#goals



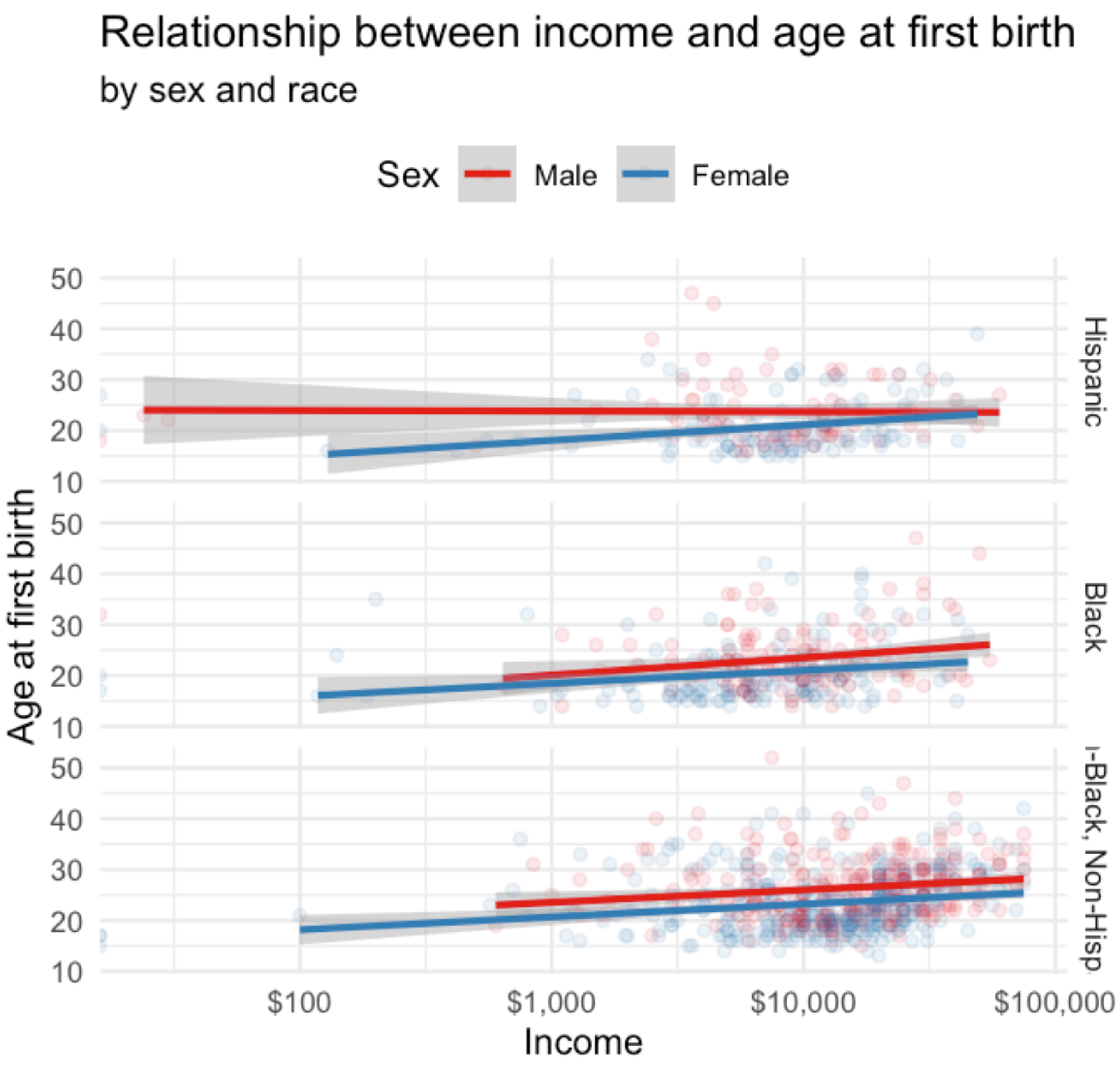
Eyesight	OR (95% CI)	P-value
Excellent	1 (ref)	NA
Very Good	0.96 (0.69, 1.34)	0.814
Good	0.88 (0.59, 1.29)	0.501
Fair	0.45 (0.24, 0.83)	0.011
Poor	1.16 (0.32, 4.59)	0.826

Basic structure of a ggplot

```
ggplot(data = {data}) +
      <geom>(aes(x = {xvar}, y = {yvar}, <characteristic> = {othvar}, ...),
             <characteristic> = "value", ...) +
      ...
```

- `{data}`: must be a dataframe (or tibble!)
- `{xvar}` and `{yvar}` are the column names (unquoted) of the variables on the x- and y-axes
- `{othvar}` is some other unquoted variable name that defines a grouping or other characteristic you want to map to an aesthetic
- `<geom>`: the geometric feature you want to use; e.g., point (scatterplot), line, histogram, bar, etc.
- `<characteristic>`: you can map `{othvar}` or a fixed `"value"` to any of a number of aesthetic features of the figure; e.g., color, shape, size, linetype, etc.
- `"value"`: a fixed value that defines some characteristic of the figure; e.g., "red", 10, "dashed"
- ... : there are numerous other options to discover!

```r
ggplot(data = nlsy, aes(x = income,
    y = age_bir, col = factor(sex))
) +
  geom_point(alpha = 0.1) +
  scale_color_brewer(palette = "Set1",
    name = "Sex",
    labels = c("Male", "Female")) +
  scale_x_log10(labels =
                scales::dollar) +
  geom_smooth(aes(
    group = factor(sex)),
    method = "lm") +
  facet_grid(rows = vars(race_eth),
    labeller = labeller(race_eth = c(
    "1" = "Hispanic",
    "2" = "Black",
    "3" = "Non-Black, Non-Hispanic"))) +
  theme_minimal() +
  theme(legend.position = "top") +
  labs(title = "Relationship between income and
    subtitle = "by sex and race",
    x = "Income",
    y = "Age at first birth")
```



Basic example

```
ggplot(data = {data}) +
      <geom>(aes(x = {xvar}, y = {yvar}, <characteristic> = {othvar}, ...),
             <characteristic> = "value", ...) +
      ...
```

Basic example

```
ggplot(data = nlsy) +
      <geom>(aes(x = {xvar}, y = {yvar}, <characteristic> = {othvar}, ...),
             <characteristic> = "value", ...) +
      ...
```

The `data =` argument must be a dataframe (or tibble)

Basic example

```
ggplot(data = nlsy) +
      geom_point(aes(x = {xvar}, y = {yvar}, <characteristic> = {othvar}, ...),
             <characteristic> = "value", ...) +
      ...
```

geom_point() gives us a scatterplot

Other helpful "geoms" include `geom_line()`, `geom_bar()`, `geom_histogram()`, `geom_boxplot()`

- A helpful reference can be found here: http://sape.inf.usi.ch/quick-reference/ggplot2/geom

Basic example

```
ggplot(data = nlsy) +
      geom_point(aes(x = income, y = age_bir, <characteristic> = {othvar}, ...),
             <characteristic> = "value", ...) +
      ...
```

Notice the variable names are not in quotation marks

`geom_point()` requires an x = and a y = variable

Other geoms require other arguments

- For example, `geom_histogram()` only requires an `x =` variable

Basic example

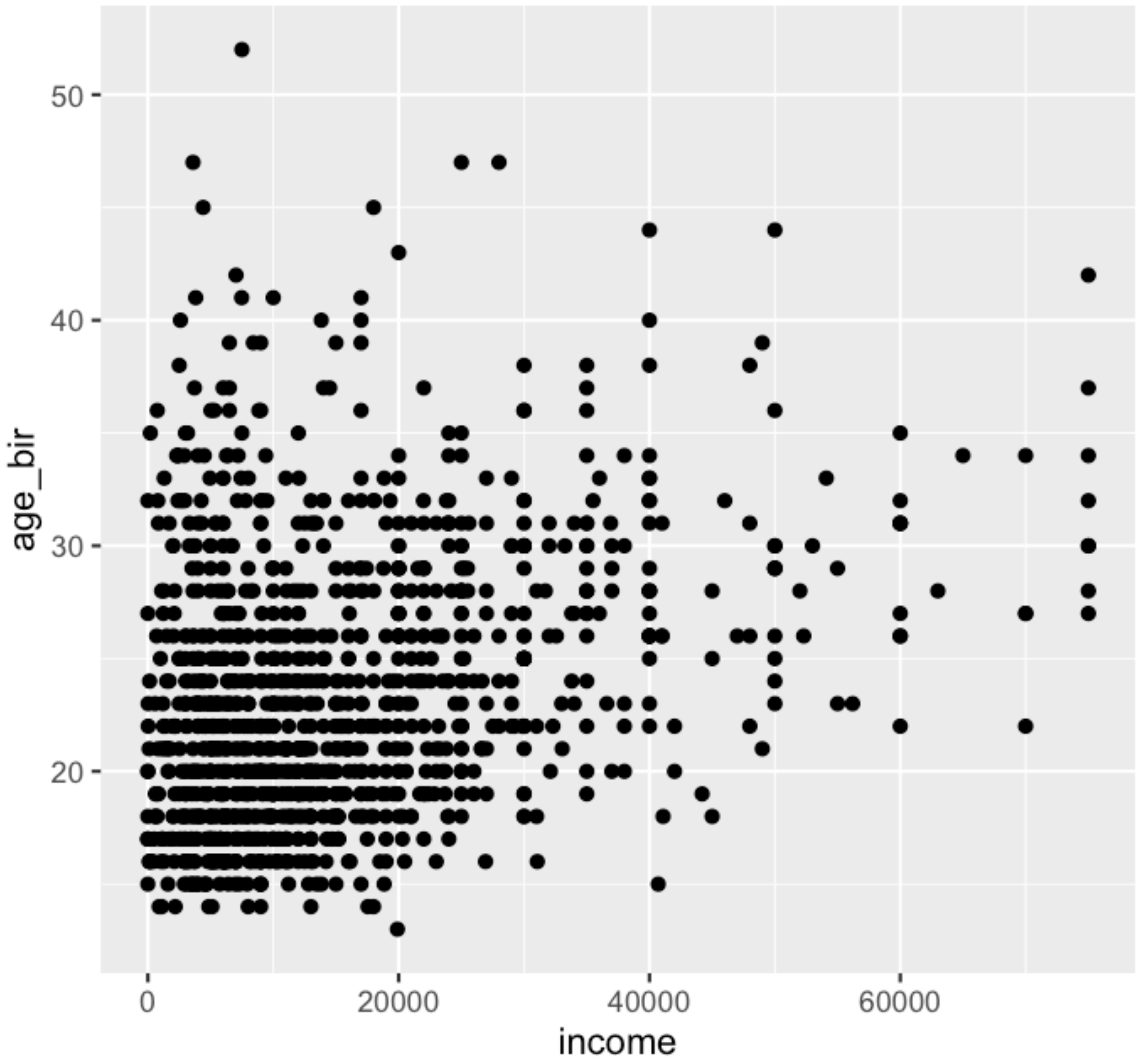
```
ggplot(data = nlsy, aes(x = income, y = age_bir, <characteristic> = {othvar}, ...)) +
      geom_point(<characteristic> = "value", ...) +
      ...
```

We could also put the aesthetics (the variables that are being mapped to the plot) in the initial `ggplot()` function

This will be helpful when we want multiple geoms (say, points and a line)

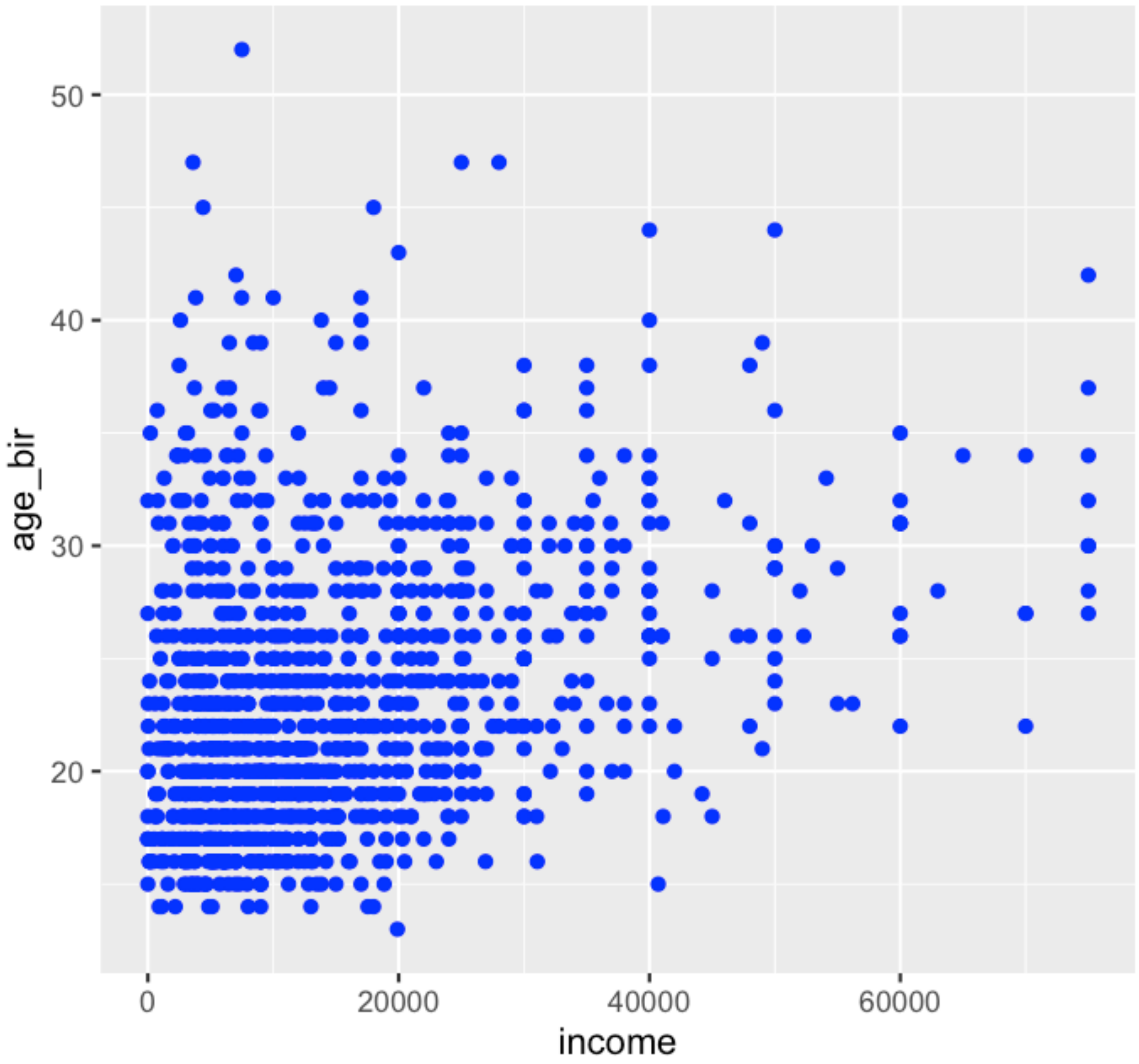
```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir))
```

What if we want to change the color of the points?



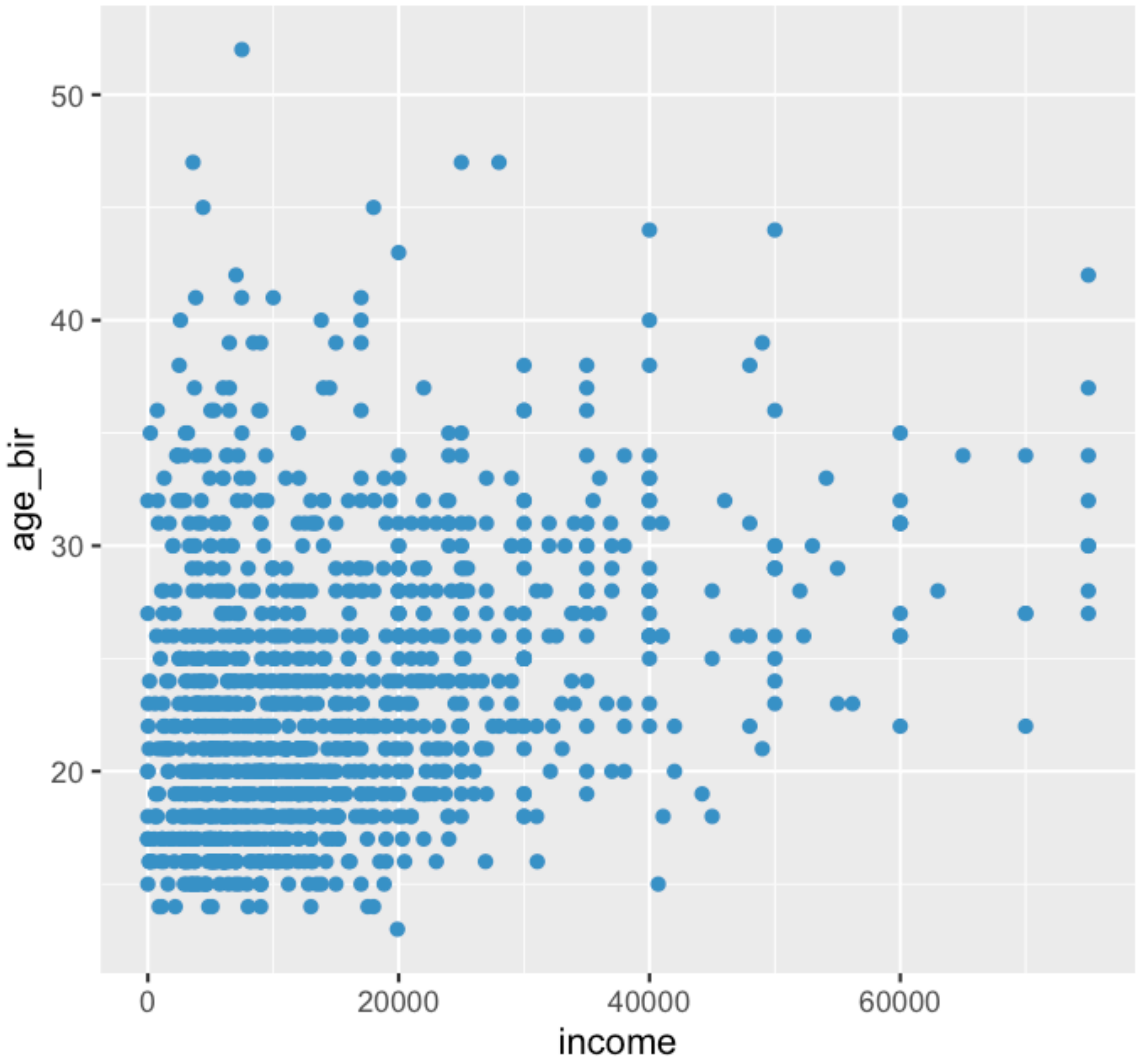
```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir),
               color = "blue")
```

When we put **`color =`** *outside* the **`aes()`**, it means we're giving it a specific color value that applies to all the points



```r
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir),
               color = "#3d93c8")
```

One of my favorite color resources:
https://www.color-hex.com

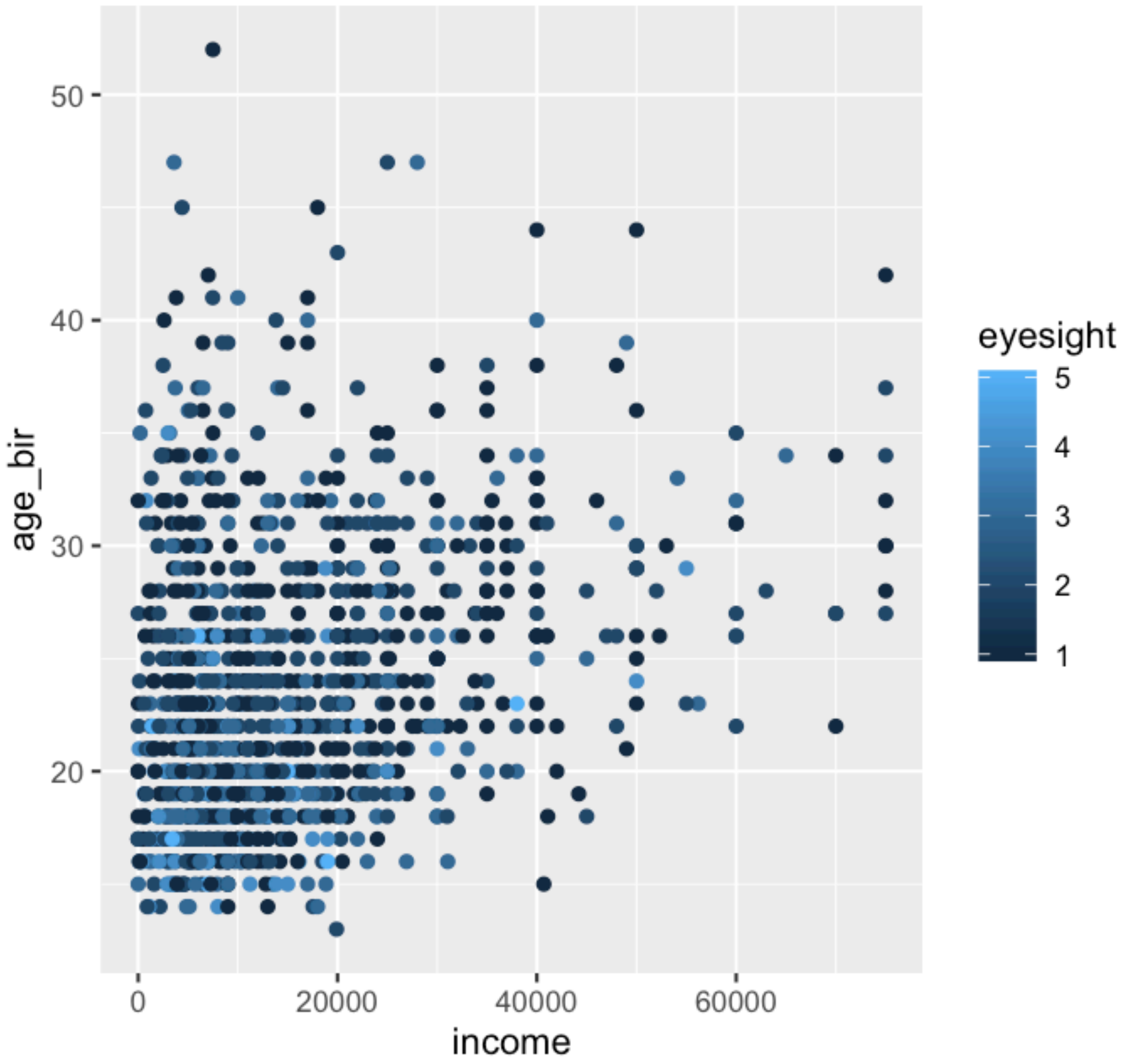


```r
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
                  color = eyesight))
```

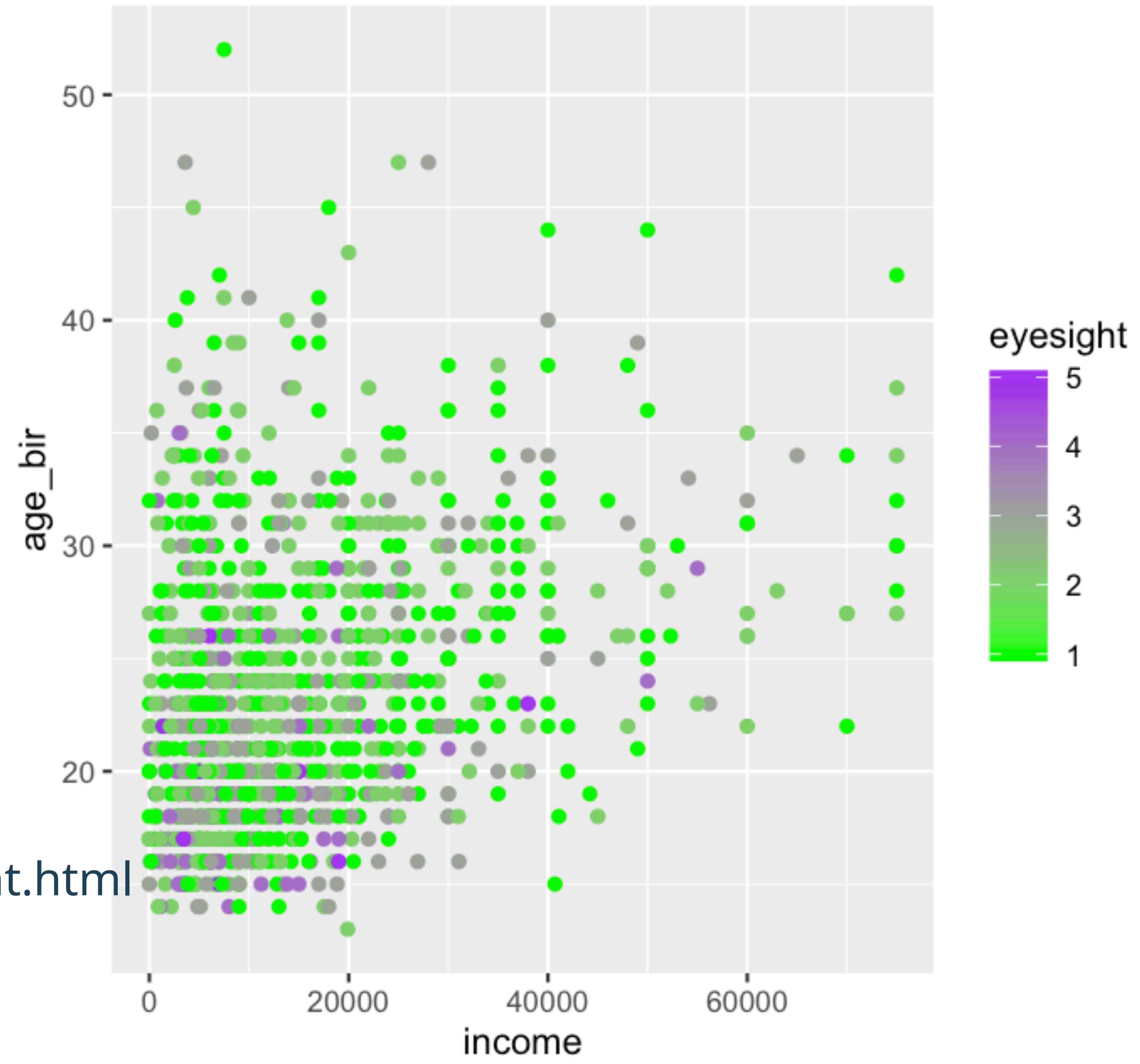
When we put **color** = *inside* the **aes()** -- with no quotation marks -- it means we're telling it how it should assign colors

Here we're plotting the values according to eyesight, where 1 is excellent and 5 is poor.

- But they're kind of hard to distinguish!



```r
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
               color = eyesight)) +
scale_color_gradient(low = "green",
                     high = "purple")
```



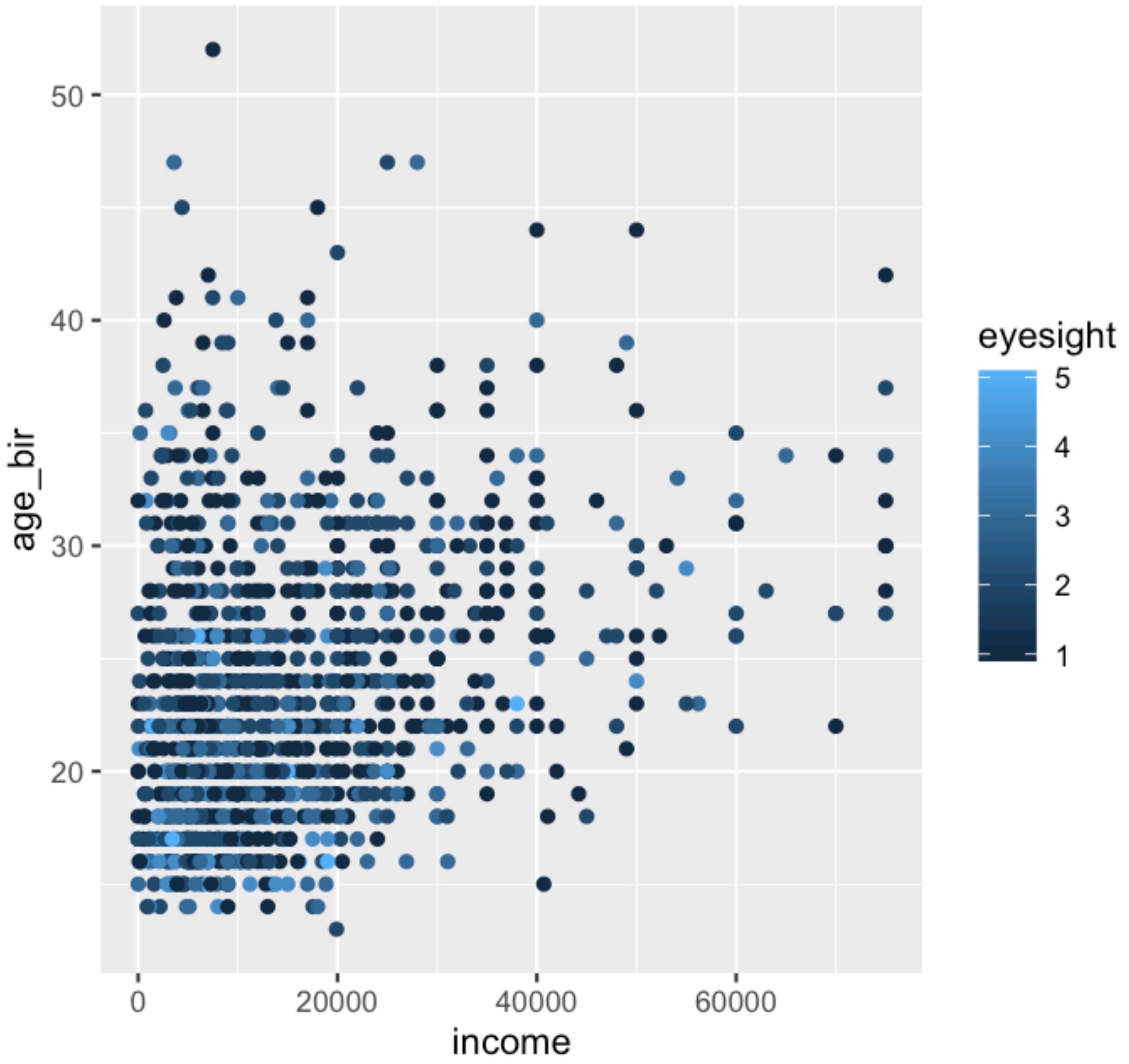
We can map the values of `eyesight` to a different continuous scale using `scale_color_gradient()`

You can read lots more about this function here, so you don't have to have such ugly color scales!

https://ggplot2.tidyverse.org/reference/scale_gradient.html

```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
            color = eyesight))
```

Returning to the nice blues, we think: But wait! The variable **eyesight** isn't really continuous: it has 5 discrete values

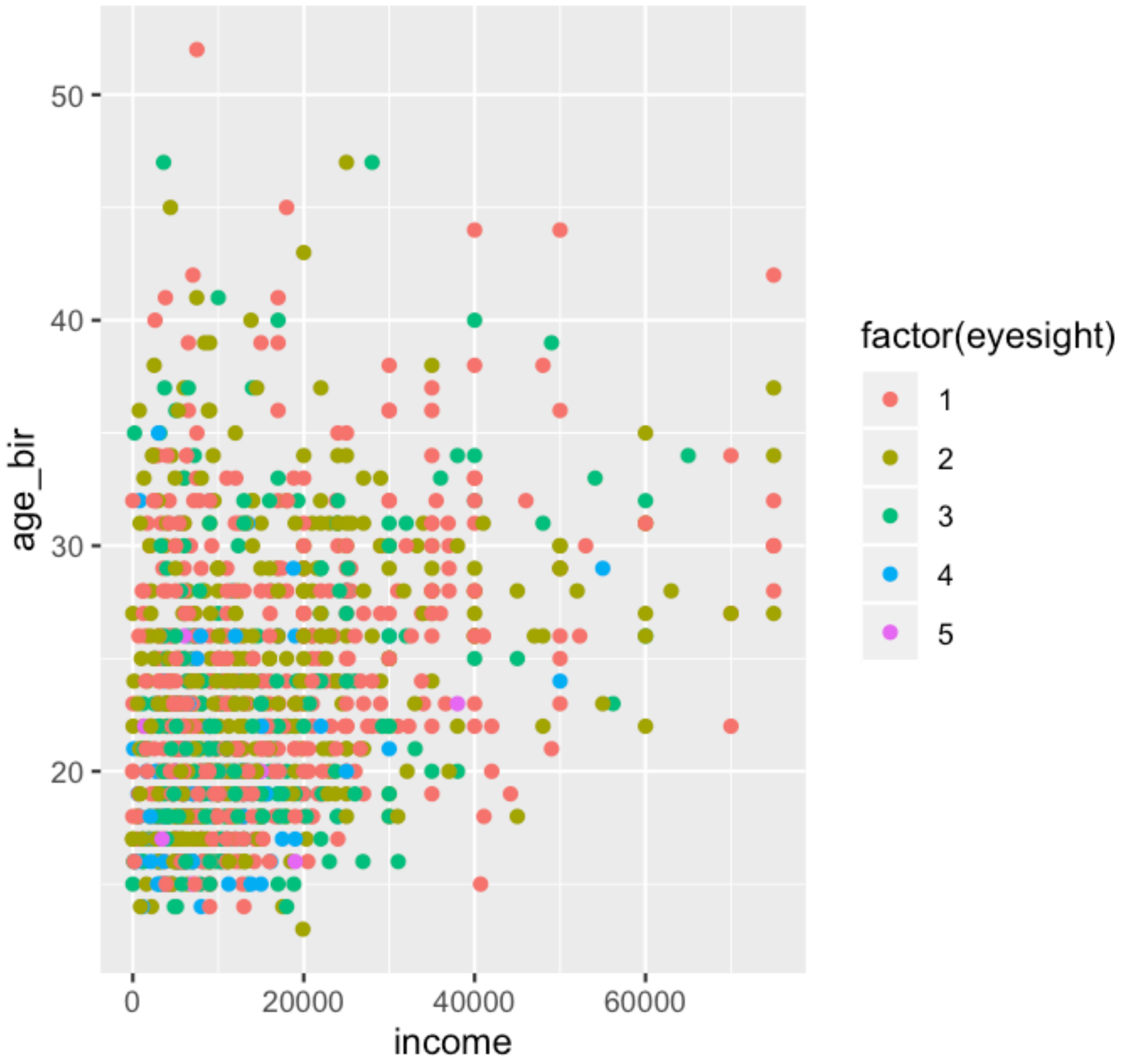


```r
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
           color = factor(eyesight)))
```

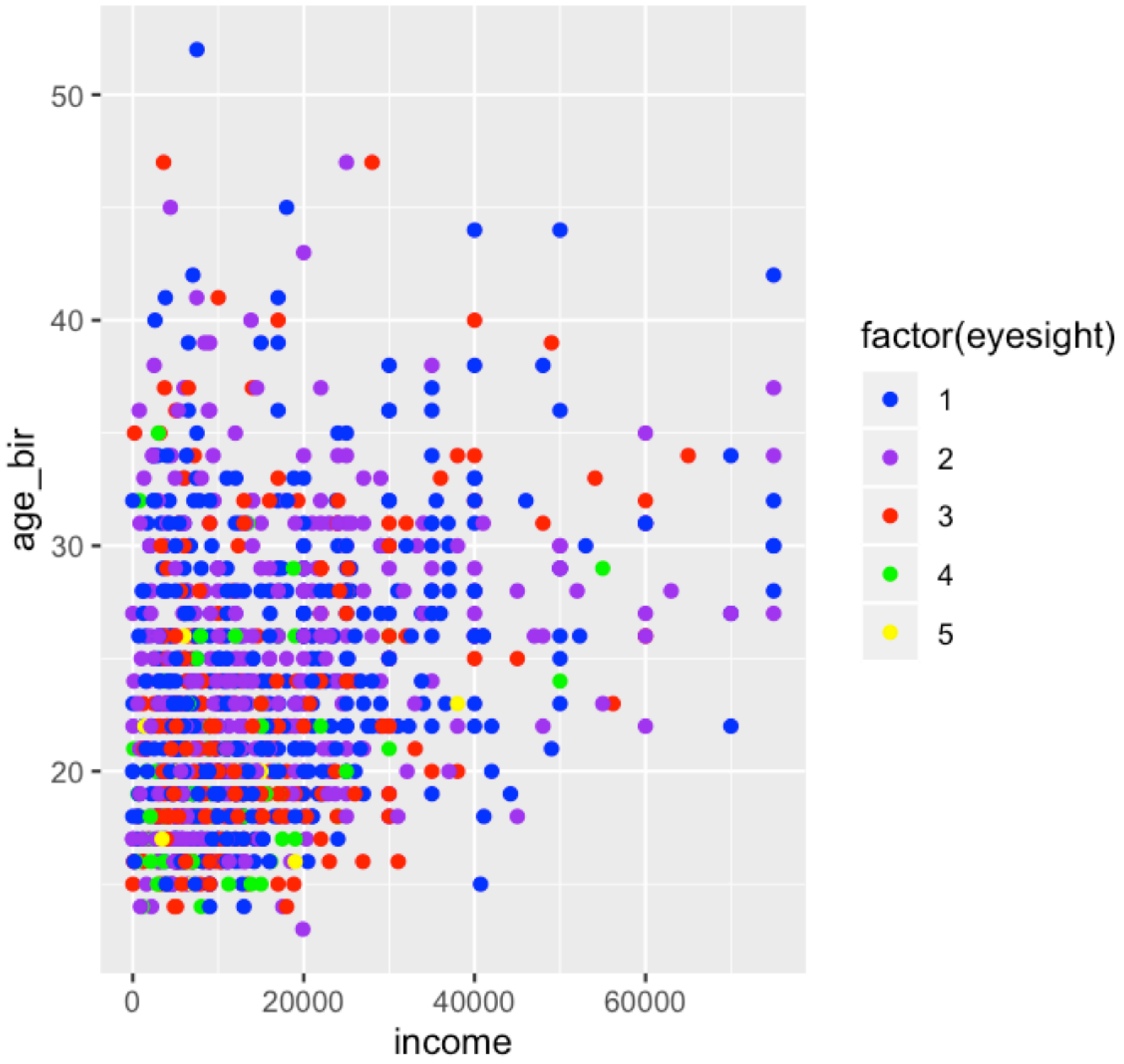
Returning to the nice blues, we think: But wait! The variable `eyesight` isn't really continuous: it has 4 discrete values

We can make R treat it as a "factor", or categorical variable, with the `factor()` function

- We'll see lots more on factors later!



```r
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
           color = factor(eyesight))) +
scale_color_manual(
    values = c("blue", "purple", "red",
               "green", "yellow"))
```



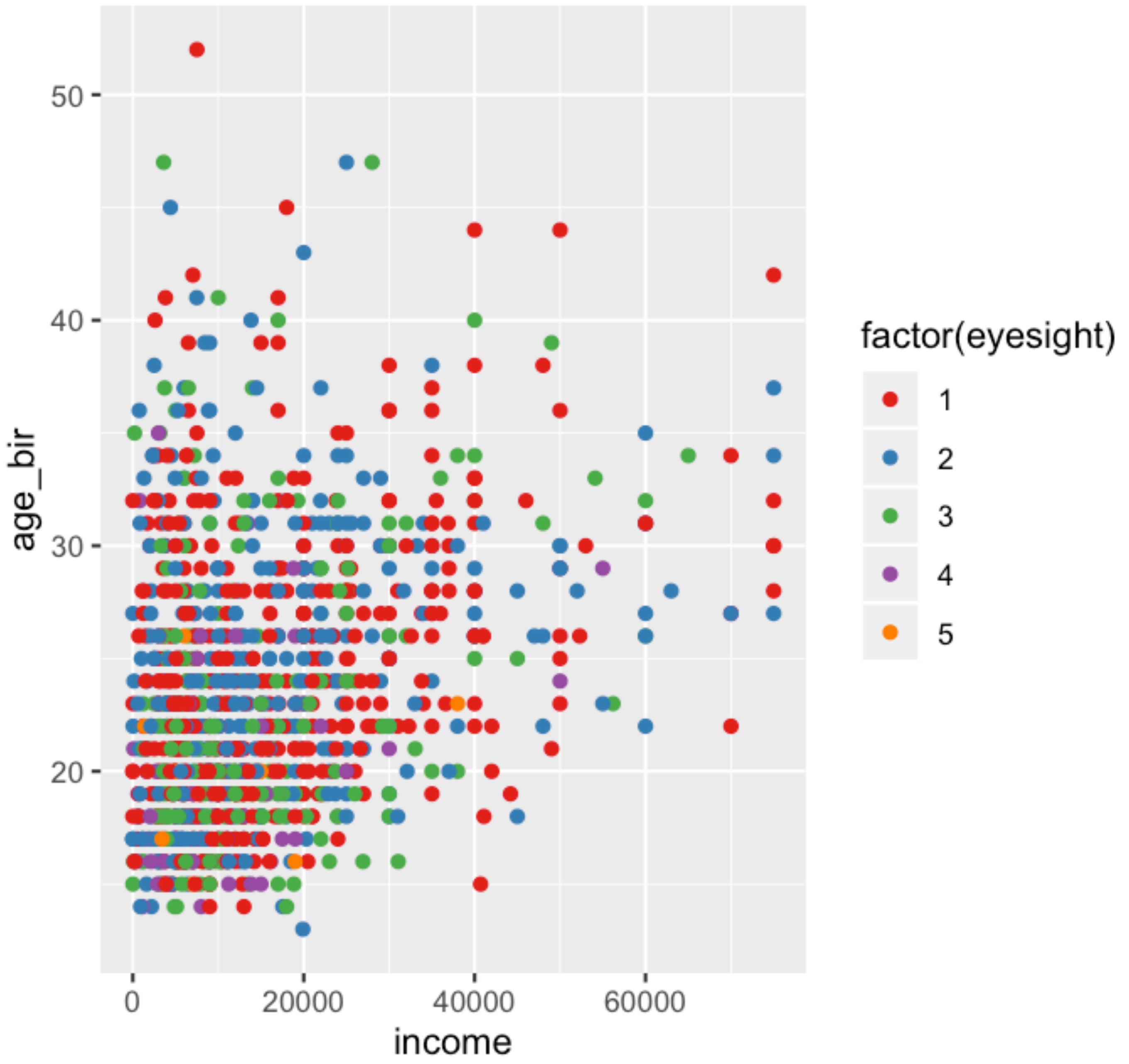
Now if we want to change the color scheme, we have to use a different function

```r
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
               color = factor(eyesight))) +
scale_color_brewer(palette = "Set1")
```

There are tons of different options in R for color palettes

You can play around with those in the `RColorBrewer` package here: http://colorbrewer2.org

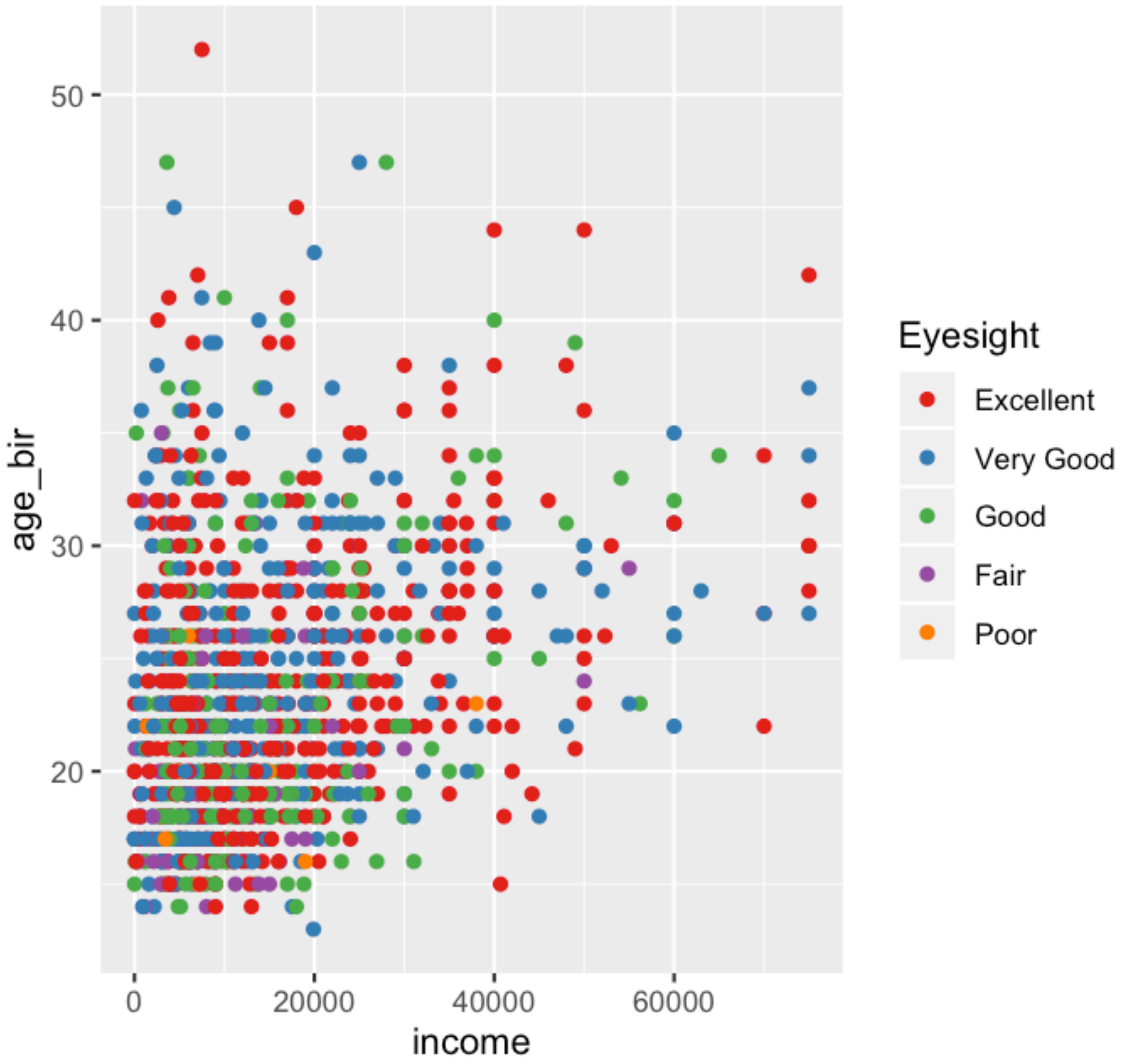
- (You can access the scales in that package with `scale_color_brewer()`, or see them all after installing the package with `RColorBrewer::display.brewer.all()`)



```r
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
             color = factor(eyesight))) +
scale_color_brewer(palette = "Set1",
             name = "Eyesight",
             labels = c("Excellent",
                        "Very Good",
                        "Good",
                        "Fair",
                        "Poor"))
```

Each of the **`scale_color_()`** functions has a lot of the same arguments

Make sure if you are labelling a factor variable in a plot like this that you get the names right!



Exercises 3

1. Using the NLSY data, make a scatter plot of the relationship between hours of sleep on weekends and weekdays. Color it according to region (where 1 = northeast, 2 = north central, 3 = south, and 4 = west).
2. Replace `geom_point()` with `geom_jitter()`. What does this do? Why might this be a good choice for this graph? Play with the `width =` and `height =` options. This site may help: https://ggplot2.tidyverse.org/reference/geom_jitter.html
3. Use the `shape =` argument to map the sex variable to different shapes. Change the shapes to squares and diamonds. (Hint: how did we manually change colors to certain values? This might help: https://ggplot2.tidyverse.org/articles/ggplot2-specs.html)

Facets

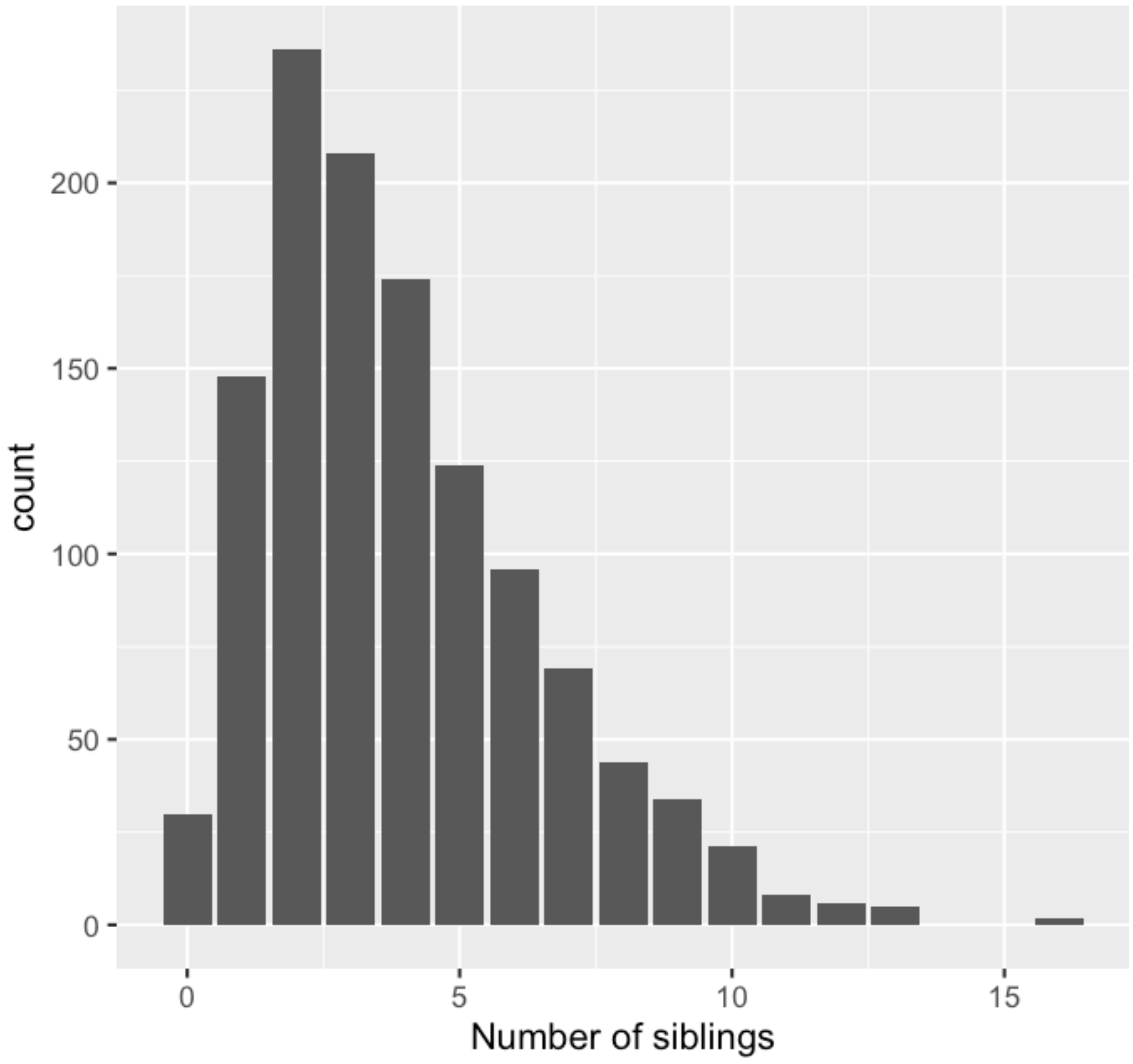
One of the most useful features of `ggplot2` is the ability to "facet" a graph by splitting it up according to the values of some variable.

You might use this to show results for a lot of outcomes or exposures at once, for example, or see how some relationship differs by something like age or geographic region

```r
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings")
```

We'll introduce bar graphs at the same time!

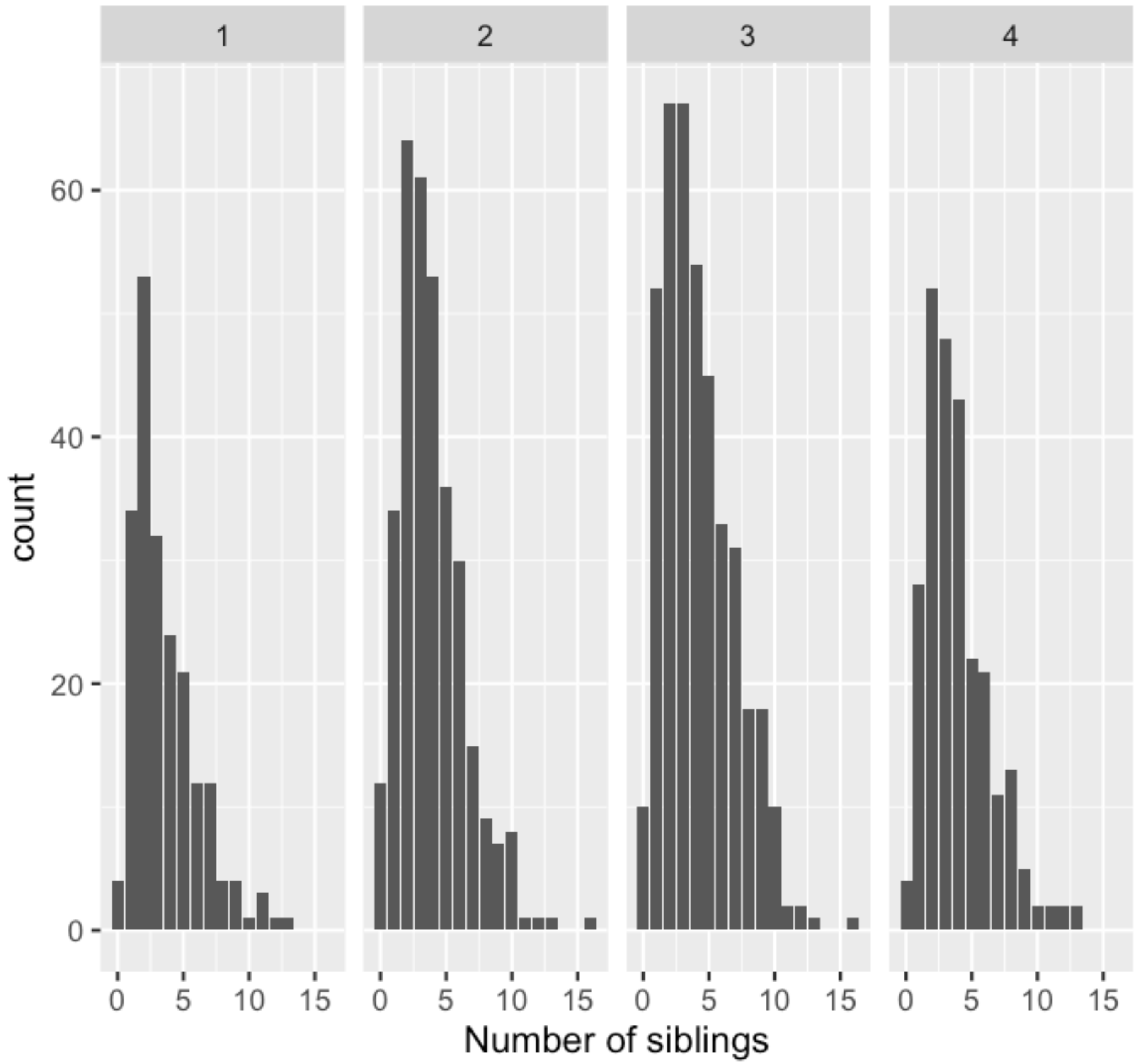
Notice how we only need an `x =` argument - the y-axis is automatically the count with this geom.



```r
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_grid(cols = vars(region))
```

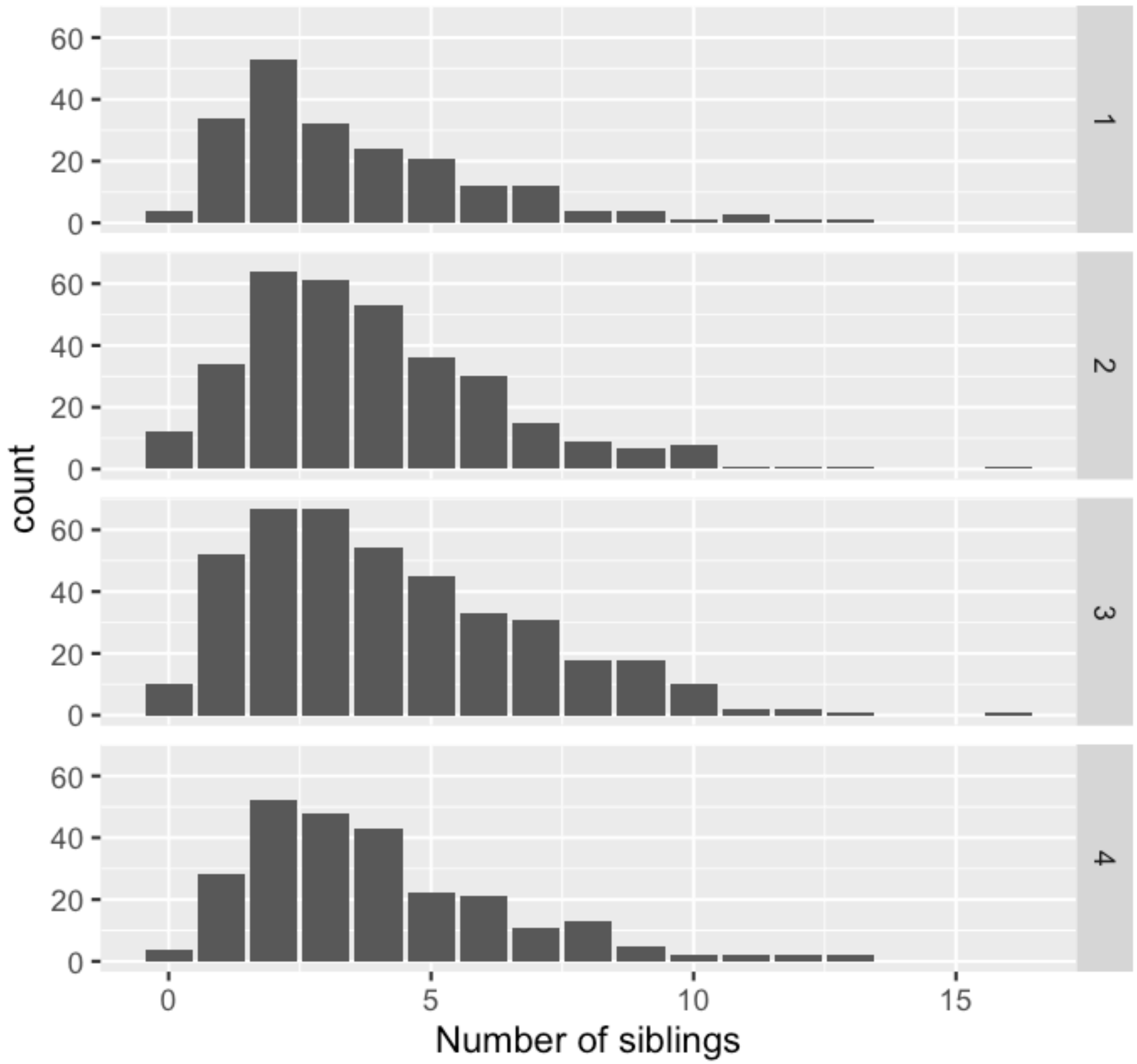
The **facet_grid()** function splits up the data according to a variable(s)

Here we've split it by region into columns



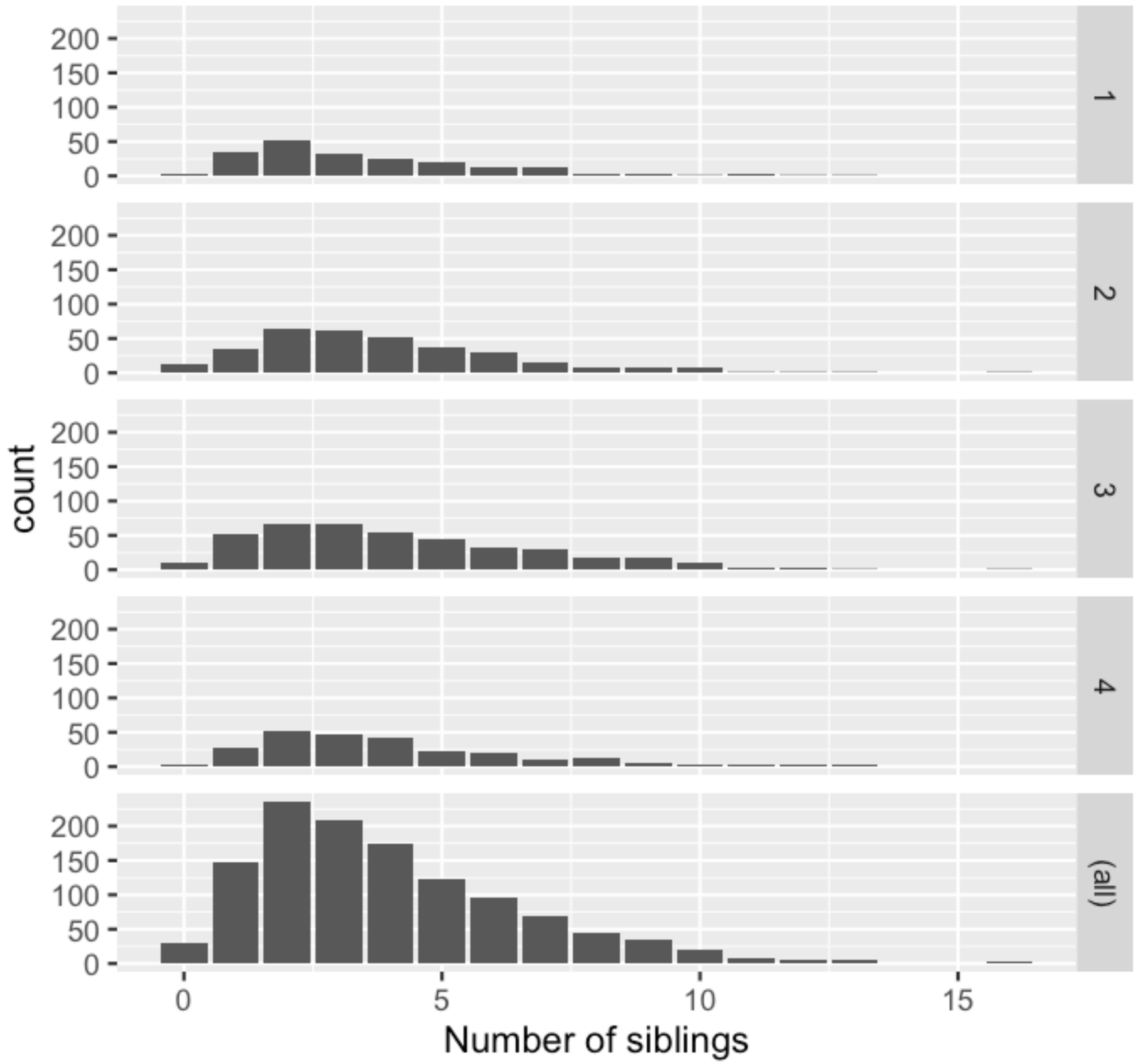
```
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_grid(rows = vars(region))
```

Since this is hard to read, we'll probably want to split by rows instead



```r
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_grid(rows = vars(region),
             margins = TRUE)
```

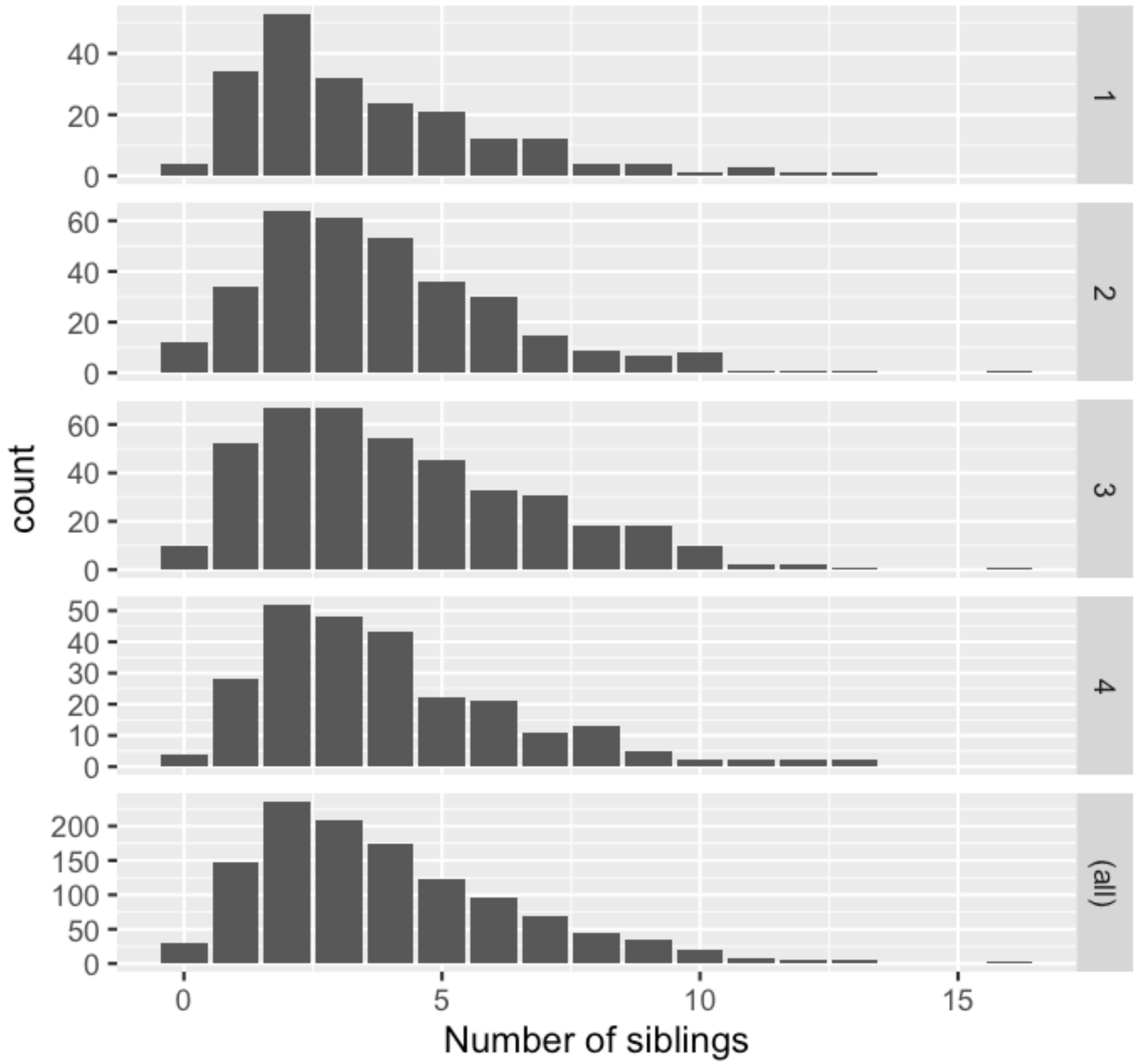
We can also add a row for all of the data together



```r
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_grid(rows = vars(region),
             margins = TRUE,
             scales = "free_y")
```

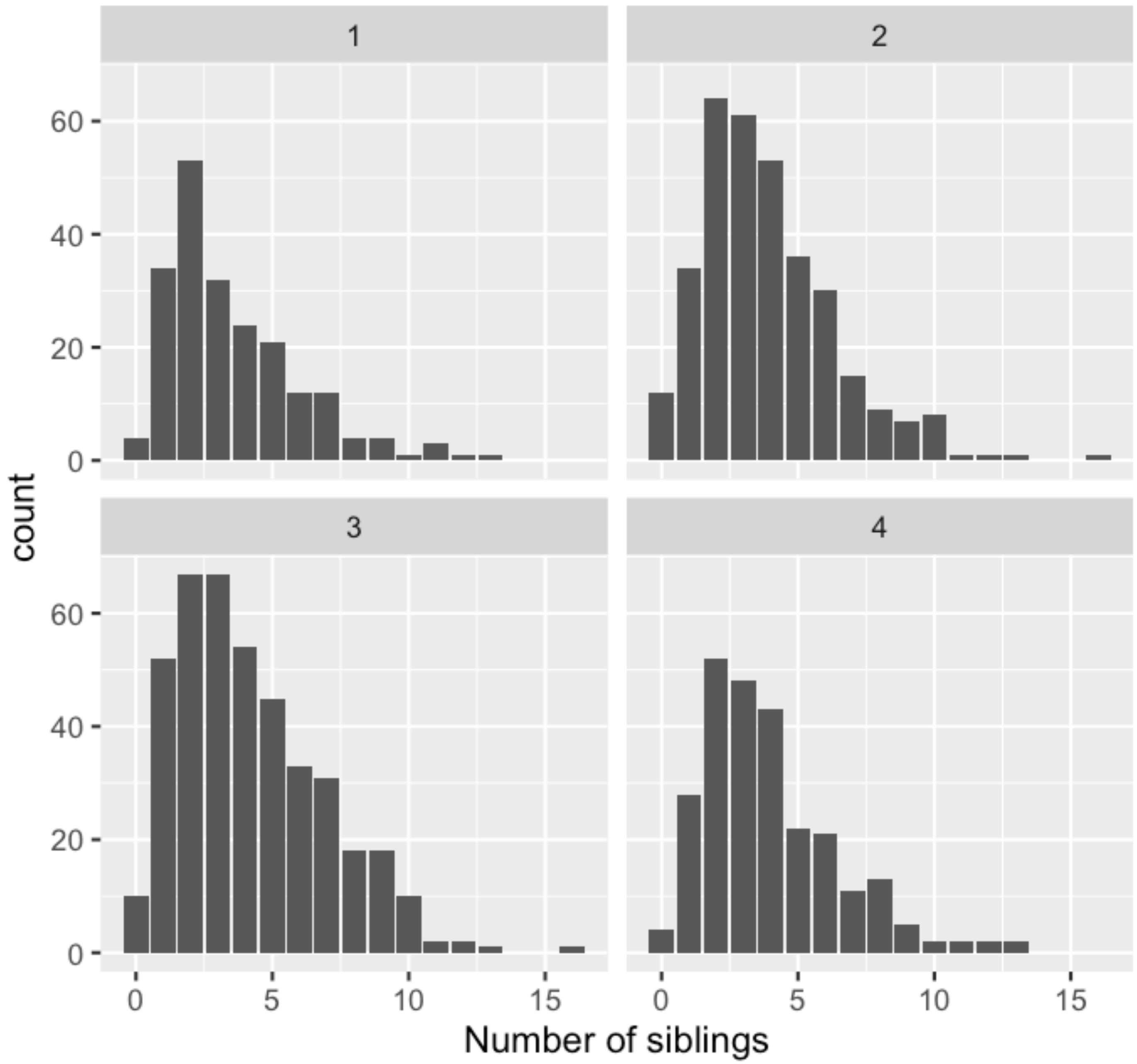
This squishes the other rows though! We can allow them all to have their own axis limits with the **`scales =`** argument

Other options are "free_x" if we want to allow the x-axis scale to vary, or just "free" to combine both.



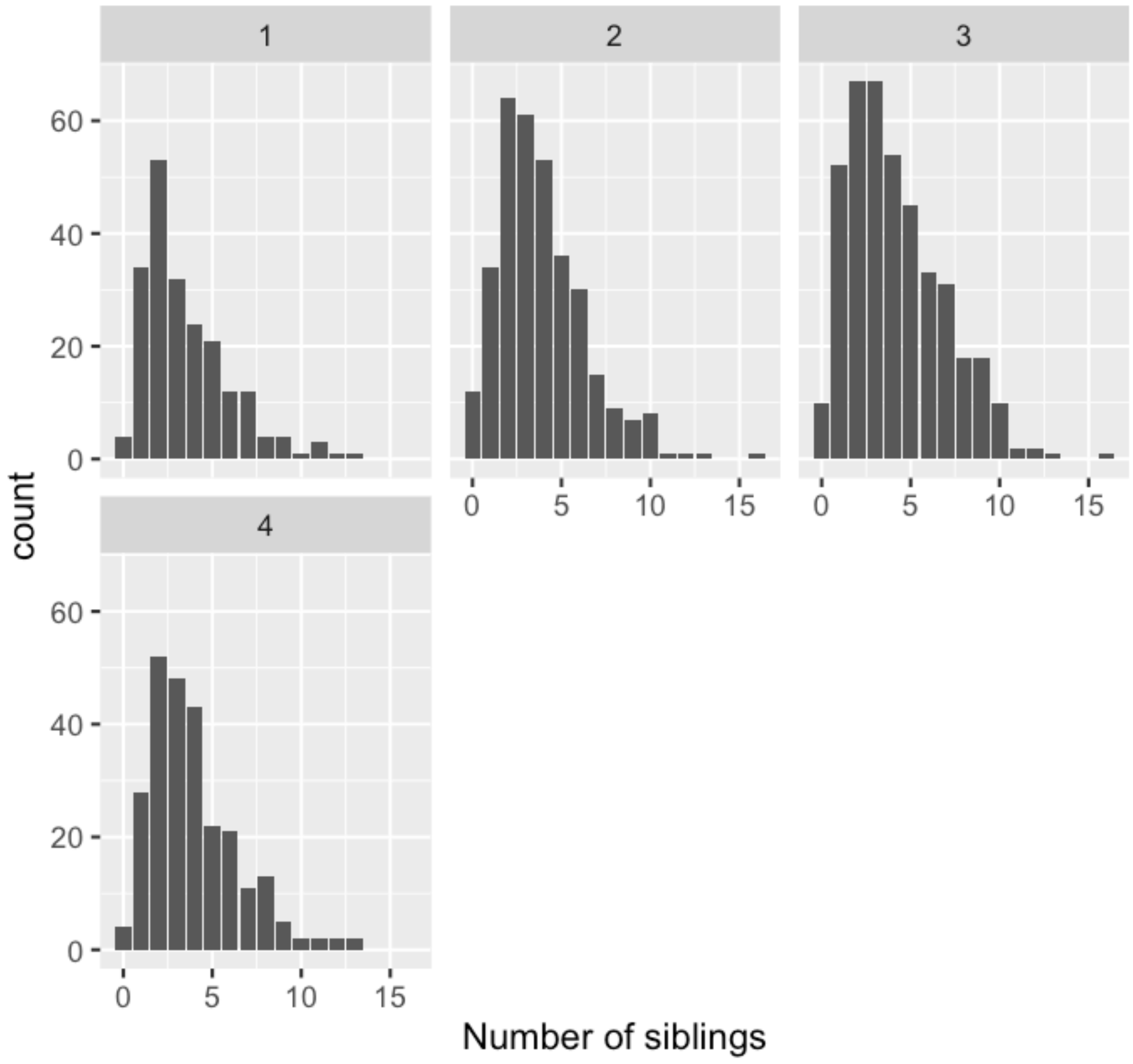
```
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_wrap(vars(region))
```

We can use **`facet_wrap()`** instead, if we want to use both multiple rows and columns for all the values of a variable



```r
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_wrap(vars(region),
             ncol = 3)
```

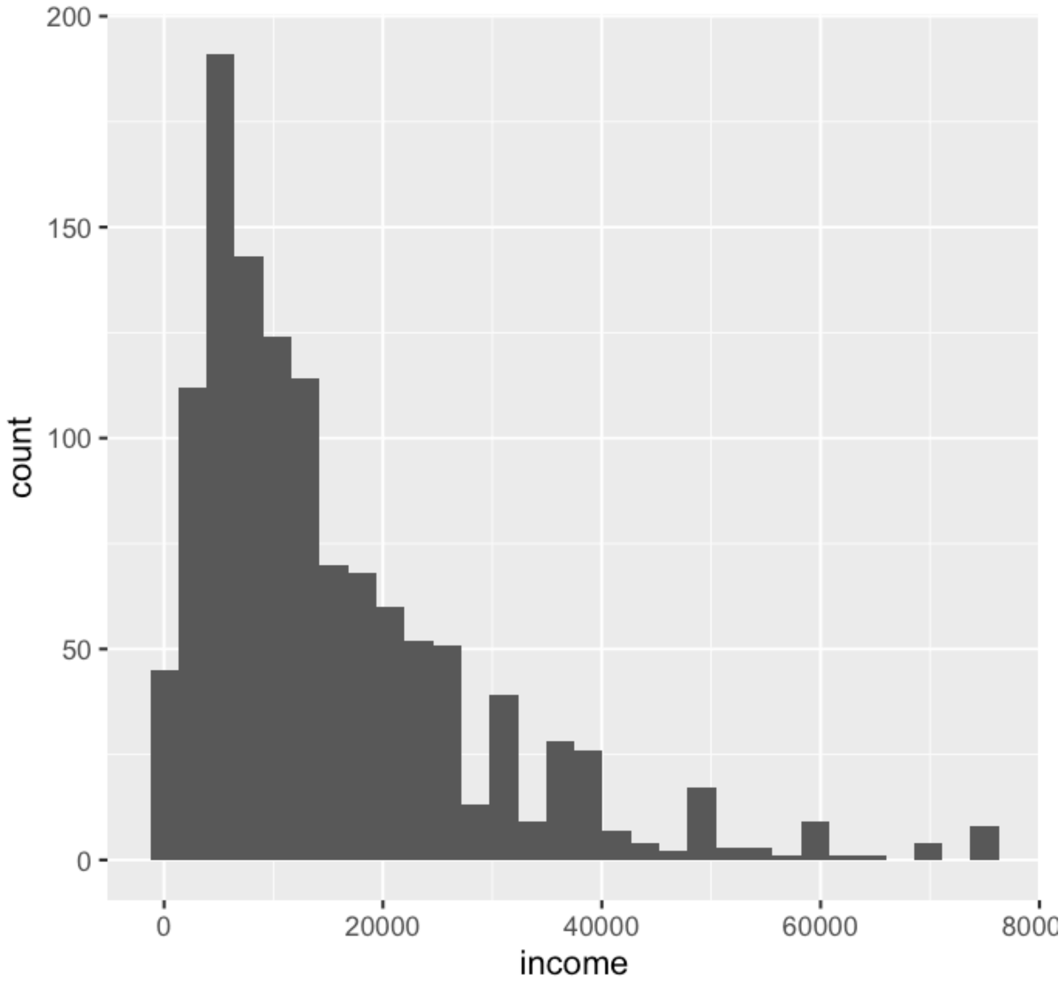
It tries to make a good decision, but you can override how many columns you want!



Wait, these look like histograms!

When we have a variable with a lot of possible values, we may want to bin them with a histogram

```
ggplot(nlsy) +
  geom_histogram(aes(x = income))
```



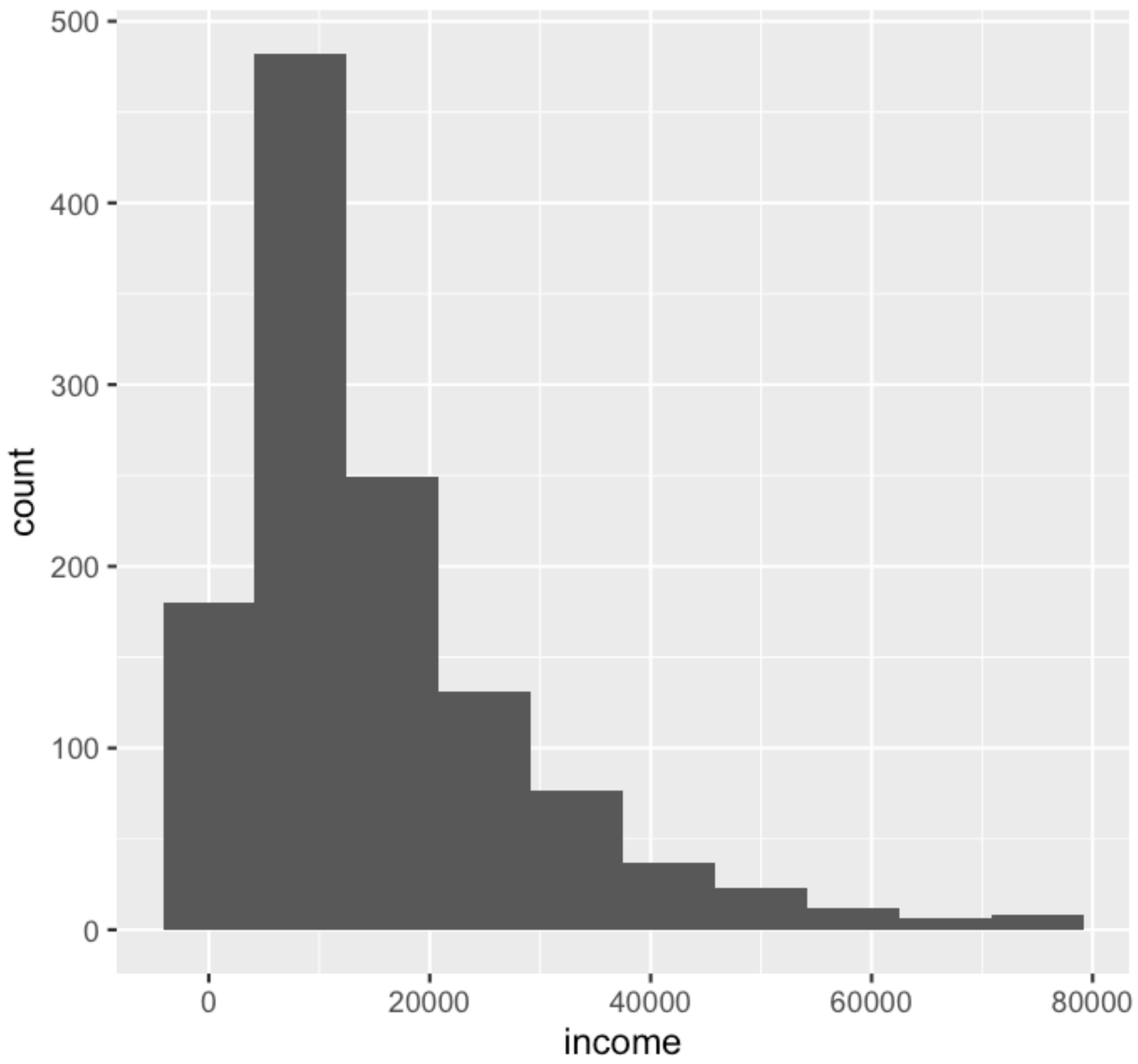
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

We used discrete values with `geom_bar()`, but with `geom_histogram()` we're combining values: the default is into 30 bins.

This is one of the most common warning messages I get in R!

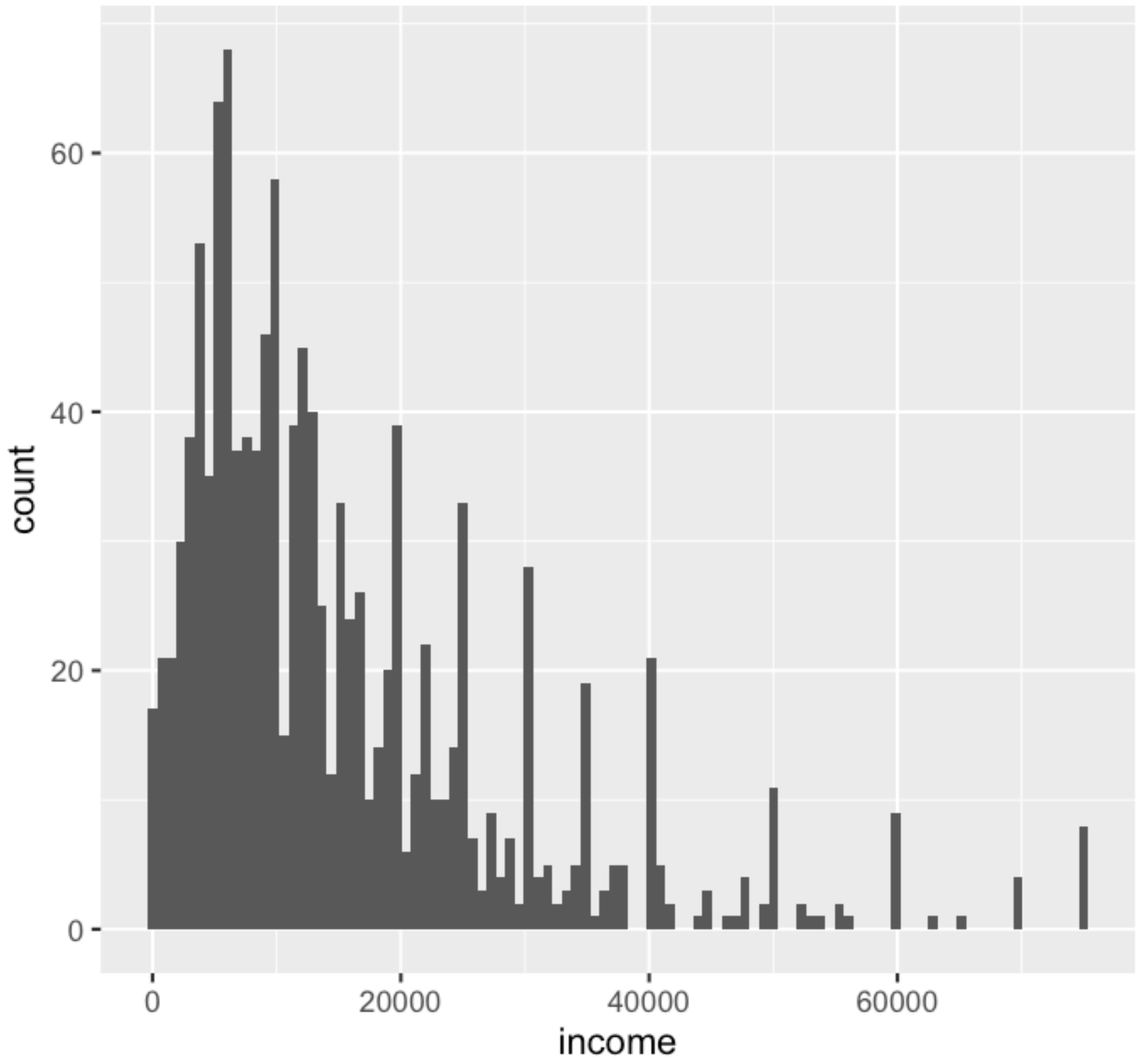
```
ggplot(data = nlsy) +
  geom_histogram(aes(x = income),
                 bins = 10)
```

We can actually use **`bins =`** instead, if we want!



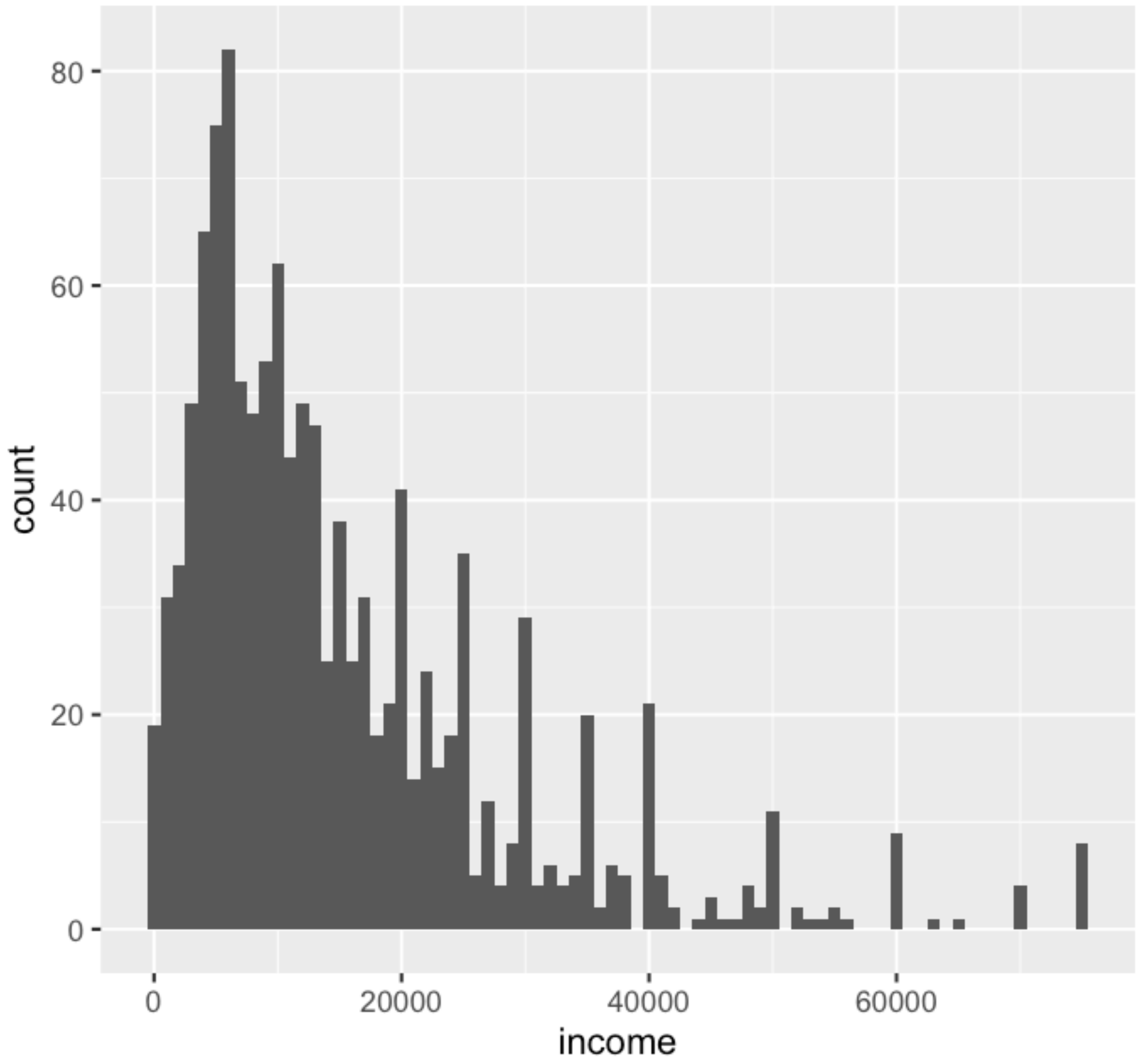
```r
ggplot(data = nlsy) +
  geom_histogram(aes(x = income),
                 bins = 100)
```

Be aware that you may interpret your data differently depending on how you bin it!



```r
ggplot(data = nlsy) +
  geom_histogram(aes(x = income),
                 binwidth = 1000)
```

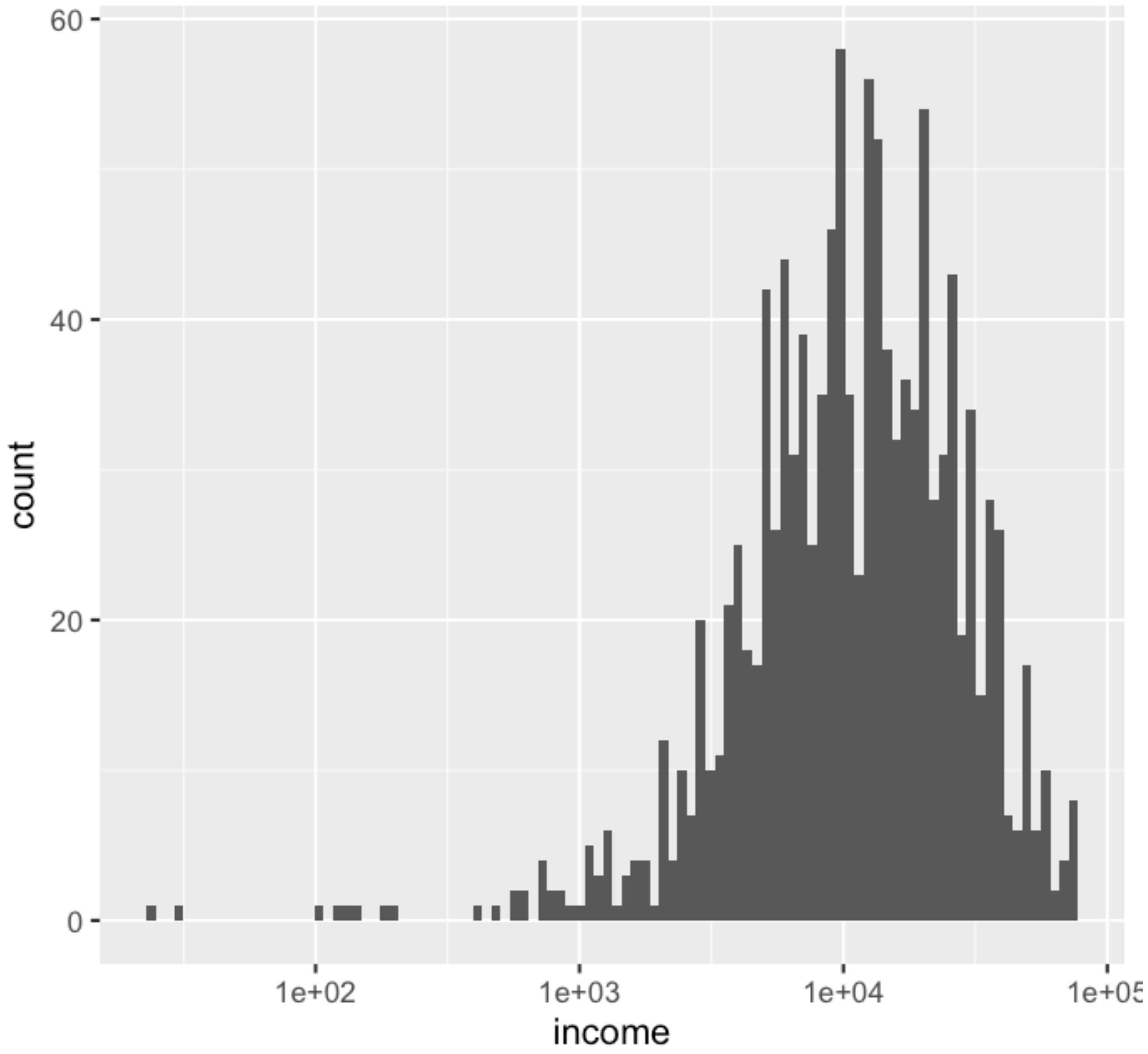
Sometimes the bin width actually has some meaning



```r
ggplot(data = nlsy) +
  geom_histogram(aes(x = income),
                 bins = 100) +
  scale_x_log10()
```

We can change the values of the axis just like we changed the values of the colors

There are a lot of `scale_x_()` and `scale_y_()` functions for you to explore!

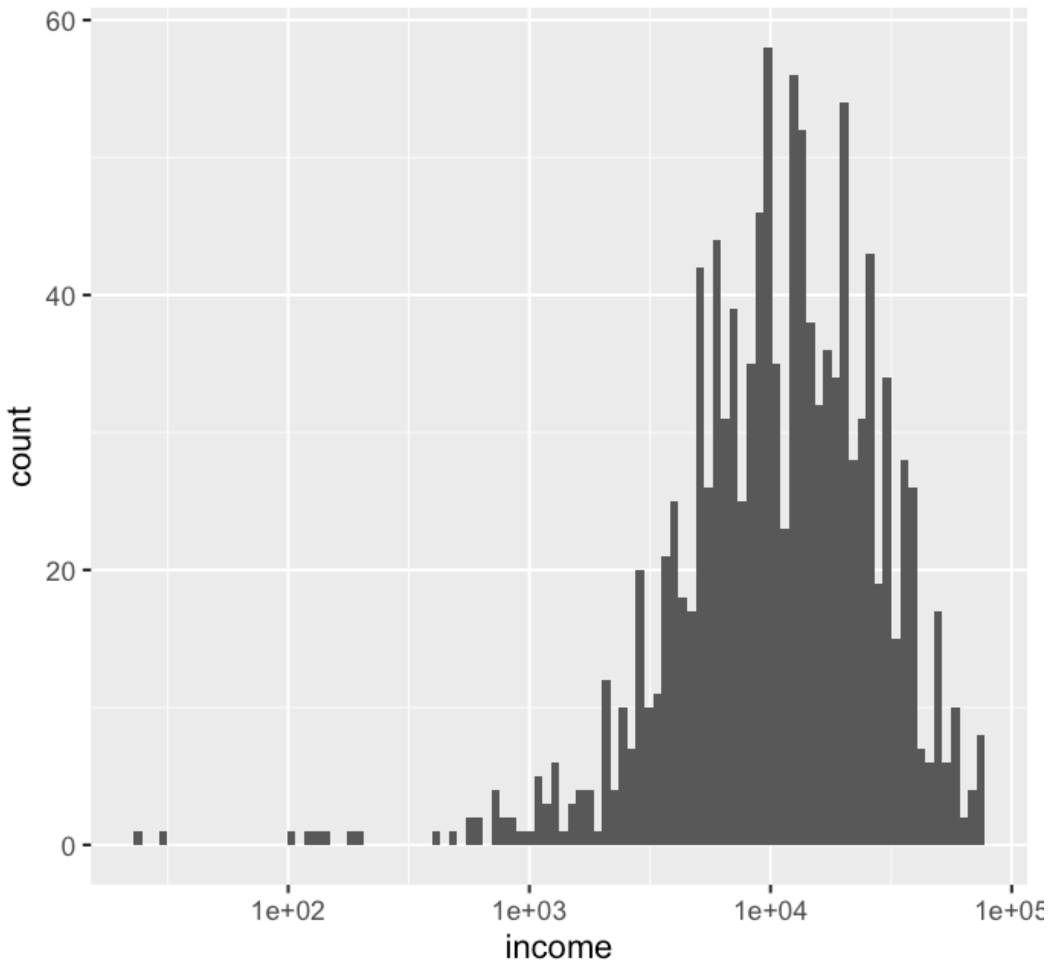


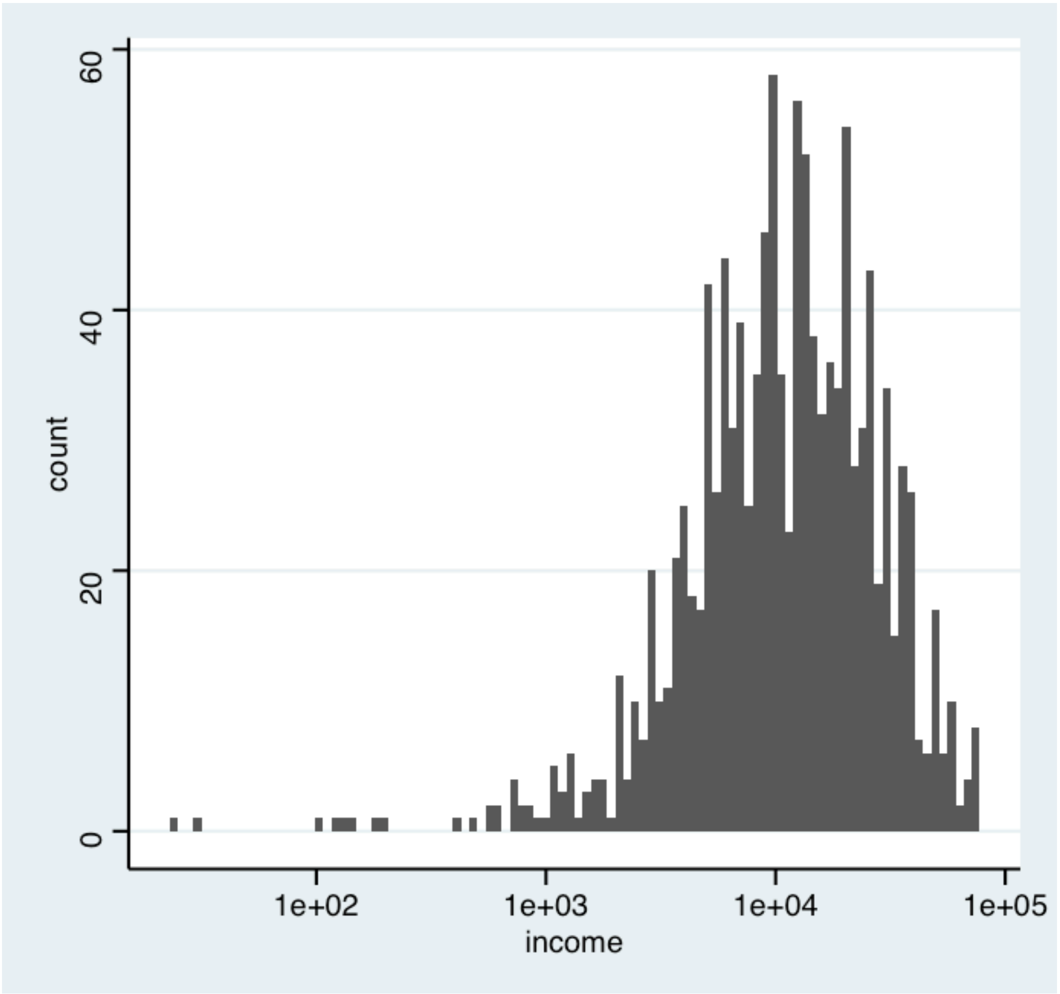
Exercises 4

1. When we're comparing distributions with very different numbers of observations, instead of scaling the y-axis like we did with the `facet_grid()` function, we might want to make density histograms. Use google to figure out how to make a density histogram of income. Facet it by region.
2. Make each of the regions in your histogram from part 1 a different color. (Hint: compare what `col =` and `fill =` do to histograms).
3. Instead of a log-transformed x-axis, make a square-root transformed x-axis.
4. Doing part 3 squishes the labels on the x-axis. Using the `breaks =` argument that all the `scale_x_()` functions have, make labels at 1000, 10000, 25000, and 50000.

Finally, themes

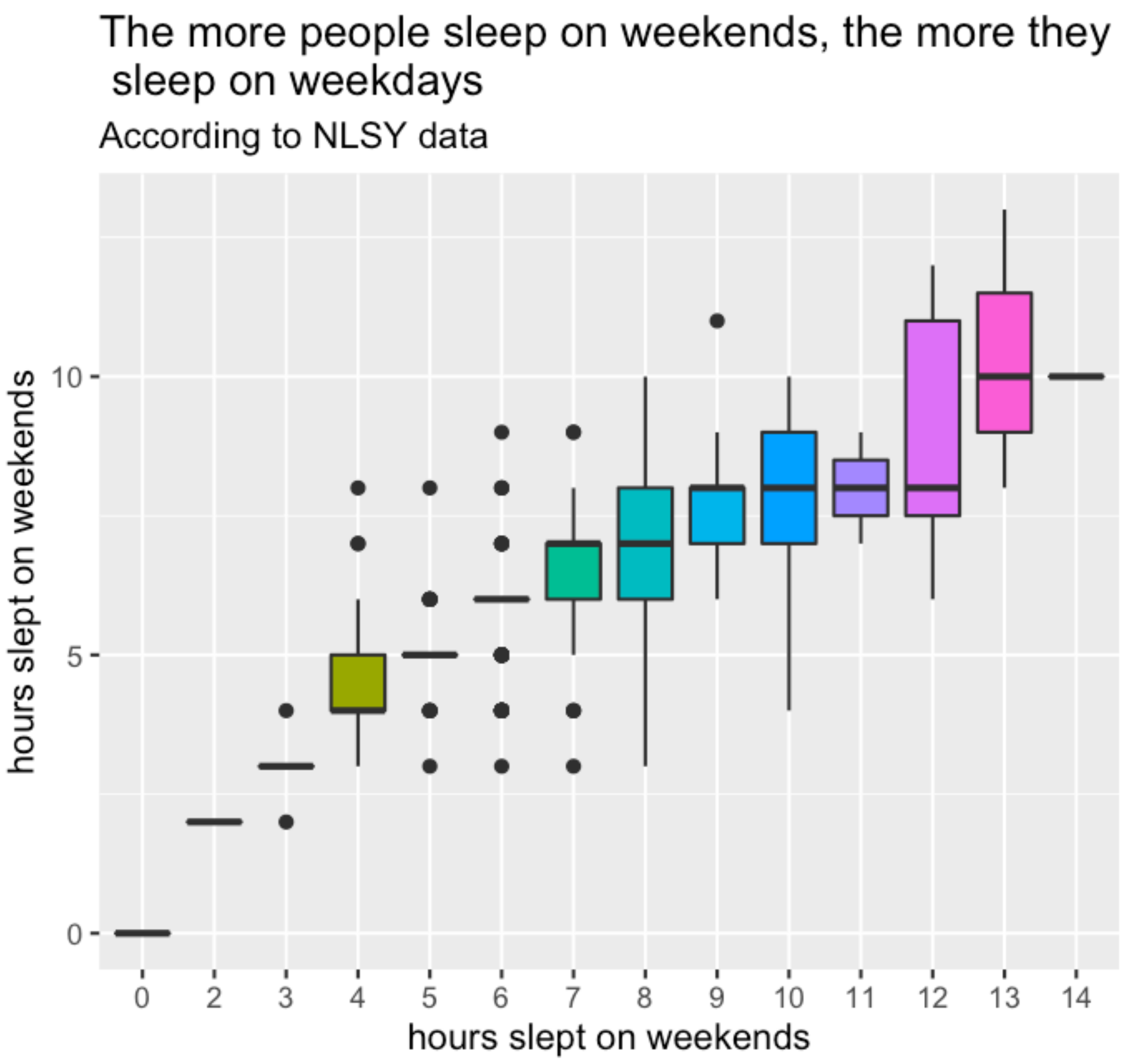
You probably recognize the ggplot theme. But did you know you can trick people into thinking you made your figures in Stata?





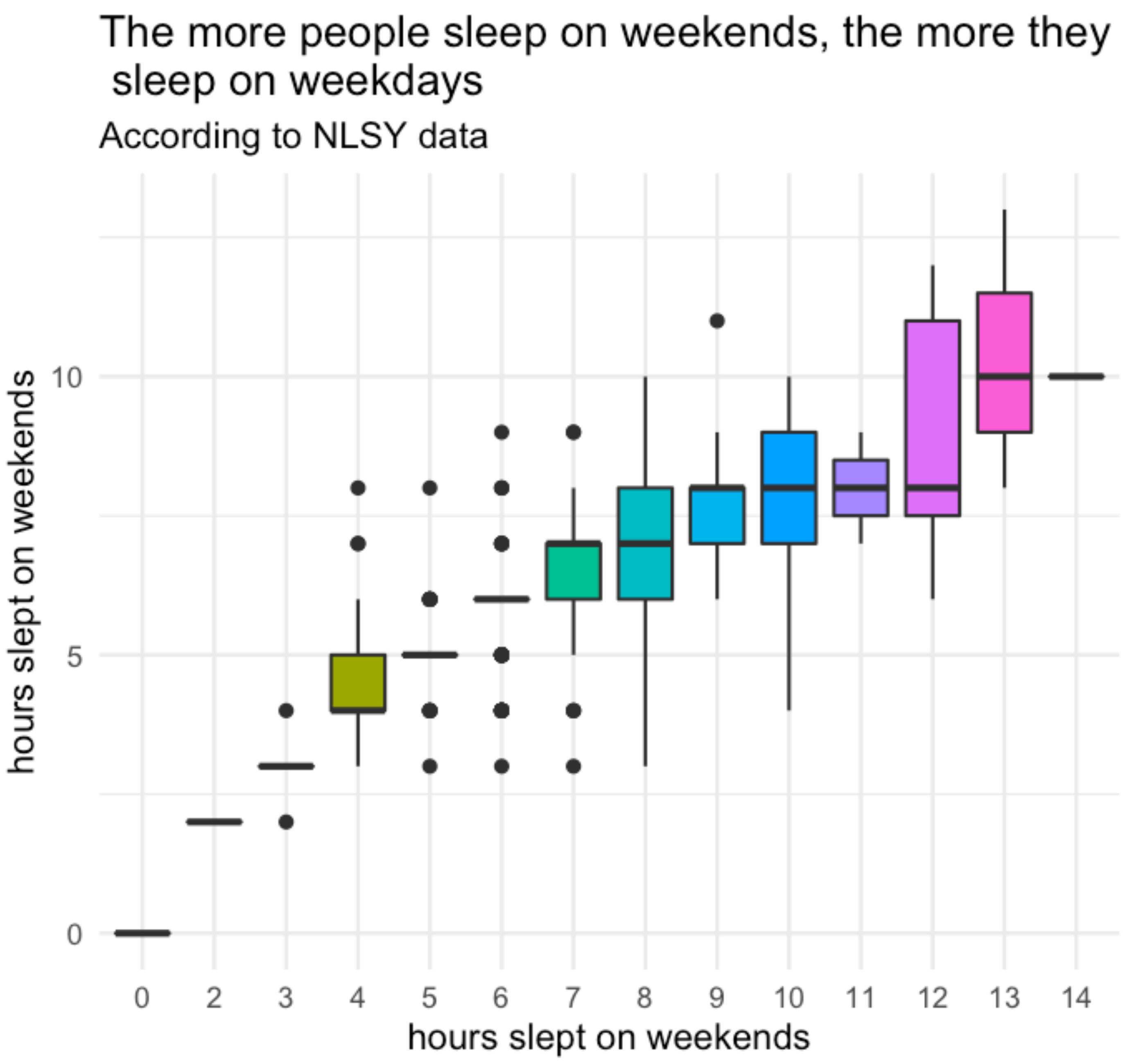
```
ggplot(data = nlsy) +
  geom_boxplot(aes(
    x = factor(sleep_wknd),
    y = sleep_wkdy,
    fill = factor(sleep_wknd))) +
  scale_fill_discrete(guide = FALSE) +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on weeken
       subtitle = "According to NLSY data")
```

Can you figure out what each chunk of this code is doing to the figure?

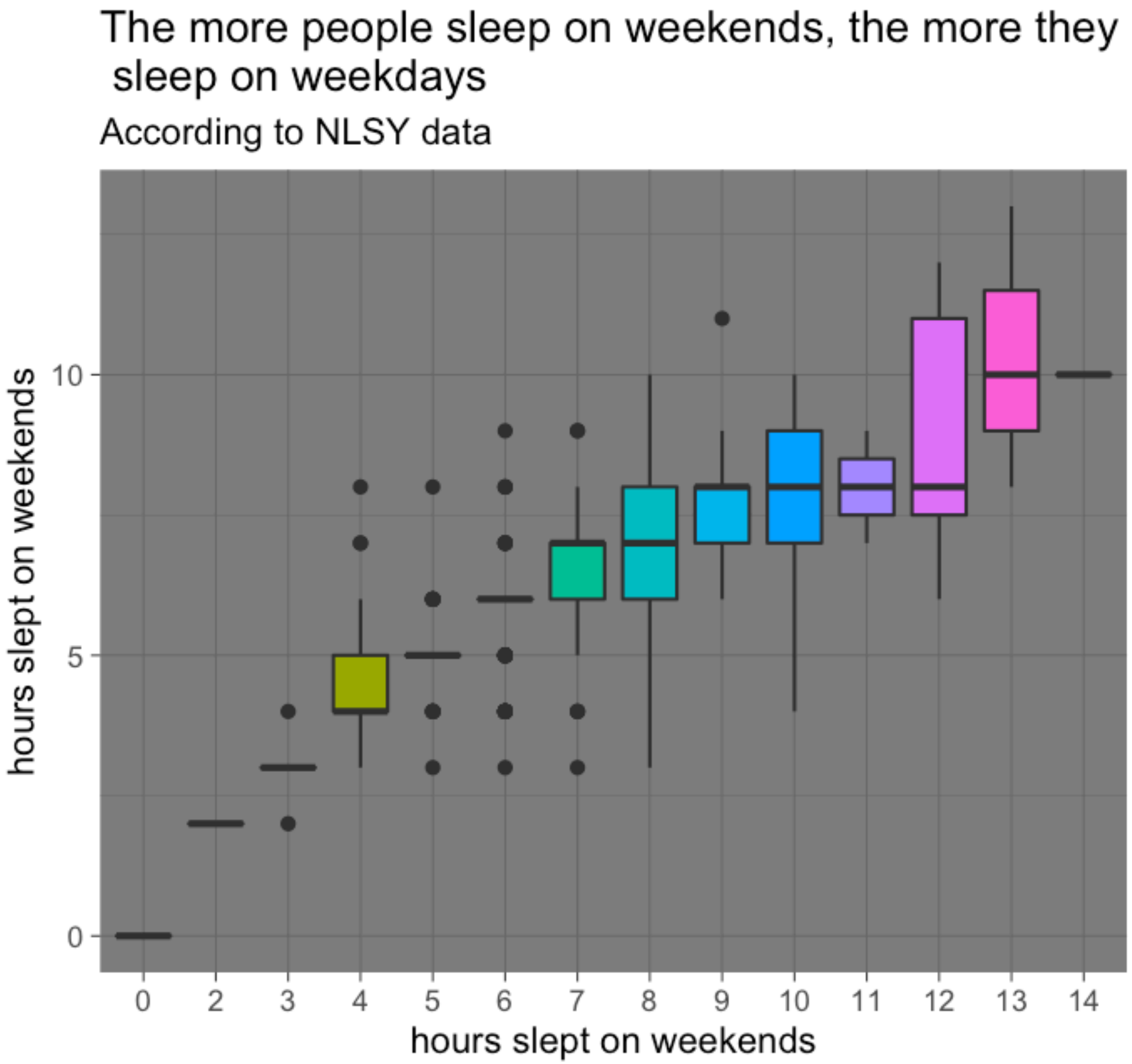


```r
ggplot(data = nlsy) +
  geom_boxplot(aes(
    x = factor(sleep_wknd),
    y = sleep_wkdy,
    fill = factor(sleep_wknd))) +
  scale_fill_discrete(guide = FALSE) +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on weeken
       subtitle = "According to NLSY data") +
  theme_minimal()
```

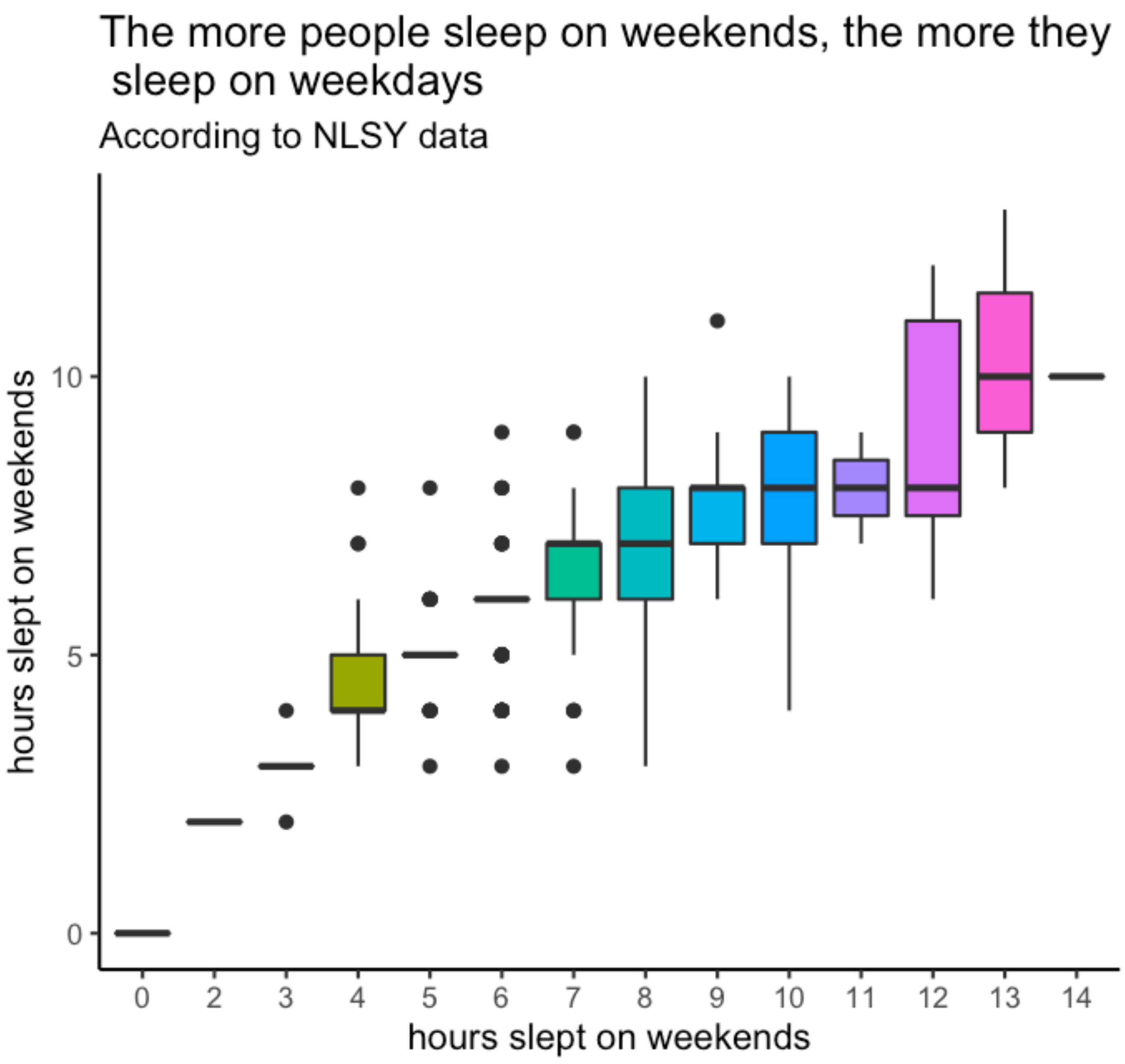
We can change the overall theme



```r
ggplot(data = nlsy) +
  geom_boxplot(aes(
    x = factor(sleep_wknd),
    y = sleep_wkdy,
    fill = factor(sleep_wknd))) +
  scale_fill_discrete(guide = FALSE) +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on weeken
       subtitle = "According to NLSY data") +
  theme_dark()
```



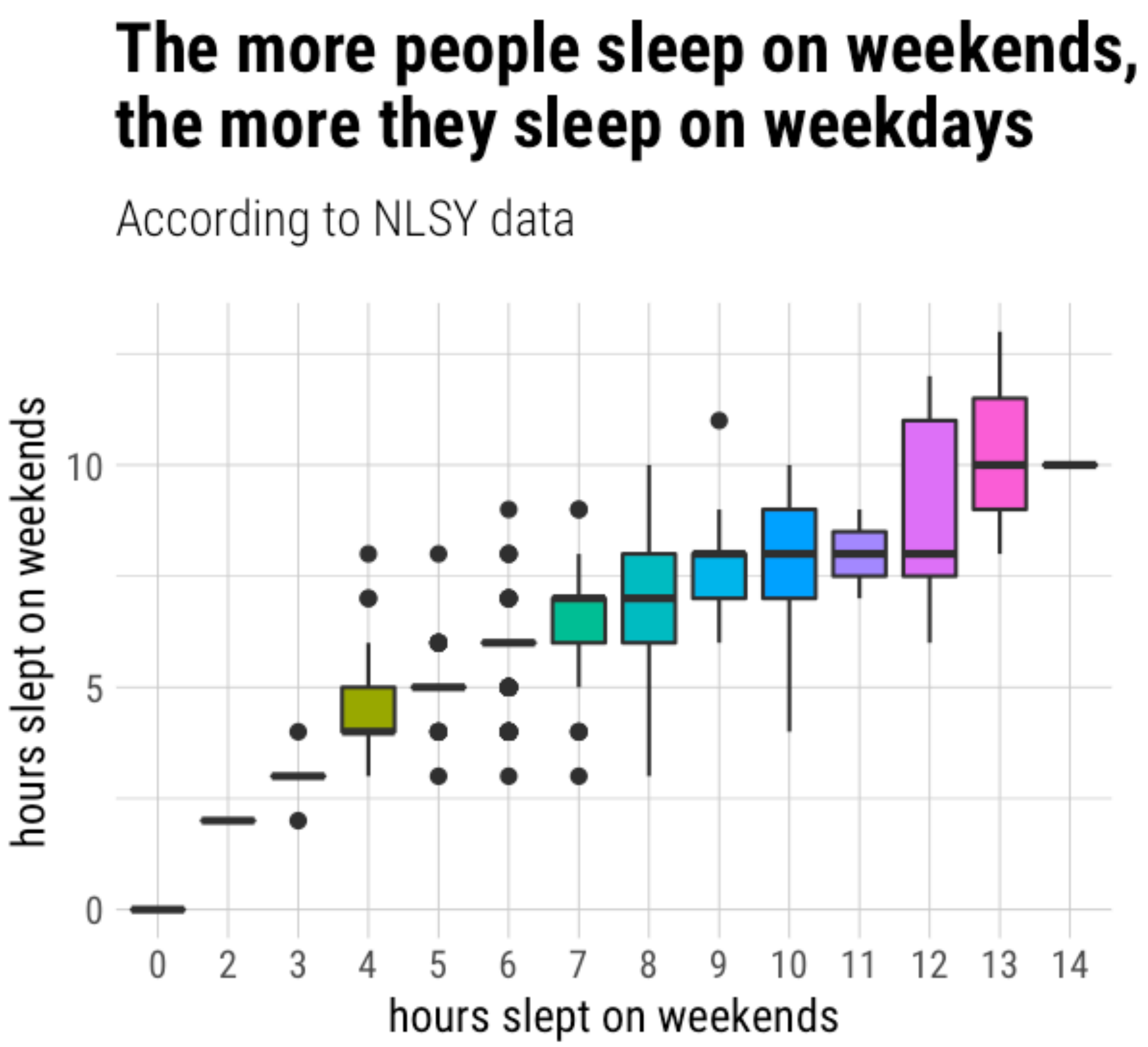
```
ggplot(data = nlsy) +
  geom_boxplot(aes(
    x = factor(sleep_wknd),
    y = sleep_wkdy,
    fill = factor(sleep_wknd))) +
  scale_fill_discrete(guide = FALSE) +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on weeken
       subtitle = "According to NLSY data") +
  theme_classic()
```



```r
ggplot(data = nlsy) +
  geom_boxplot(aes(
    x = factor(sleep_wknd),
    y = sleep_wkdy,
    fill = factor(sleep_wknd))) +
  scale_fill_discrete(guide = FALSE) +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on weeken
       subtitle = "According to NLSY data") +
  theme_void()
```

```r
ggplot(data = nlsy) +
  geom_boxplot(aes(
    x = factor(sleep_wknd),
    y = sleep_wkdy,
    fill = factor(sleep_wknd))) +
  scale_fill_discrete(guide = FALSE) +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on weeken
       subtitle = "According to NLSY data") +
  louisahstuff::my_theme()
```

Here is a good list of themes and instructions to make your own: https://www.datanovia.com/en/blog/ggplot-themes-gallery/



Finally, save it!

If your data changes, you can easily run the whole script again

```r
library(tidyverse)
dataset <- read_csv("dataset.csv")
ggplot(dataset) +
  geom_point(aes(x = xvar, y = yvar))
ggsave(filename = "scatterplot.pdf")
```

More resources

- Cheat sheet: https://www.rstudio.com/resources/cheatsheets/#ggplot2
- Catalog: http://shiny.stat.ubc.ca/r-graph-catalog/
- Cookbook: http://www.cookbook-r.com/Graphs/
- Official package reference: https://ggplot2.tidyverse.org/index.html

Final challenge

Recreate this plot using the NLSY data!