Introduction to R

Week 2: Making figures

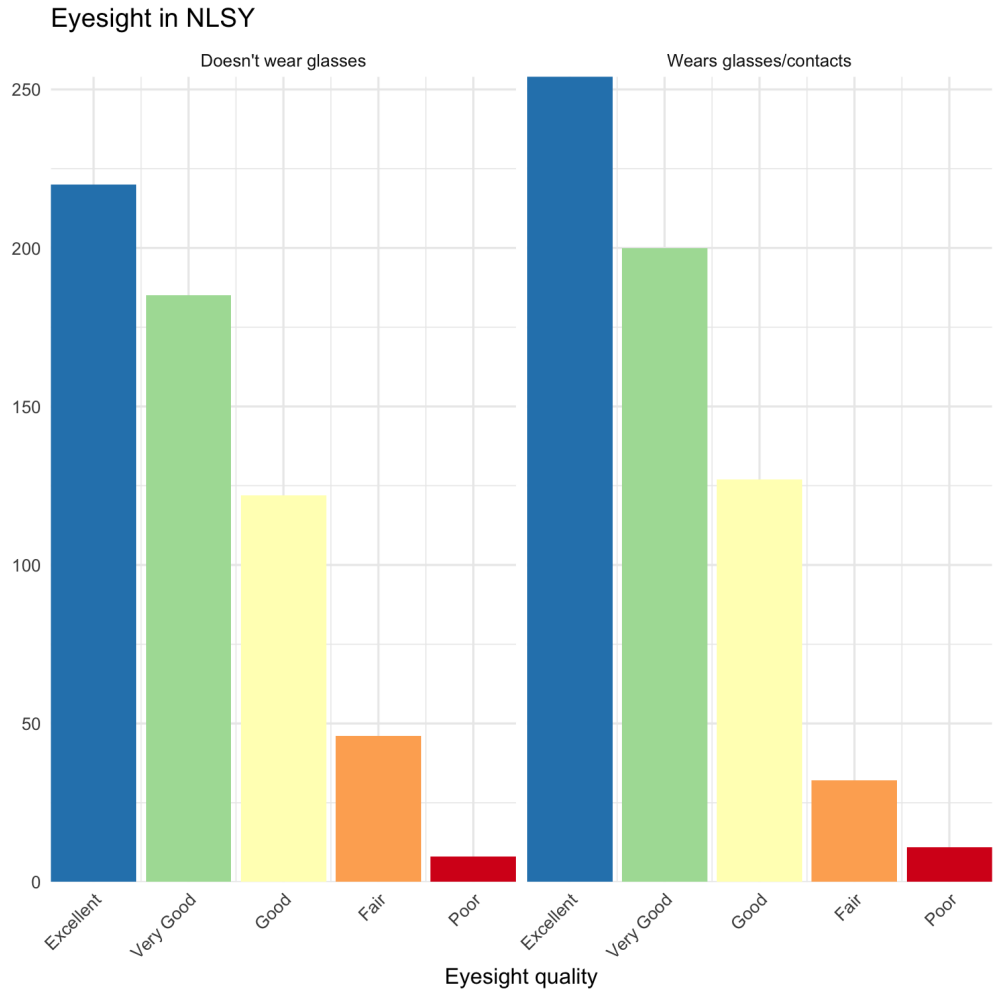
Louisa Smith

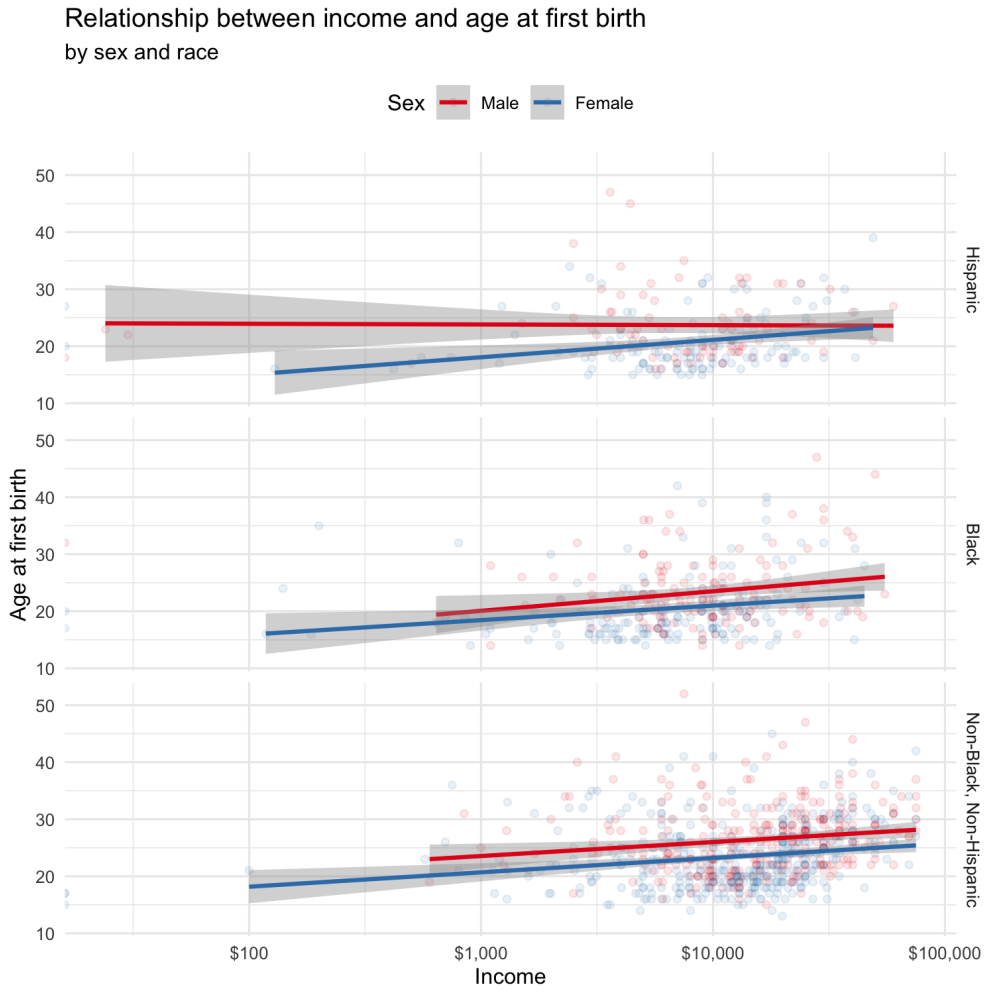
July 20 - July 24

Let's make our data...

beautiful

#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

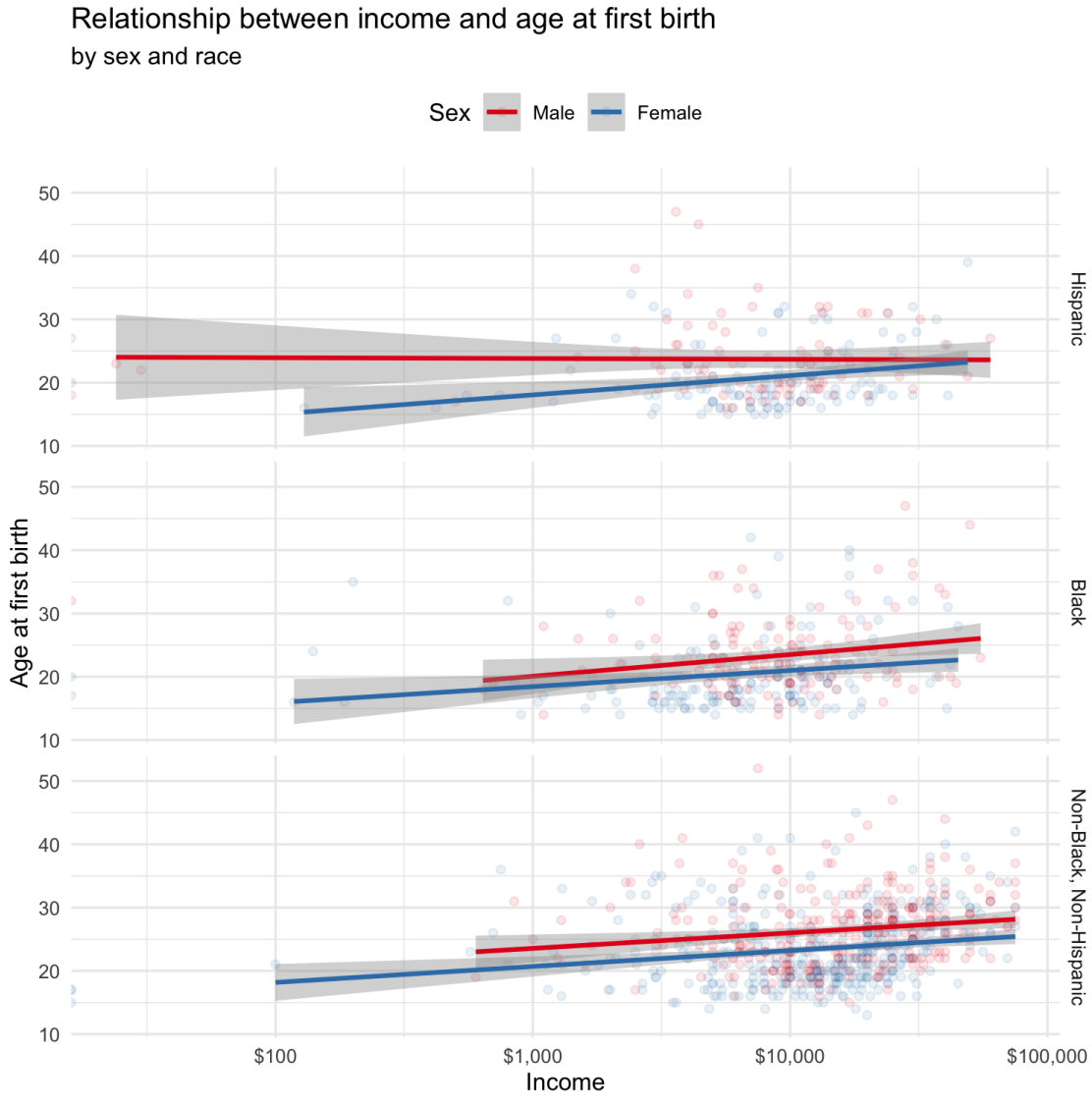

Relationship between income and age at first birth
by sex and race
Sex
Male
Female
Hispanic
Black
Non-Black, Non-Hispanic
Age at first birth
50
40
30
20
10
$100
$1,000
$10,000
$100,000
Income

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 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!

```
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()`

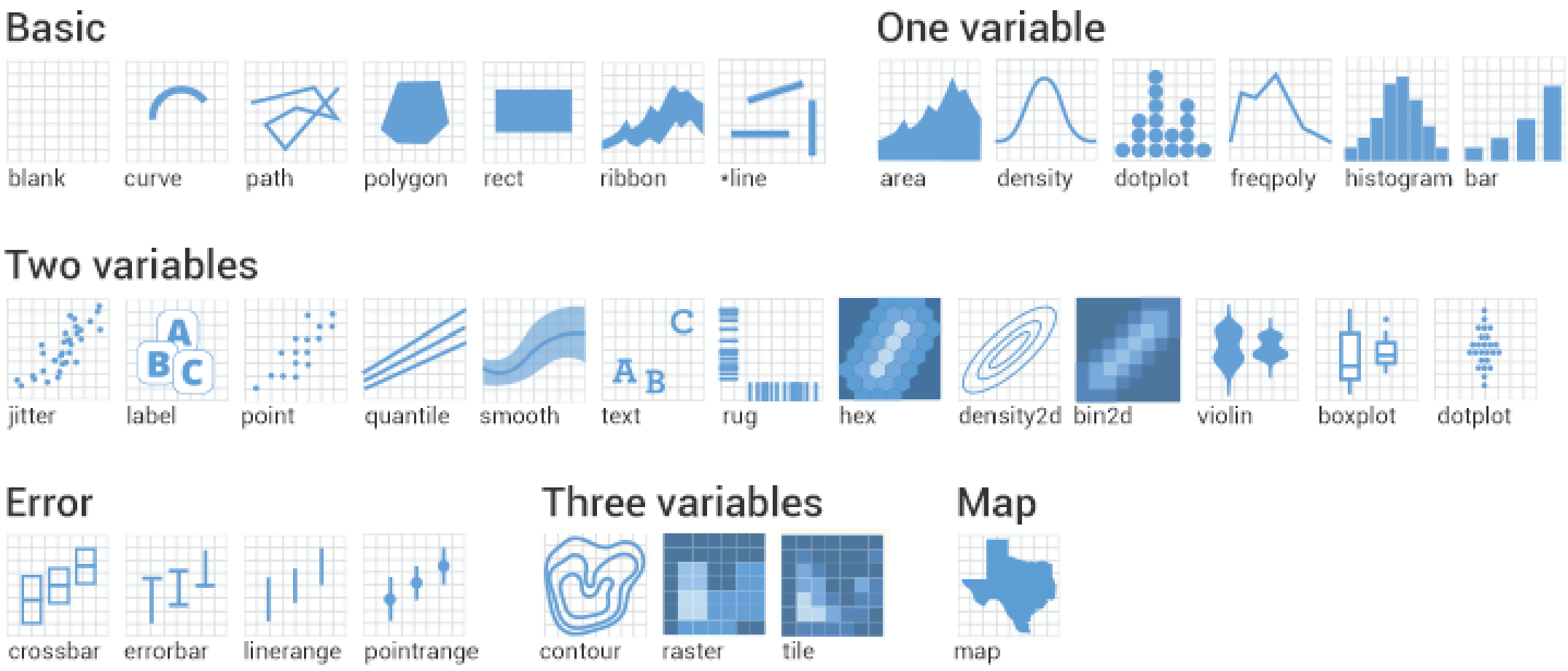


Image via https://nbisweden.github.io/RaukR-2019/ggplot/presentation/ggplot_presentation.html

Basic example

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

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

Other geoms require other arguments

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

> Notice that the variable names are not in quotation marks

Basic example

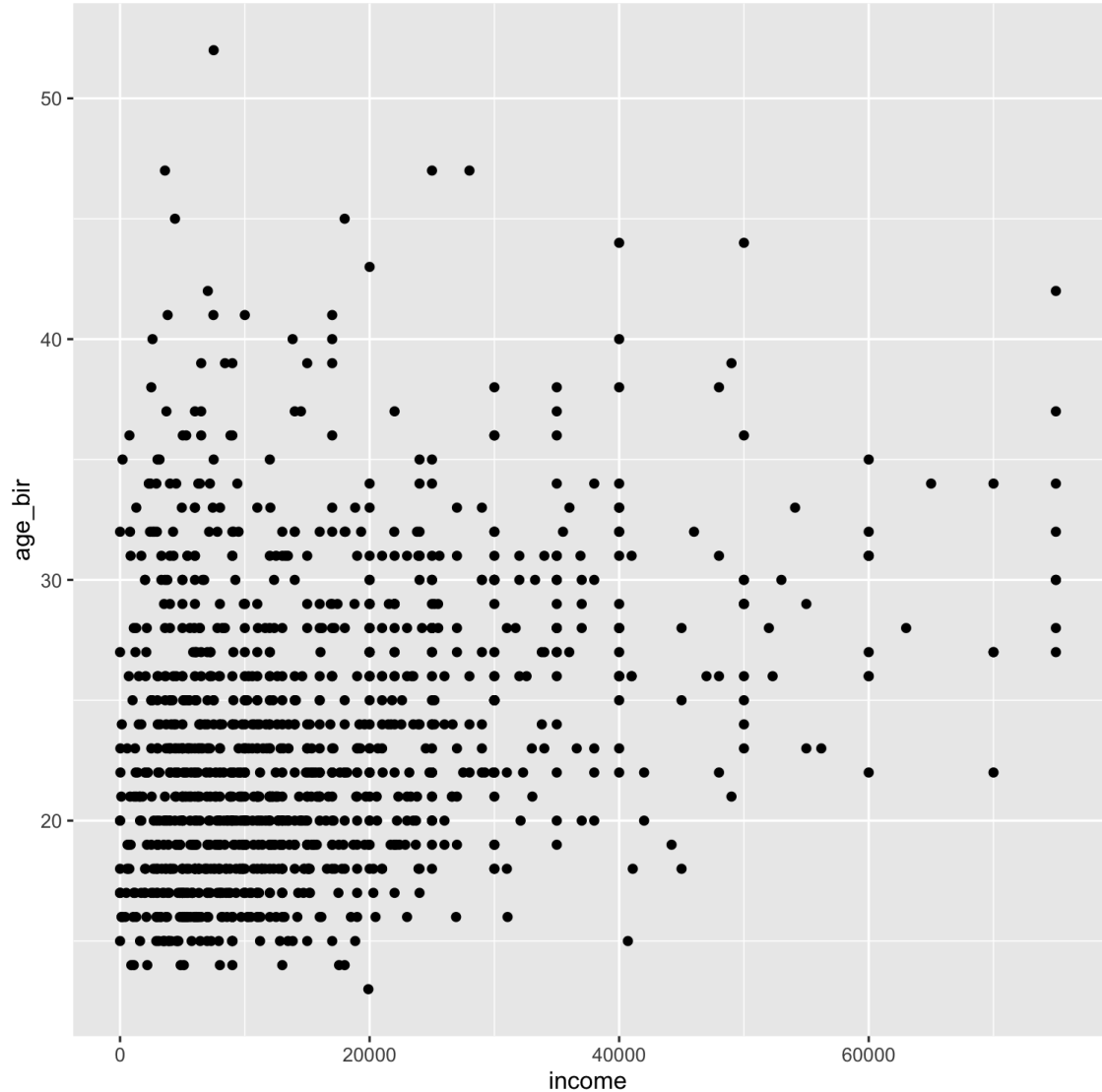
```
ggplot(data = nlsy, aes(x = income, y = age_bir, <characteristic> = {othvar}, ...))
      geom_point(<characteristic> = "value", ...) +
      ...
```

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)

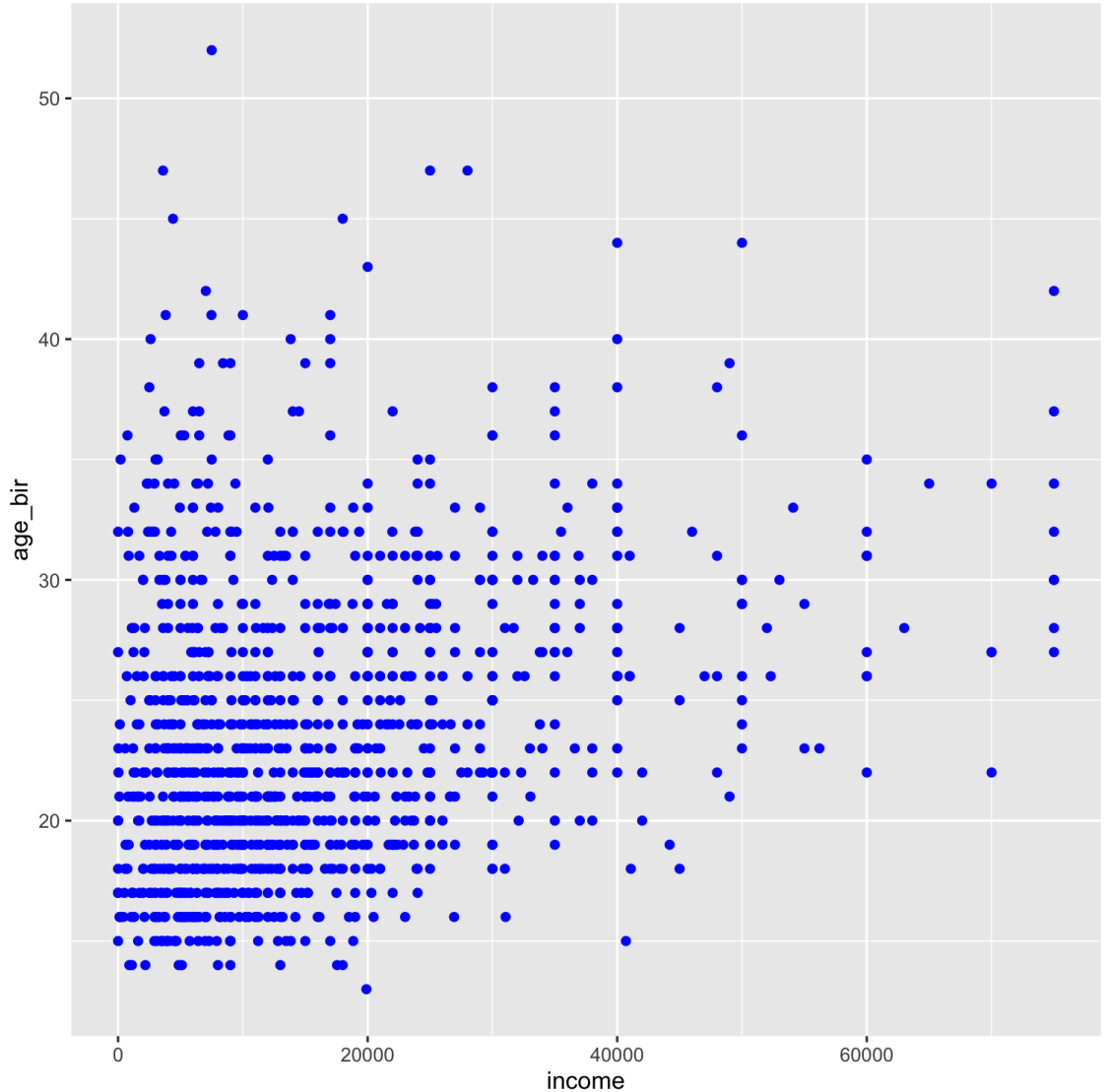
```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir))
```

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



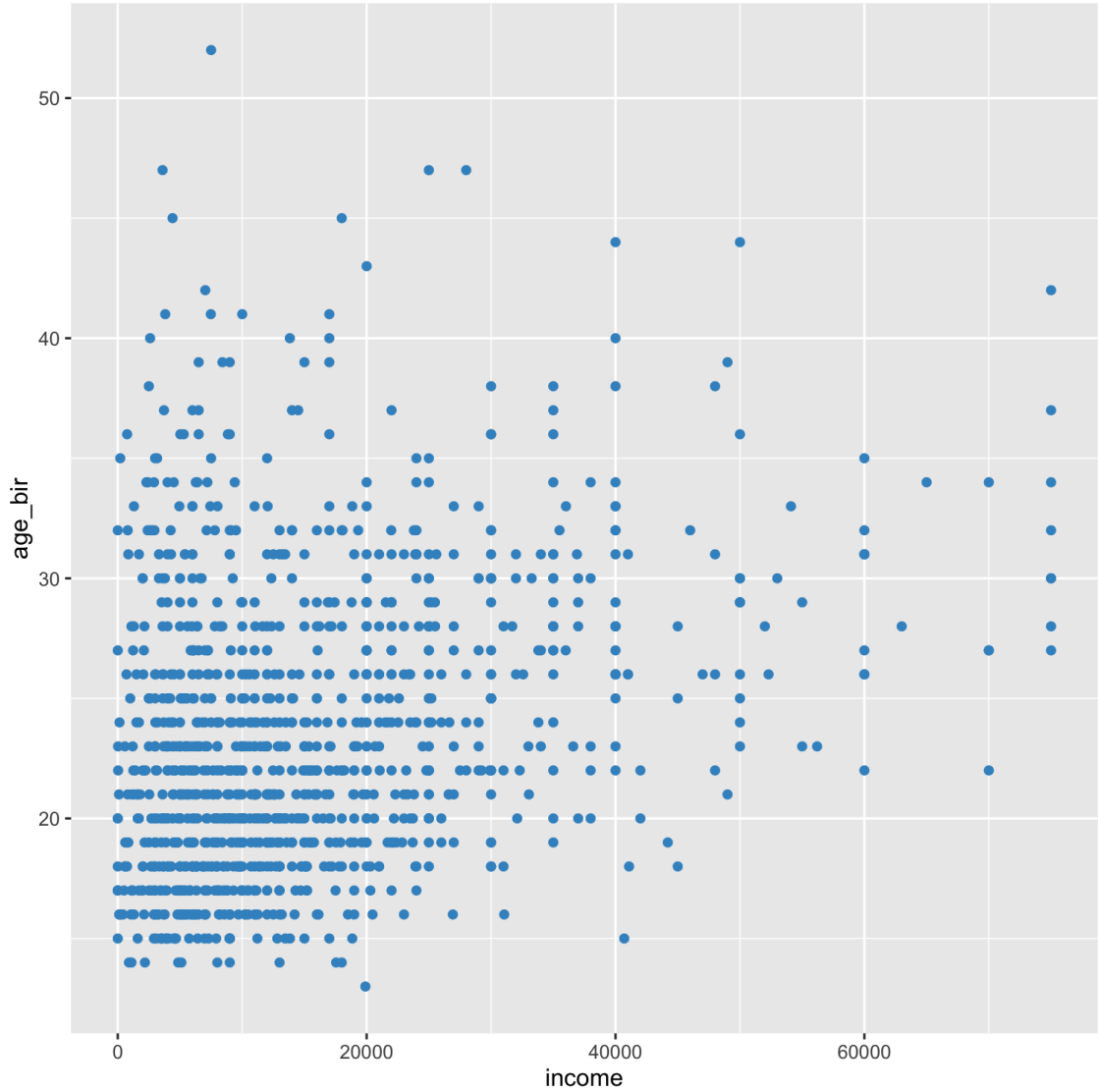
```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir),
               color = "blue")
```

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



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

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

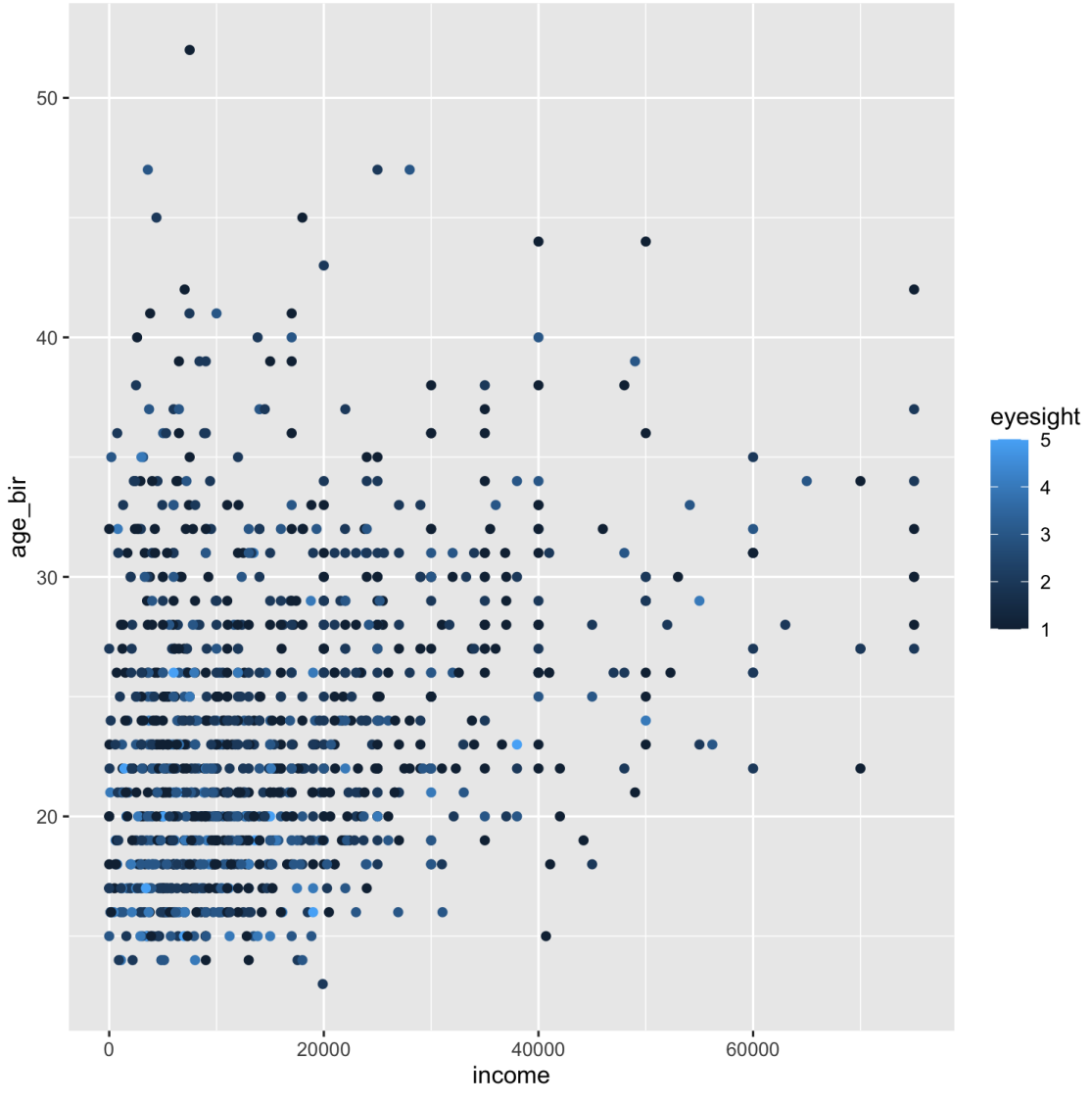


```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
                   color = eyesight))
```

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.

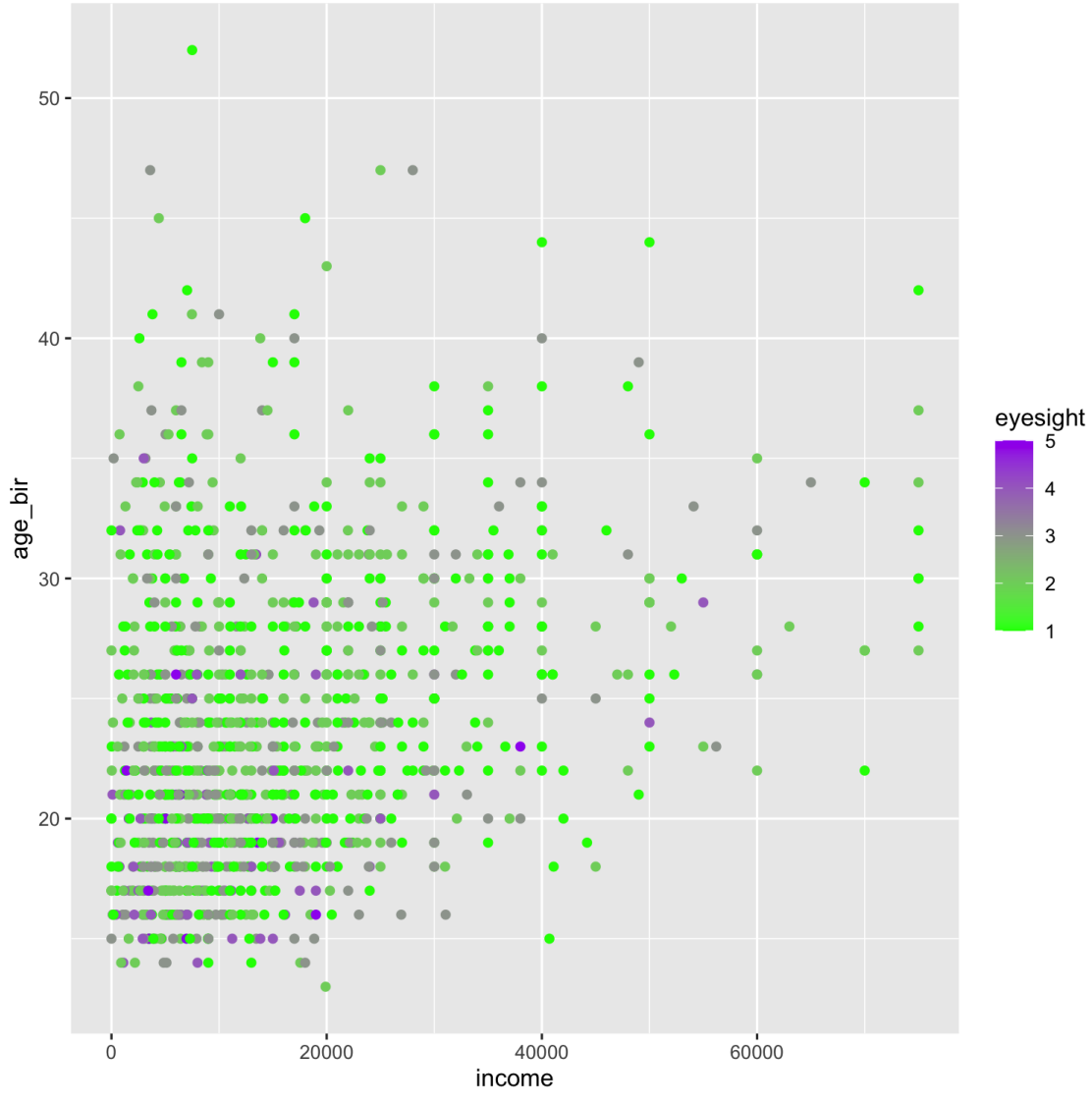
- But they're kind of hard to distinguish!



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

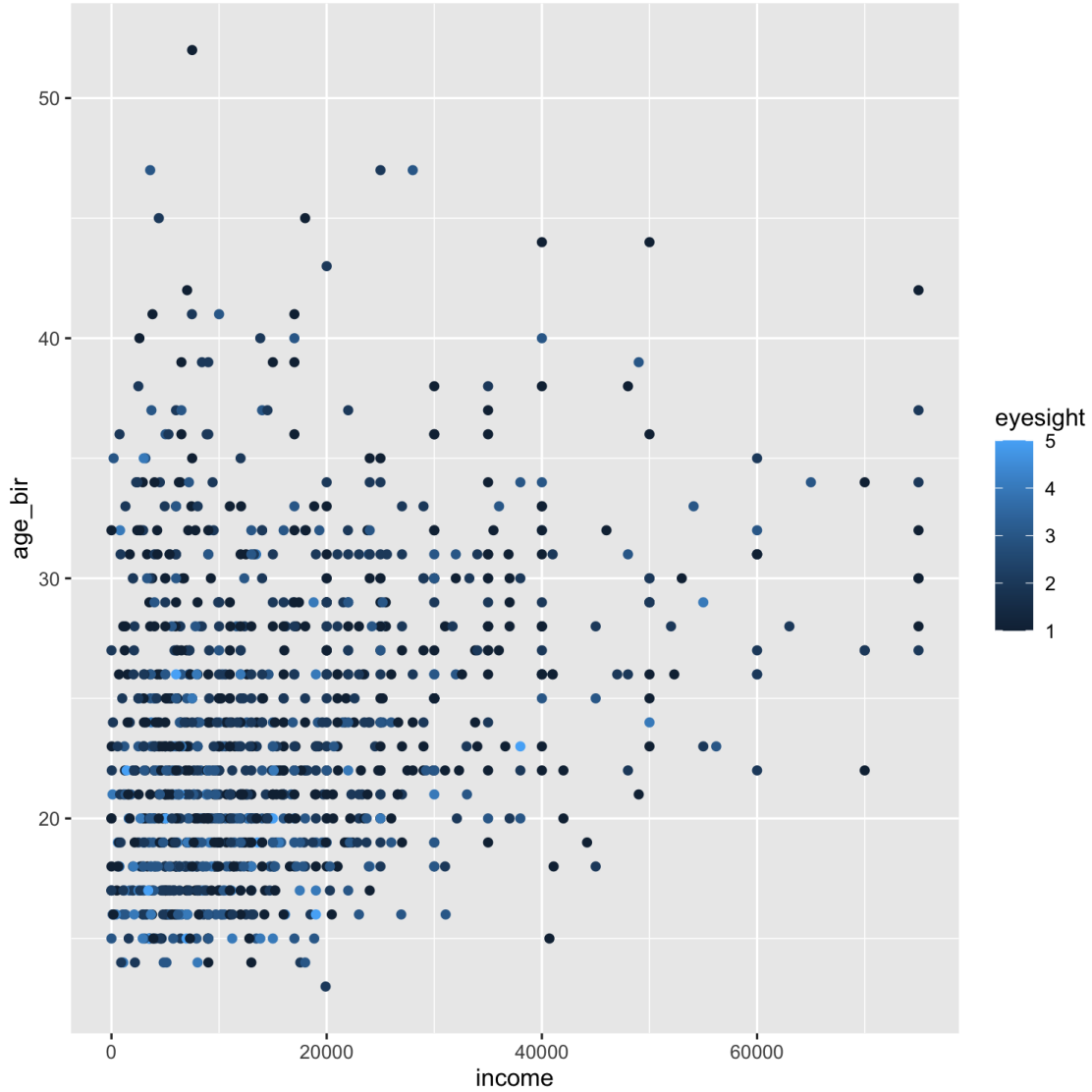
We can map the values of `eyesight` to a different continuous scale using `scale_color_gradient()`

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



```
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.

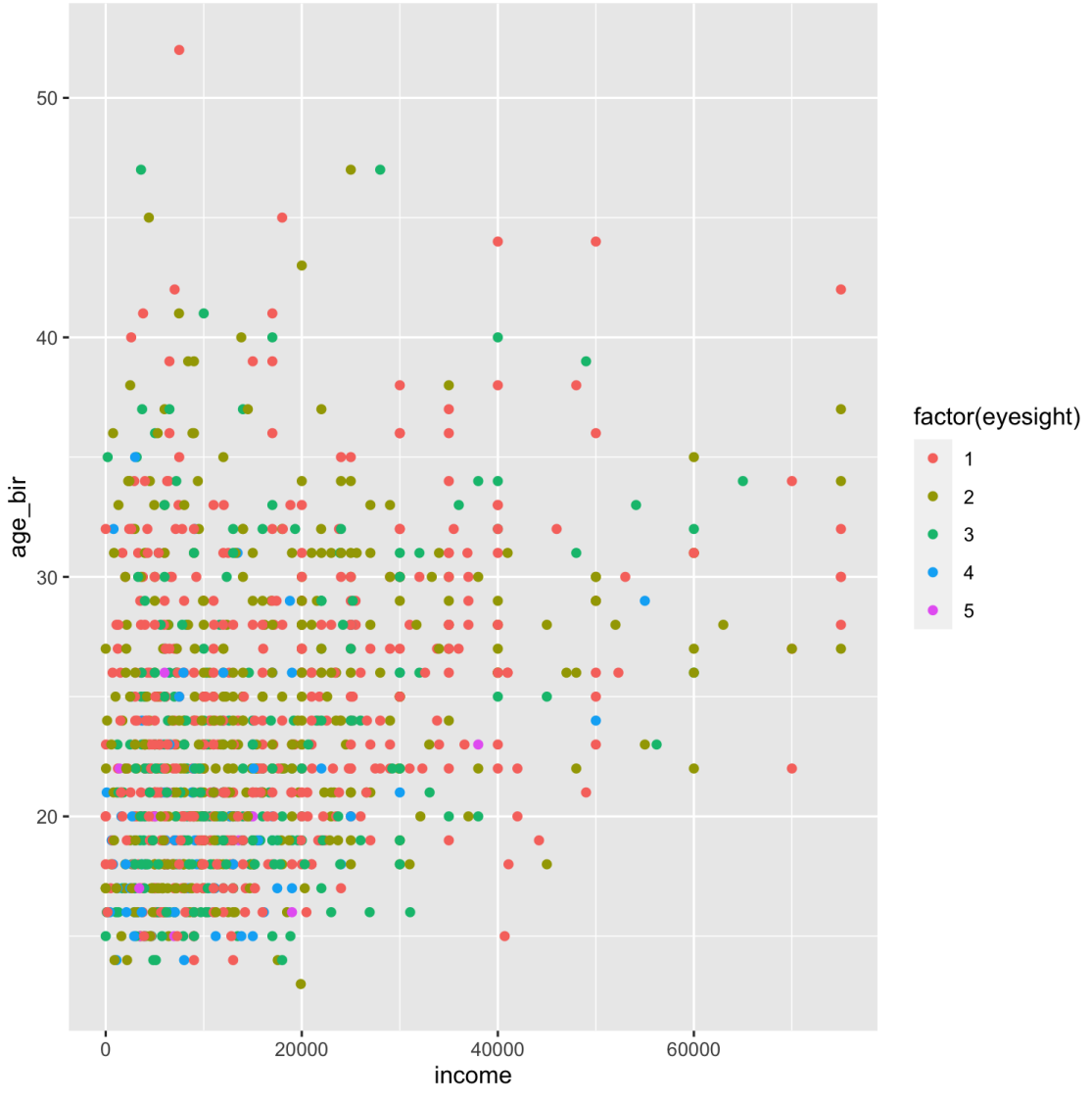


```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
               color = factor(eyesight)))
```

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

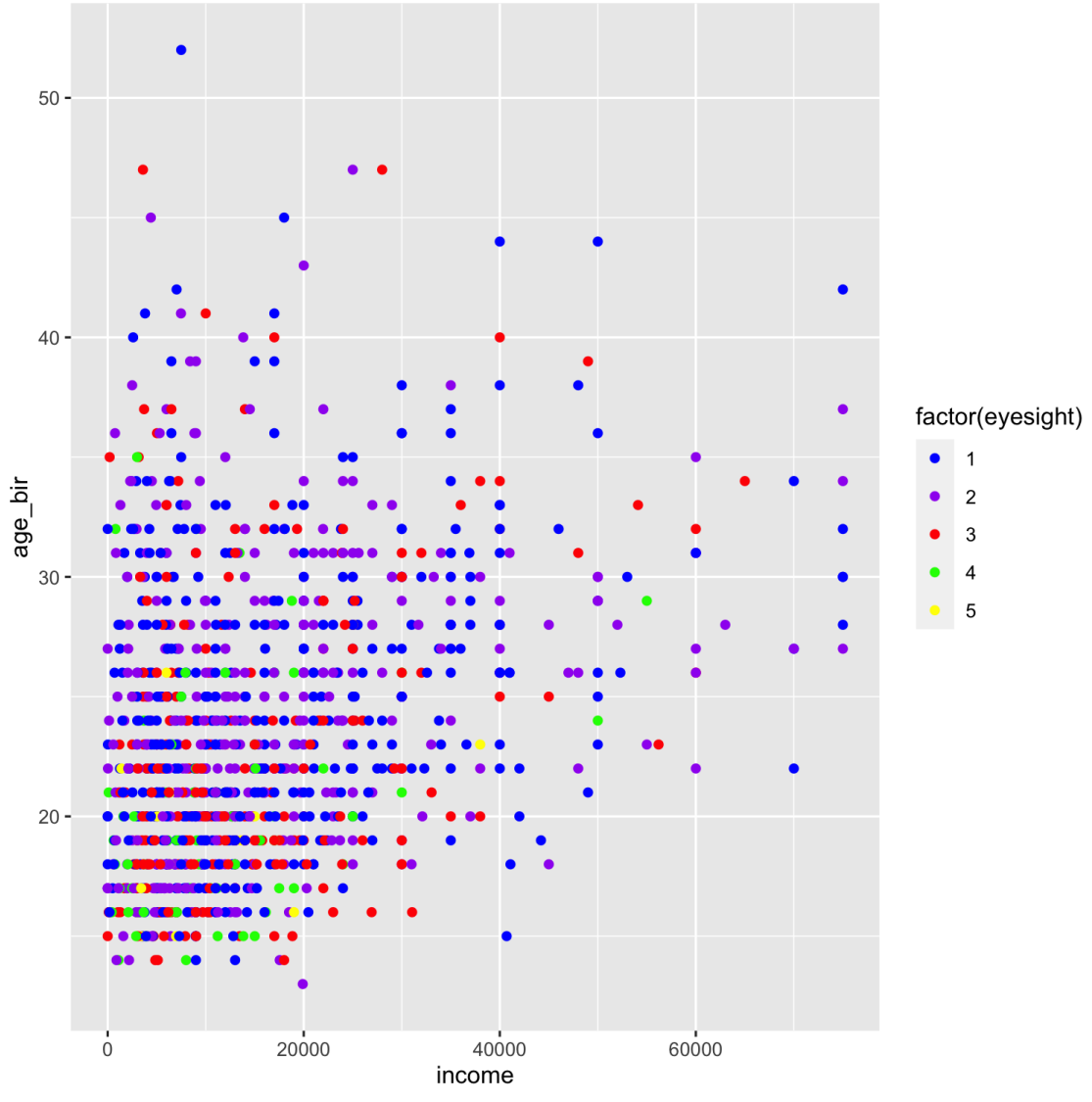


```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
               color = factor(eyesight))) +
scale_color_manual(
    values = c("blue", "purple", "red",
"green", "yellow"))
```

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

Before we used `scale_color_gradient`, now `scale_color_manual`.

- There are a lot of options that follow the same naming scheme.

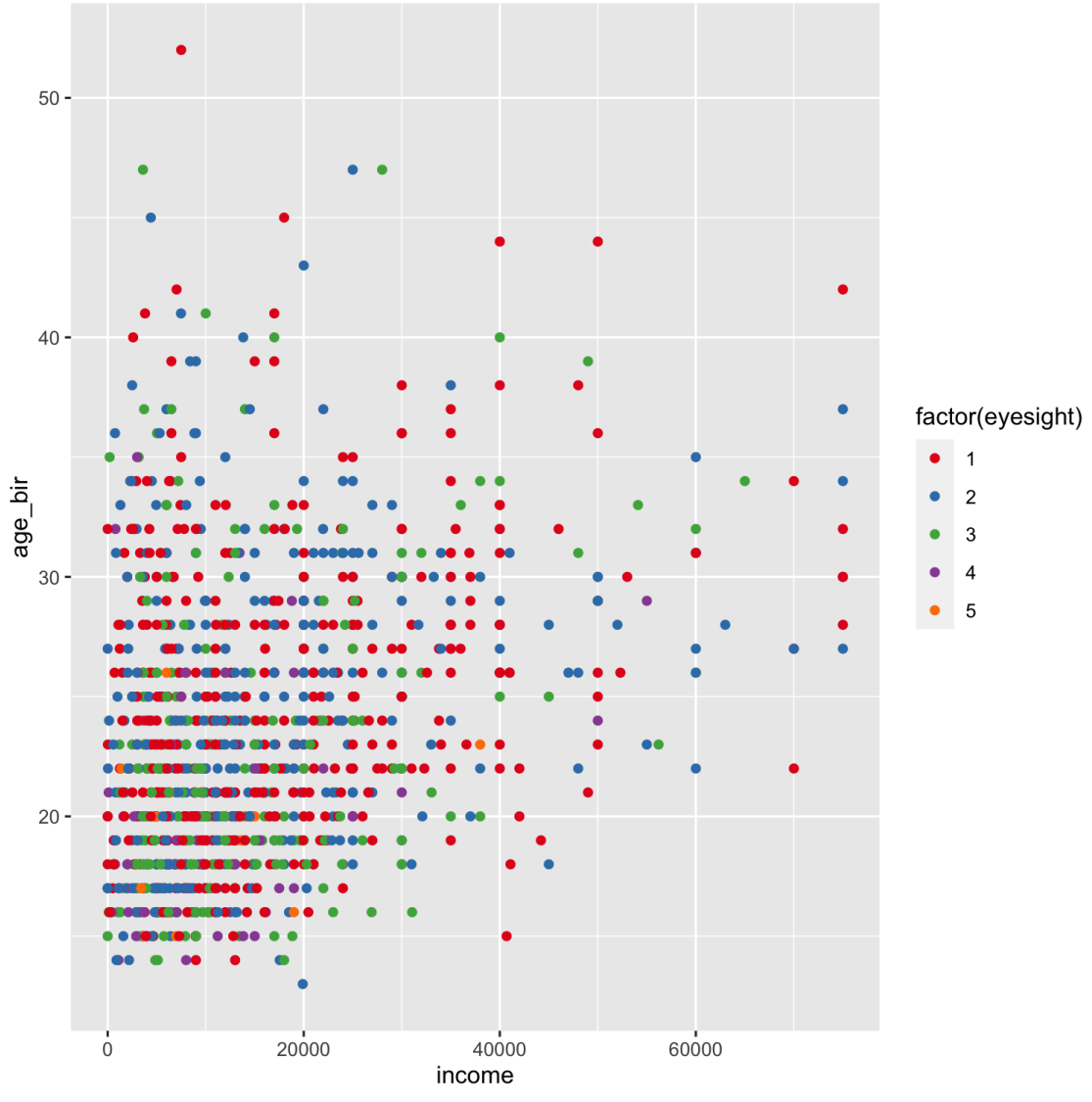


```
ggplot(data = nlsy) +
geom_point(aes(x = income, y = age_bir,
               color = factor(eyesight))) +
scale_color_brewer(palette = "Set1")
```

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

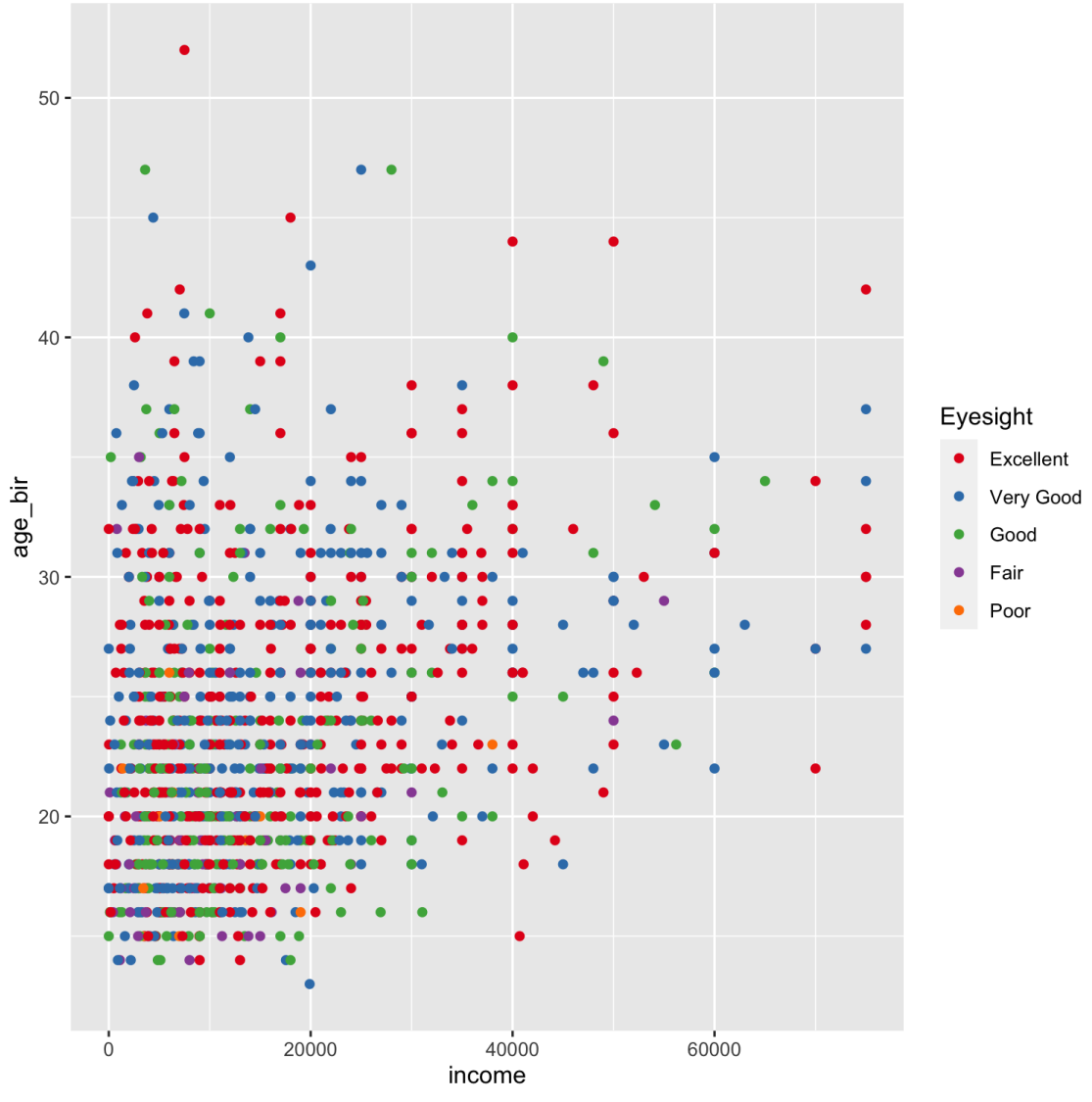
- You can access the scales in that package with `scale_color_brewer()`, or see them all after installing the package with `RColorBrewer::display.brewer.all()`



```
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_x()` 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!



Your turn...

1

Exercises 2.1: Make a fancy scatterplot showing the relationship between sleep on weekdays and on weekends.

Facets

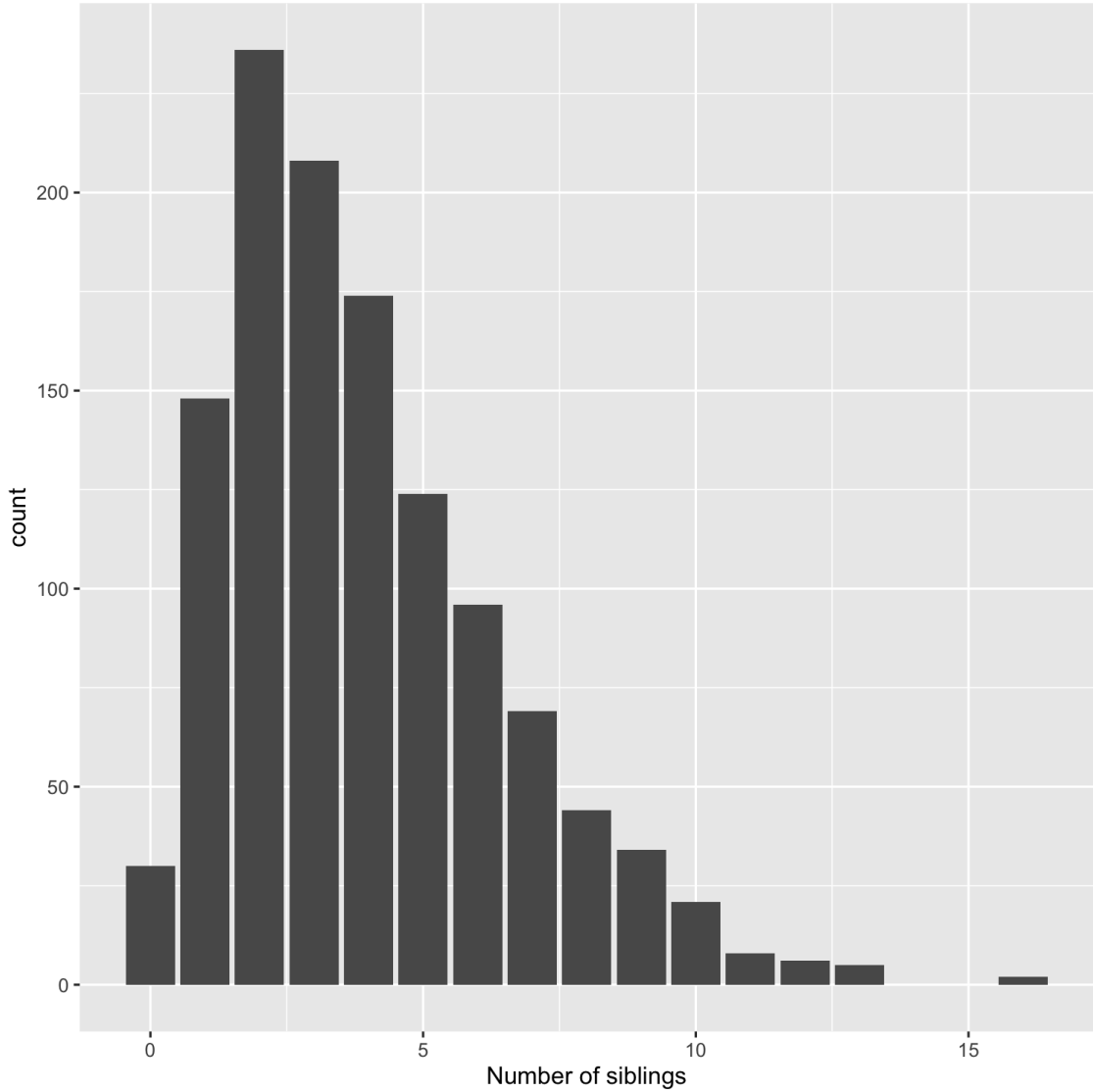
One of the most useful features of `ggplot2` is the ability to "facet" a graph by splitting it up according to the values of some variable.

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

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

We'll introduce bar graphs at the same time!

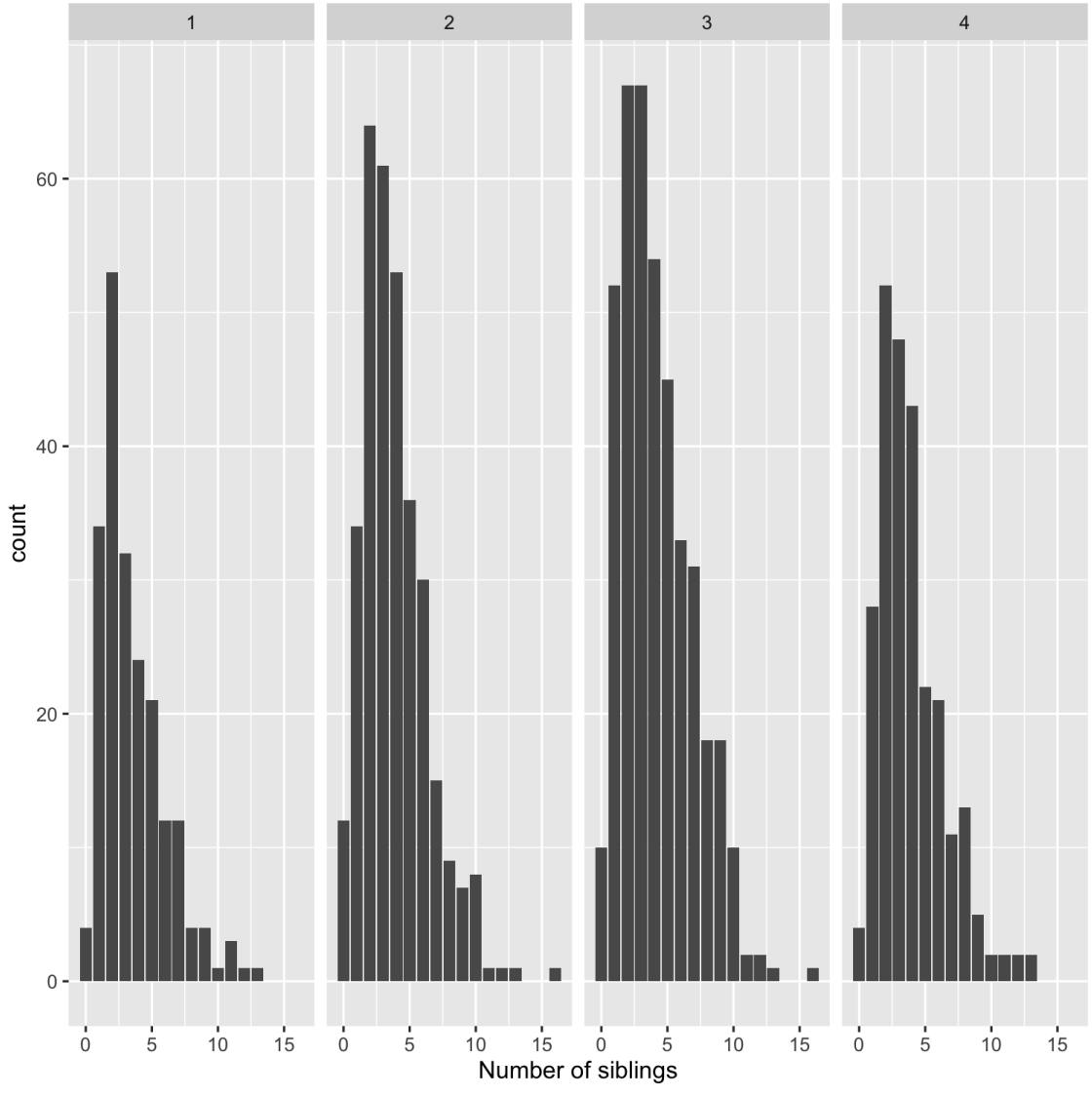
Notice how we only need an `x =` argument - the y-axis is automatically the count with this geom.



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

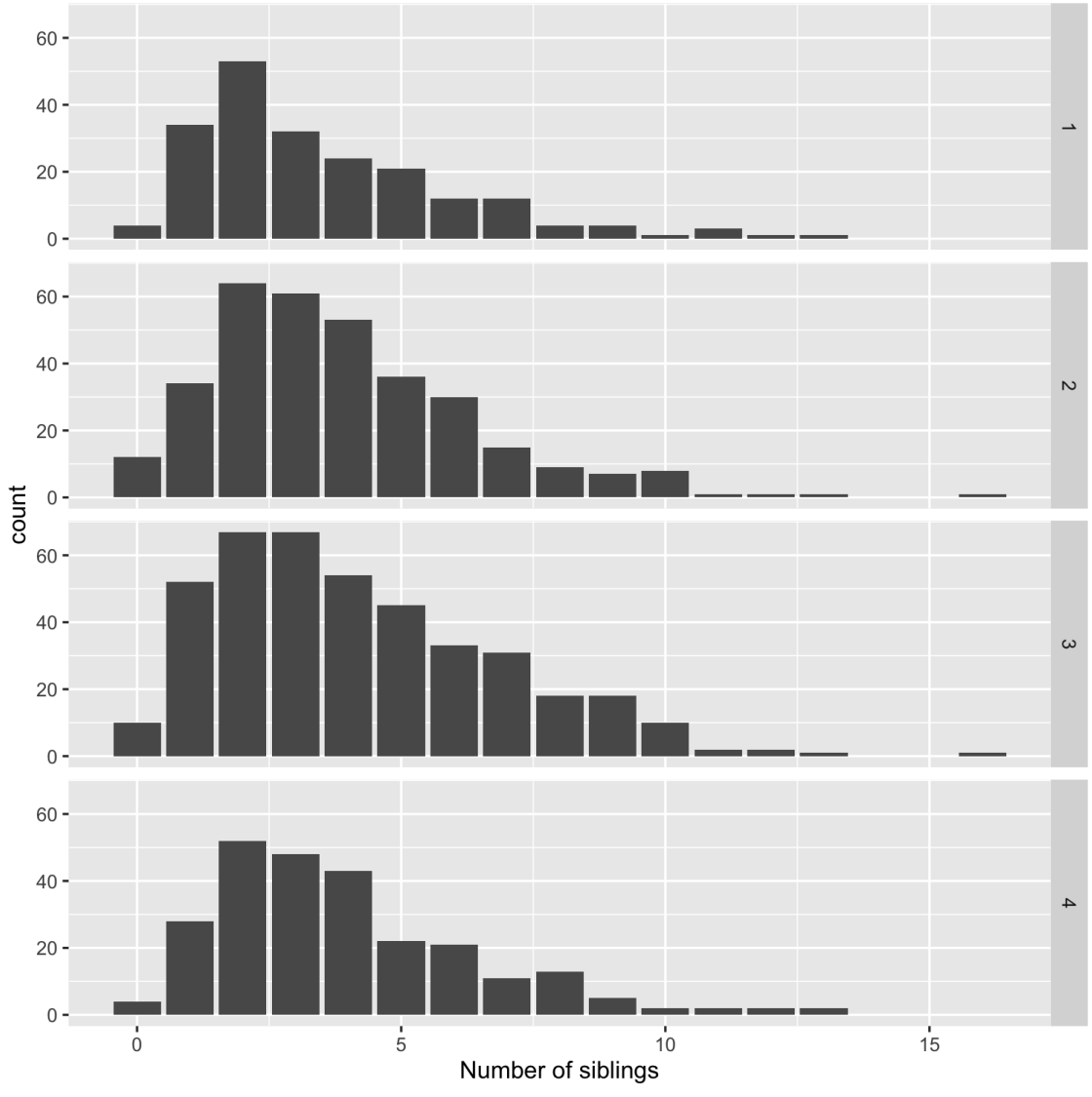
The `facet_grid()` function splits up the data according to a variable(s).

Here we've split it by region into columns.



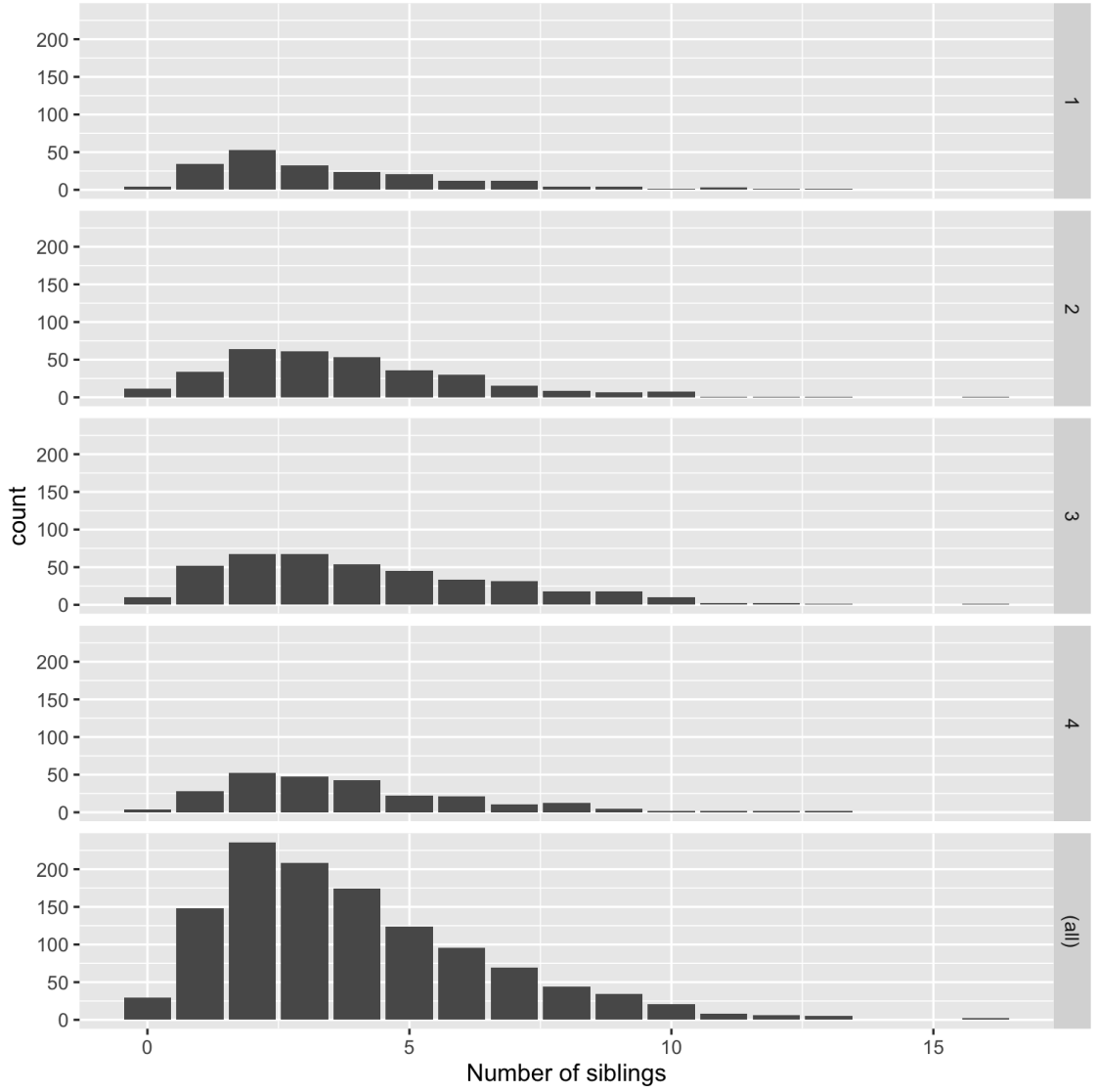
```
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_grid(rows = vars(region))
```

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



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

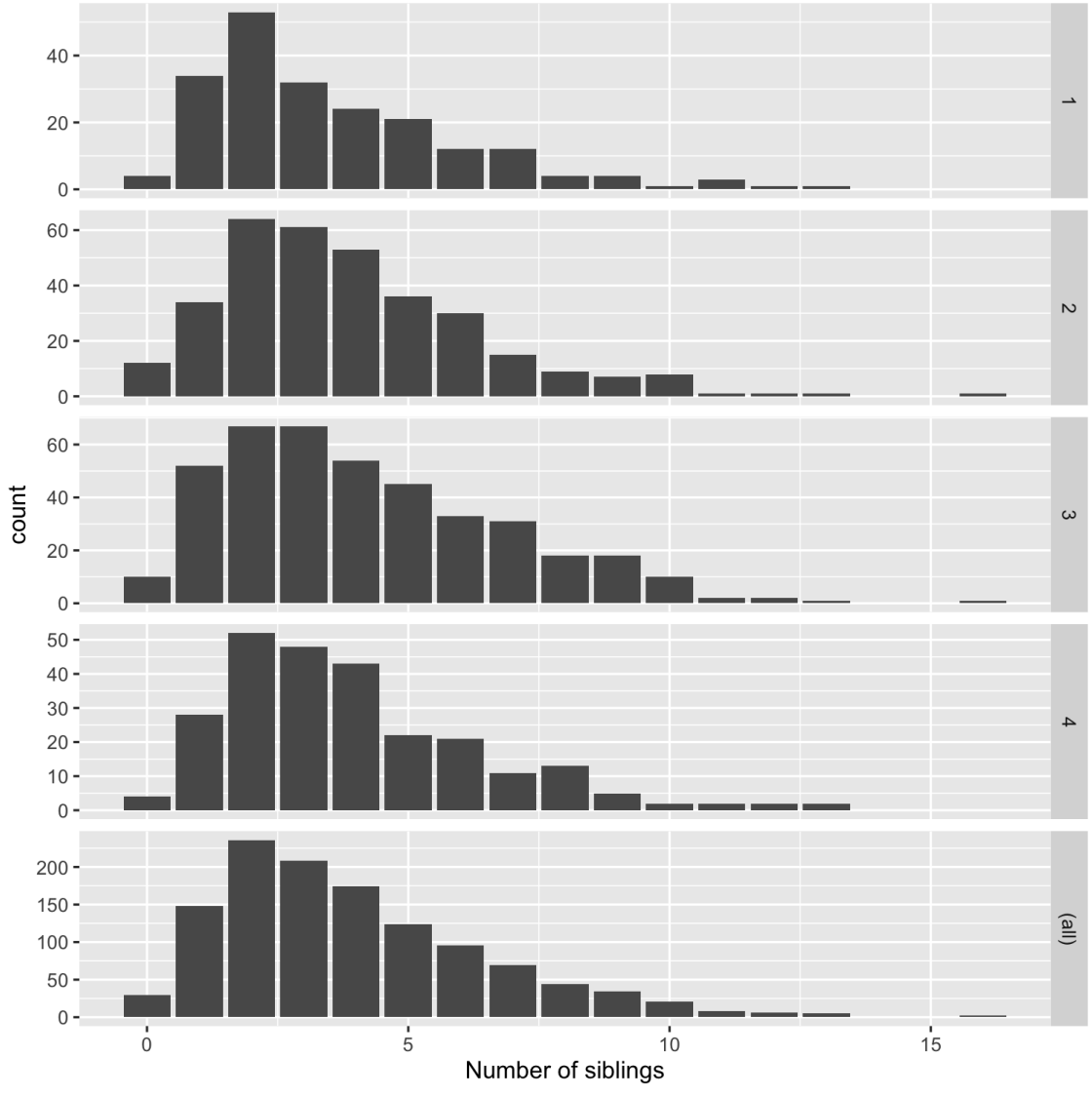
We can also add a row for all of the data together.



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

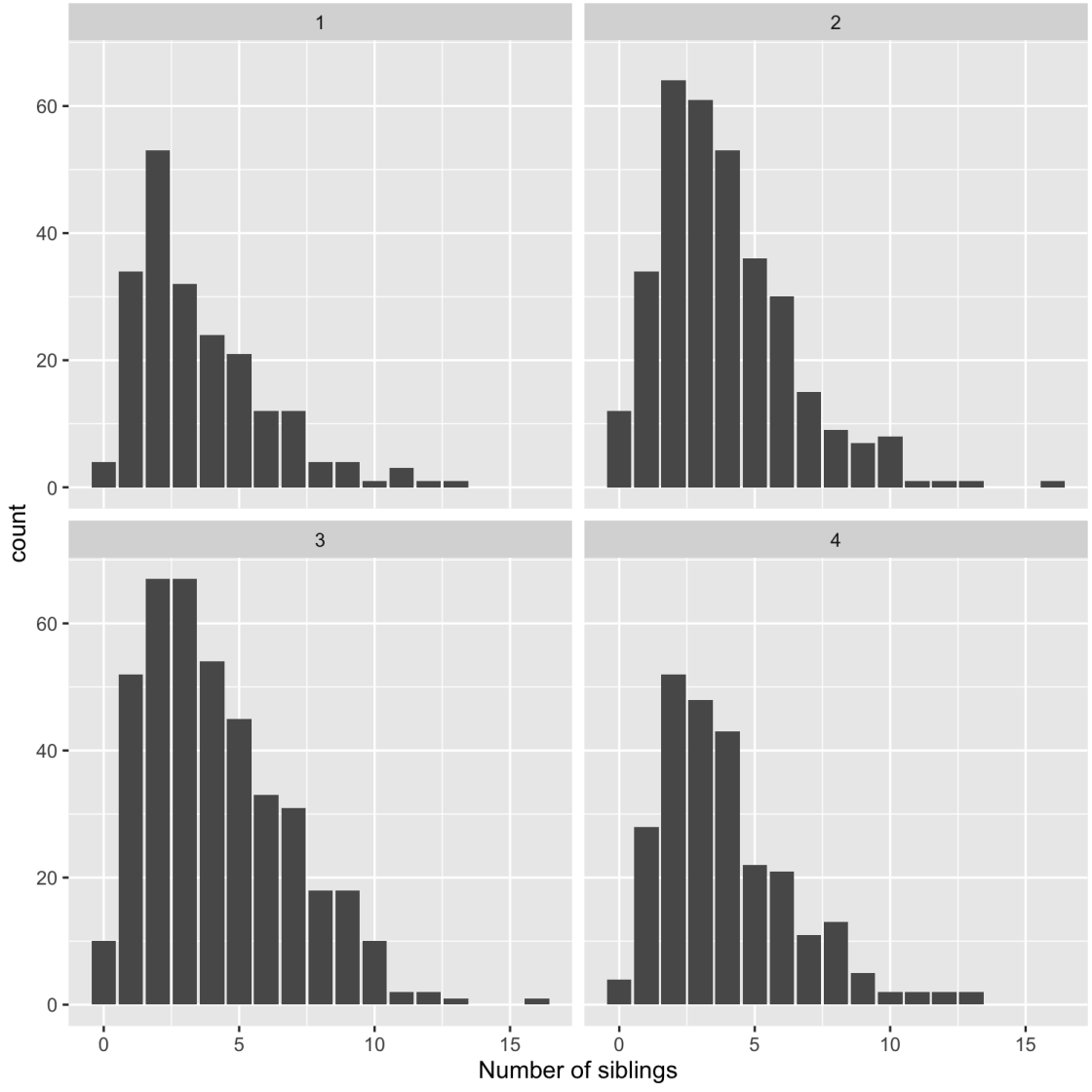
This squishes the other rows though! We can allow them all to have their own axis limits with the `scales =` argument.

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



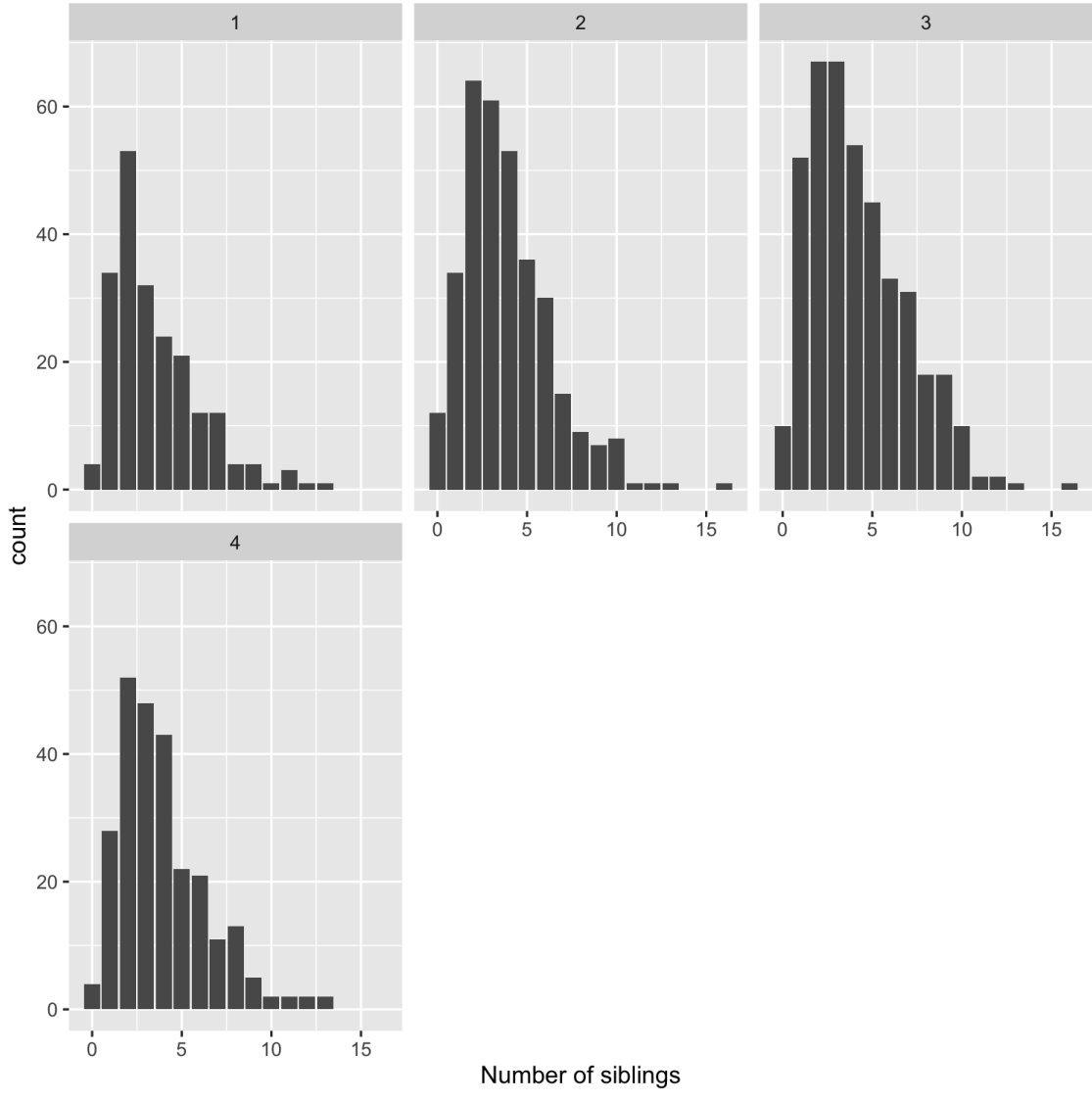
```
ggplot(data = nlsy) +
  geom_bar(aes(x = nsibs)) +
  labs(x = "Number of siblings") +
  facet_wrap(vars(region))
```

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



```
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))
```

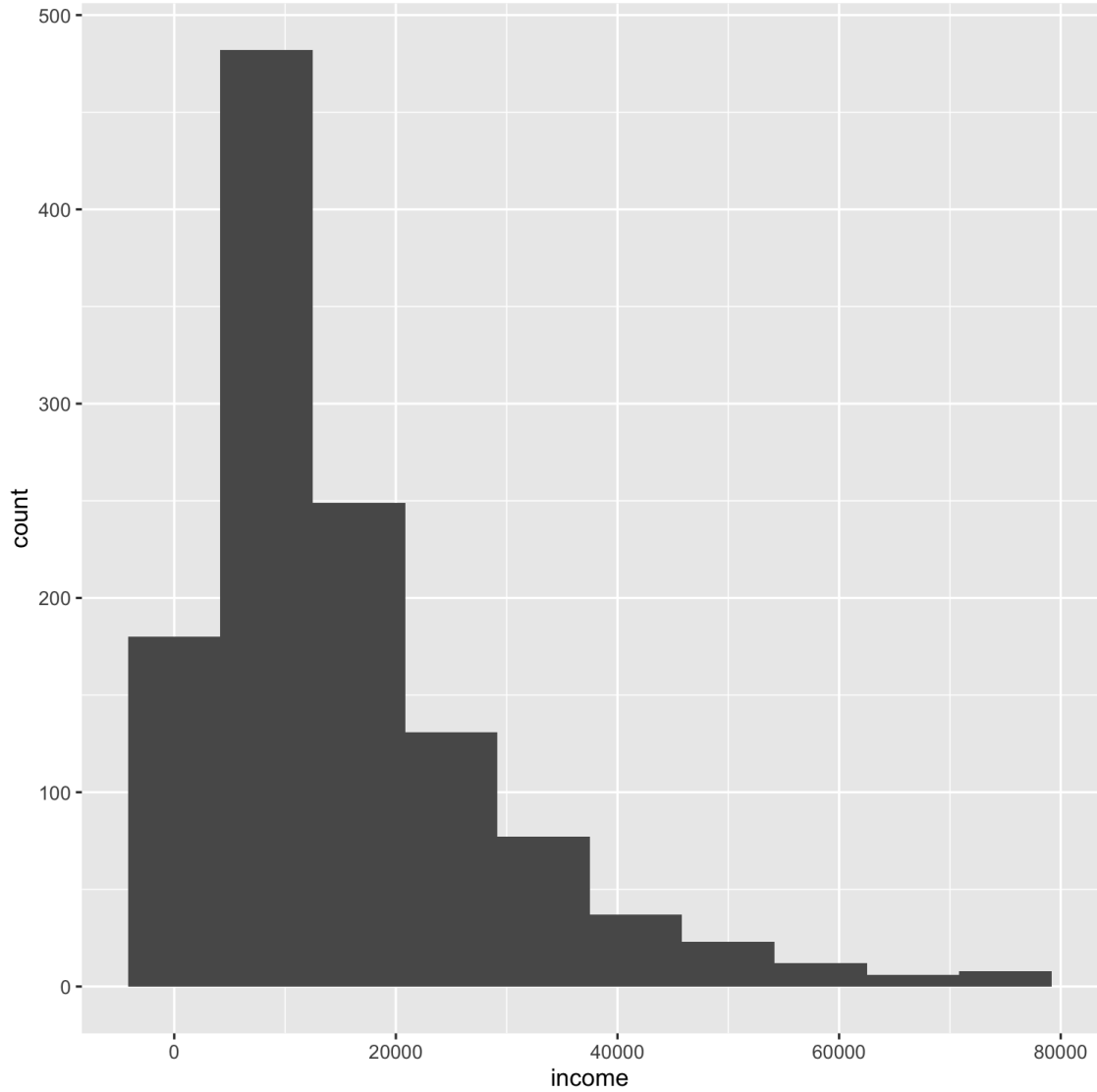
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

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!

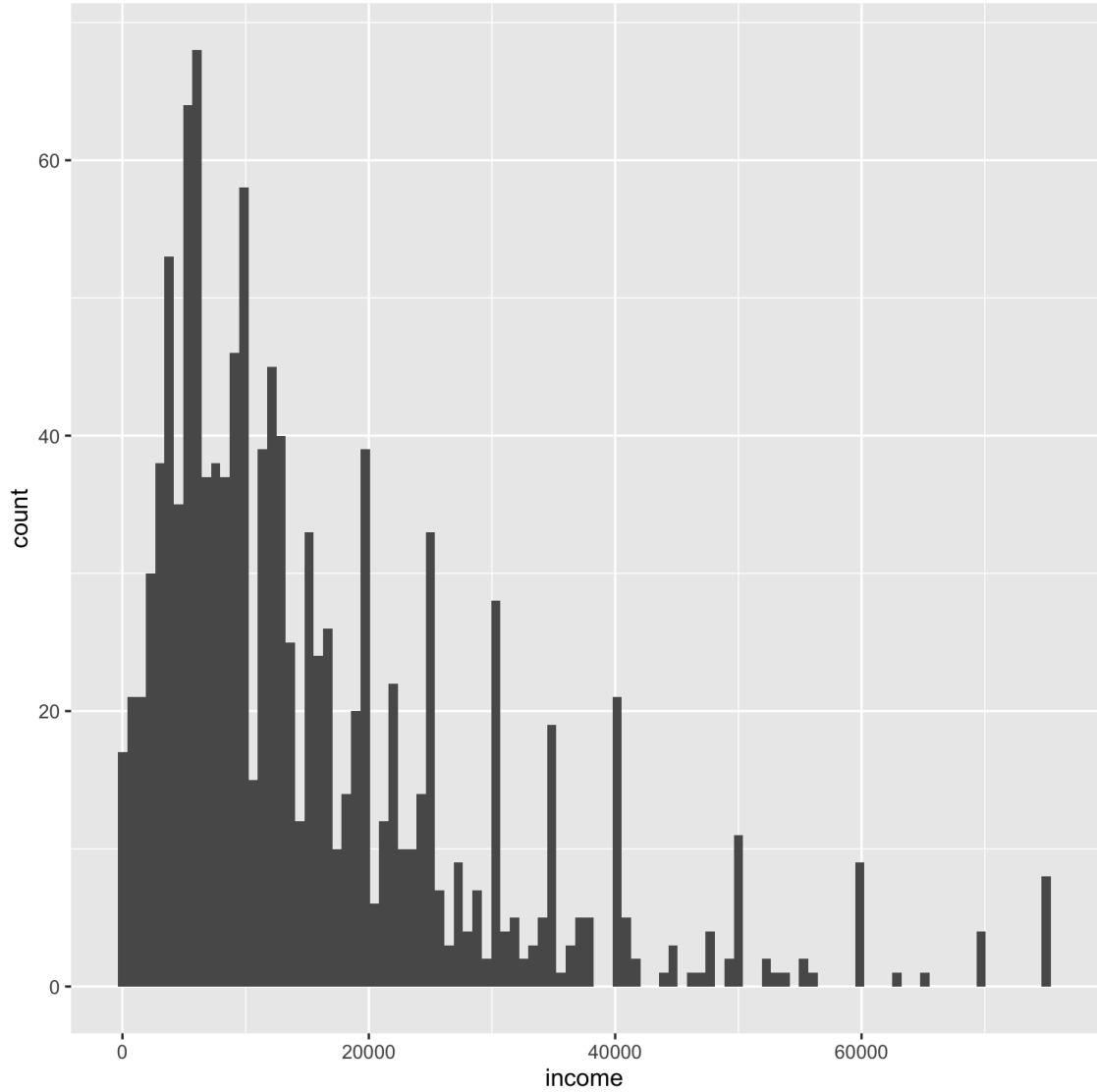
```
ggplot(data = nlsy) +
  geom_histogram(aes(x = income),
                 bins = 10)
```

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



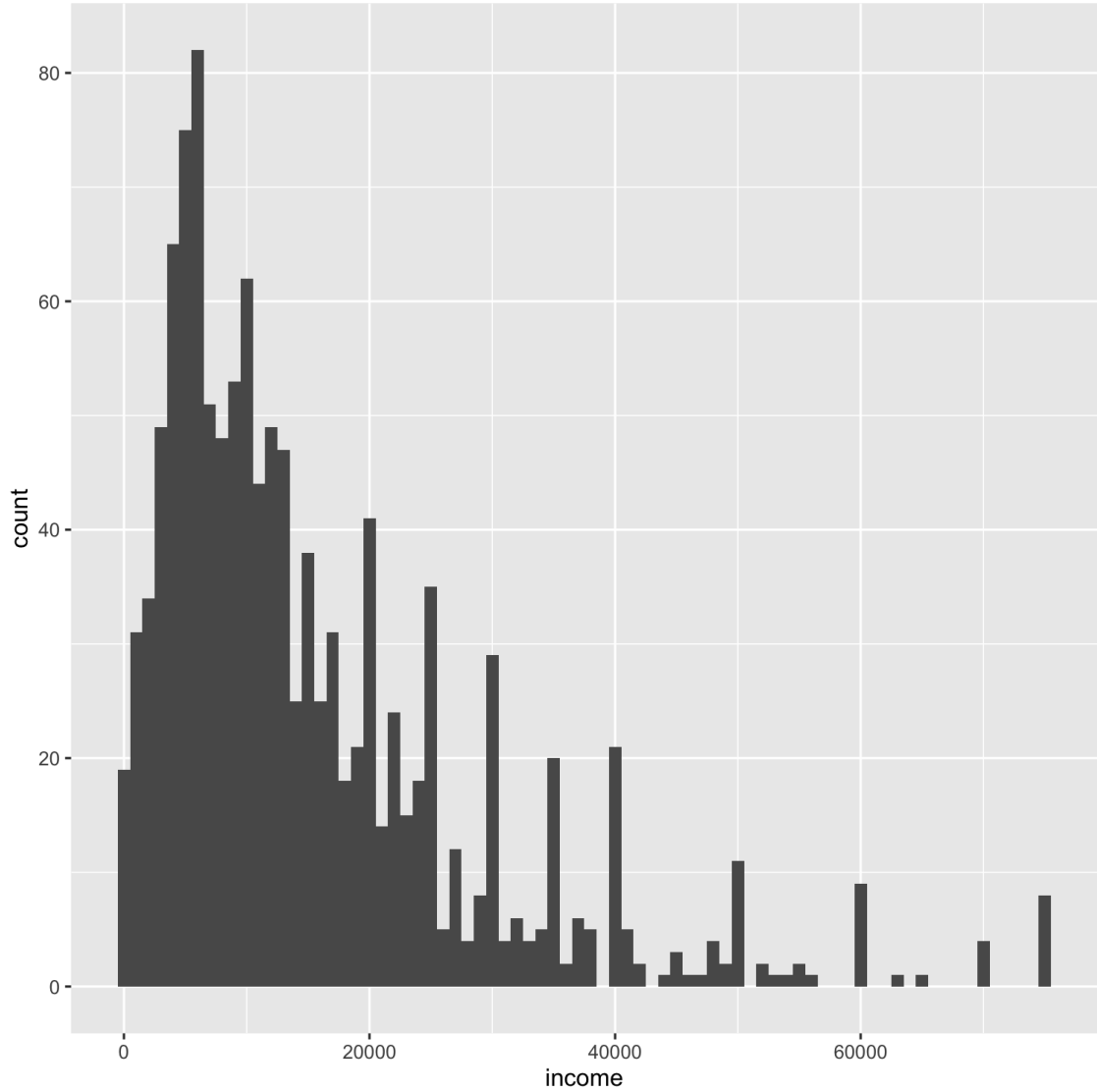
```r
ggplot(data = nlsy) +
  geom_histogram(aes(x = income),
                 bins = 100)
```

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



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

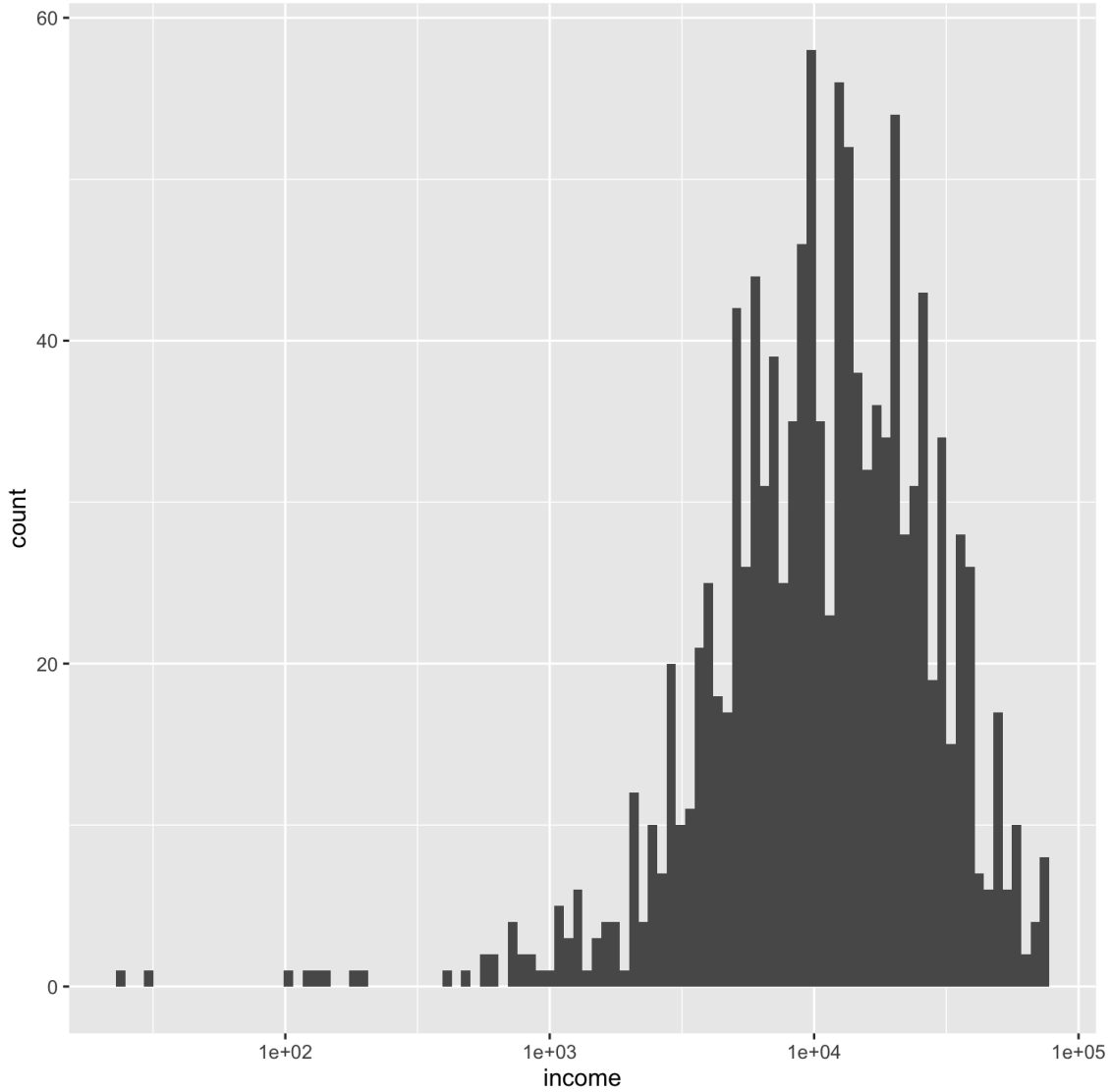
> Sometimes the bin width actually has some meaning



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

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

> The naming schemes work similarly to the `scale_color` ones, just with different options!



Your turn...

2

Exercises 2.2: Make a fancy histogram showing the distribution of income in this data.

Finally, themes to make our plots prettier

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

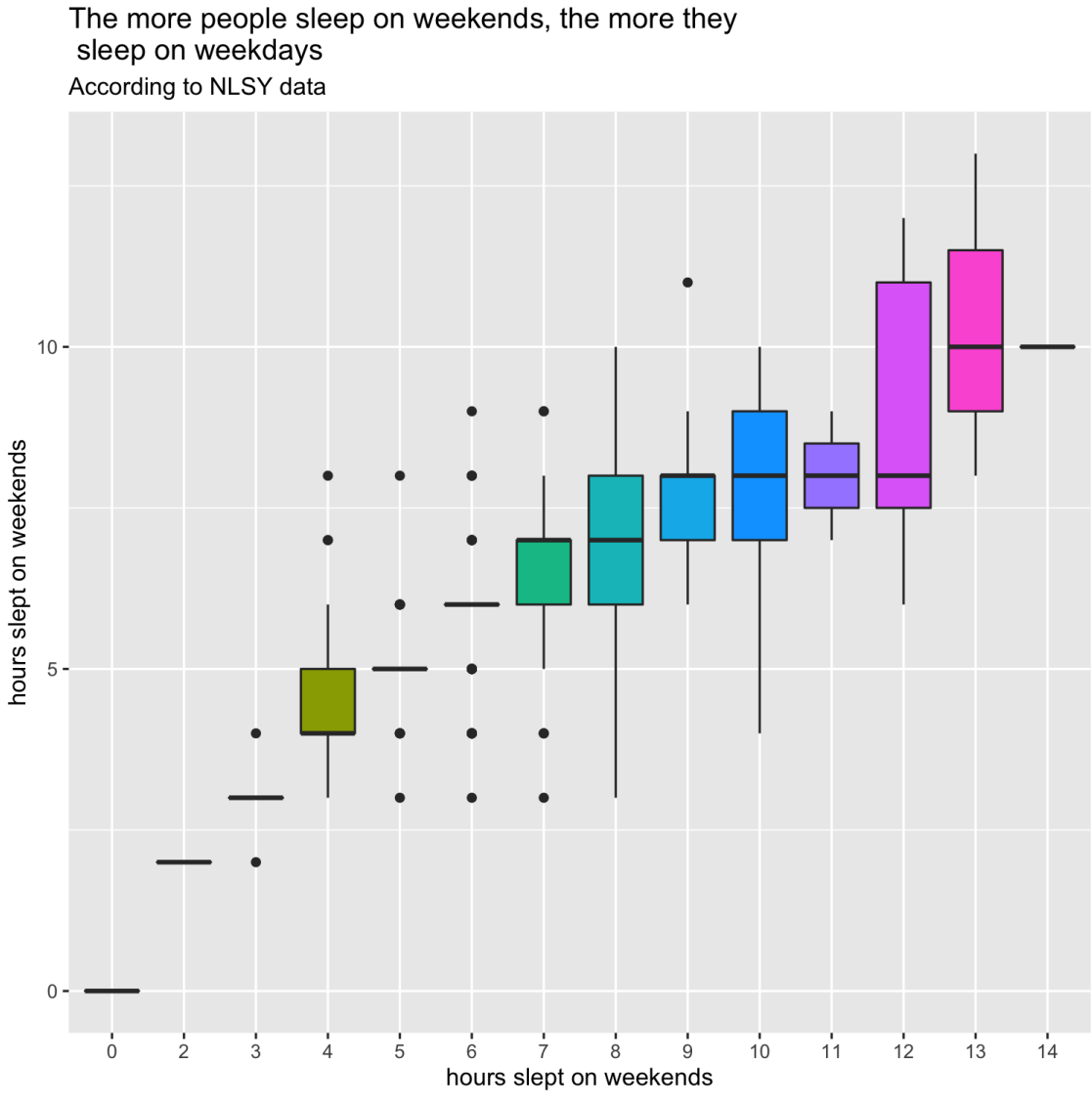
```r
p <- ggplot(data = nlsy) +
  geom_boxplot(aes(
    x = factor(sleep_wknd),
    y = sleep_wkdy,
    fill = factor(sleep_wknd))) +
  scale_fill_discrete(guide = FALSE) +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on weekend
  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

p
```

Let's store our plot first.

Plots work just like other R objects, meaning we can use the assignment arrow.

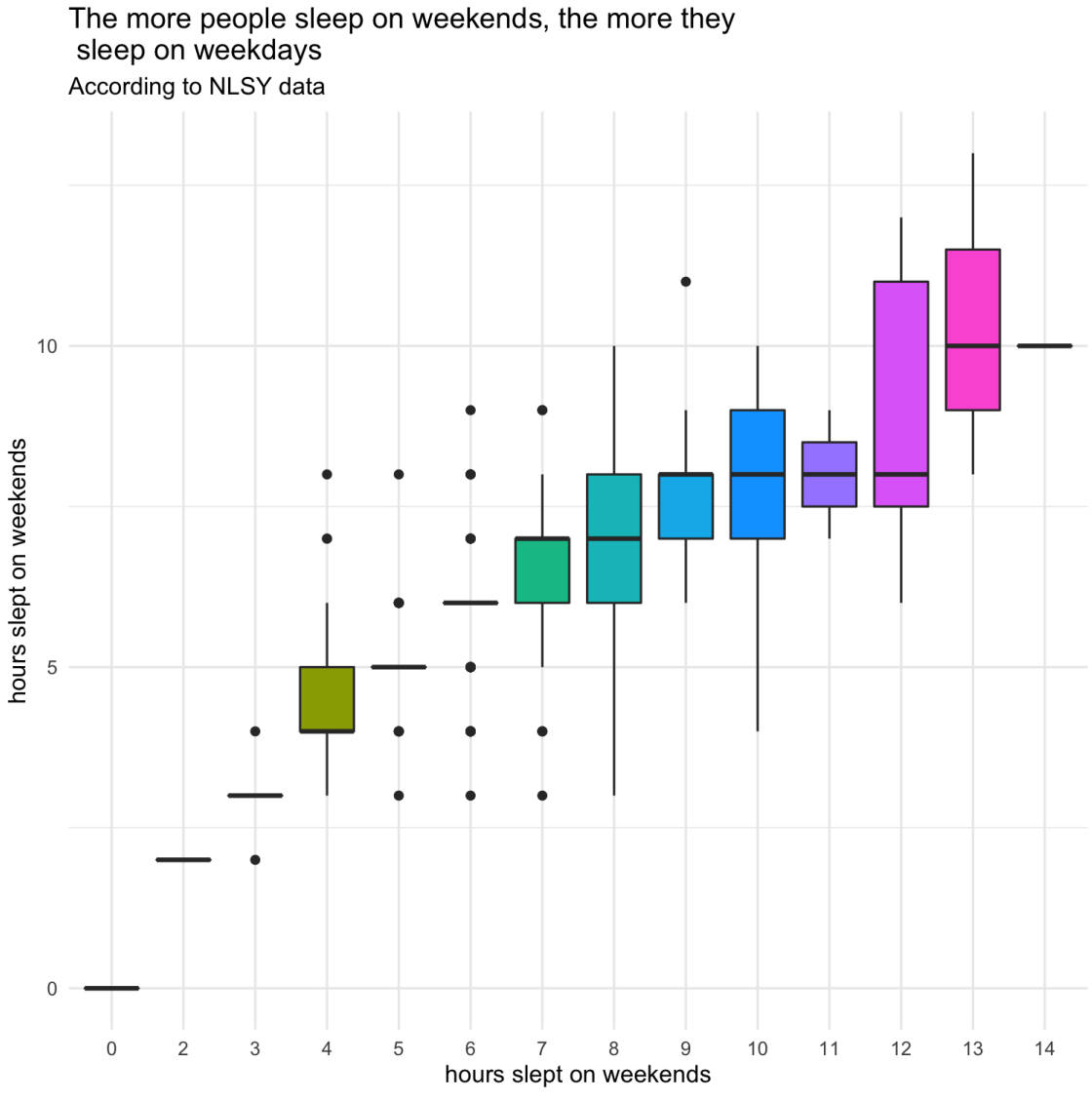
> Can you figure out what each chunk of this code is doing to the figure?



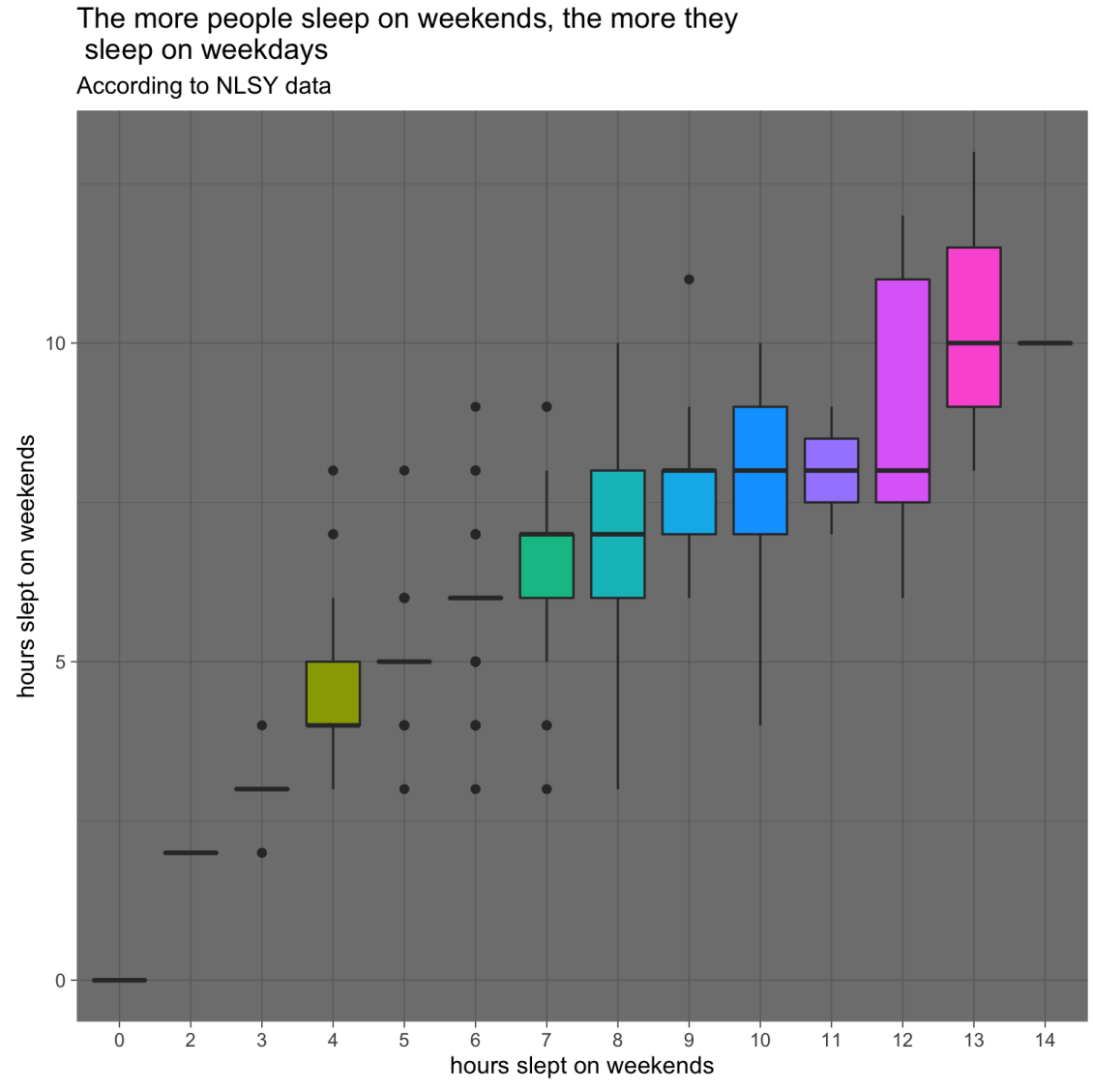
```
p +
  theme_minimal()
```

We can change the overall theme

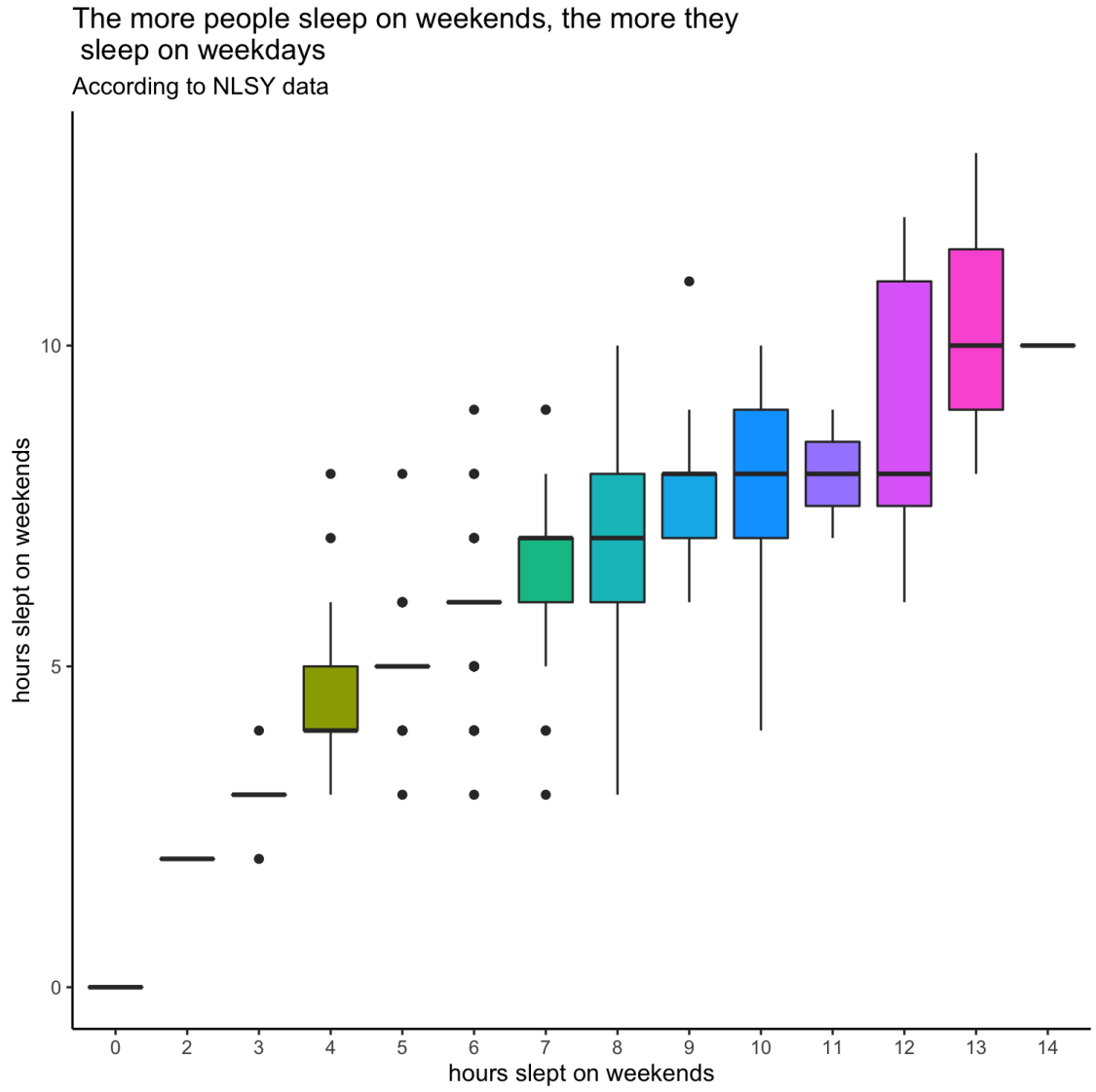
> Since we stored the plot as `p`, it's easy to add on / try different things



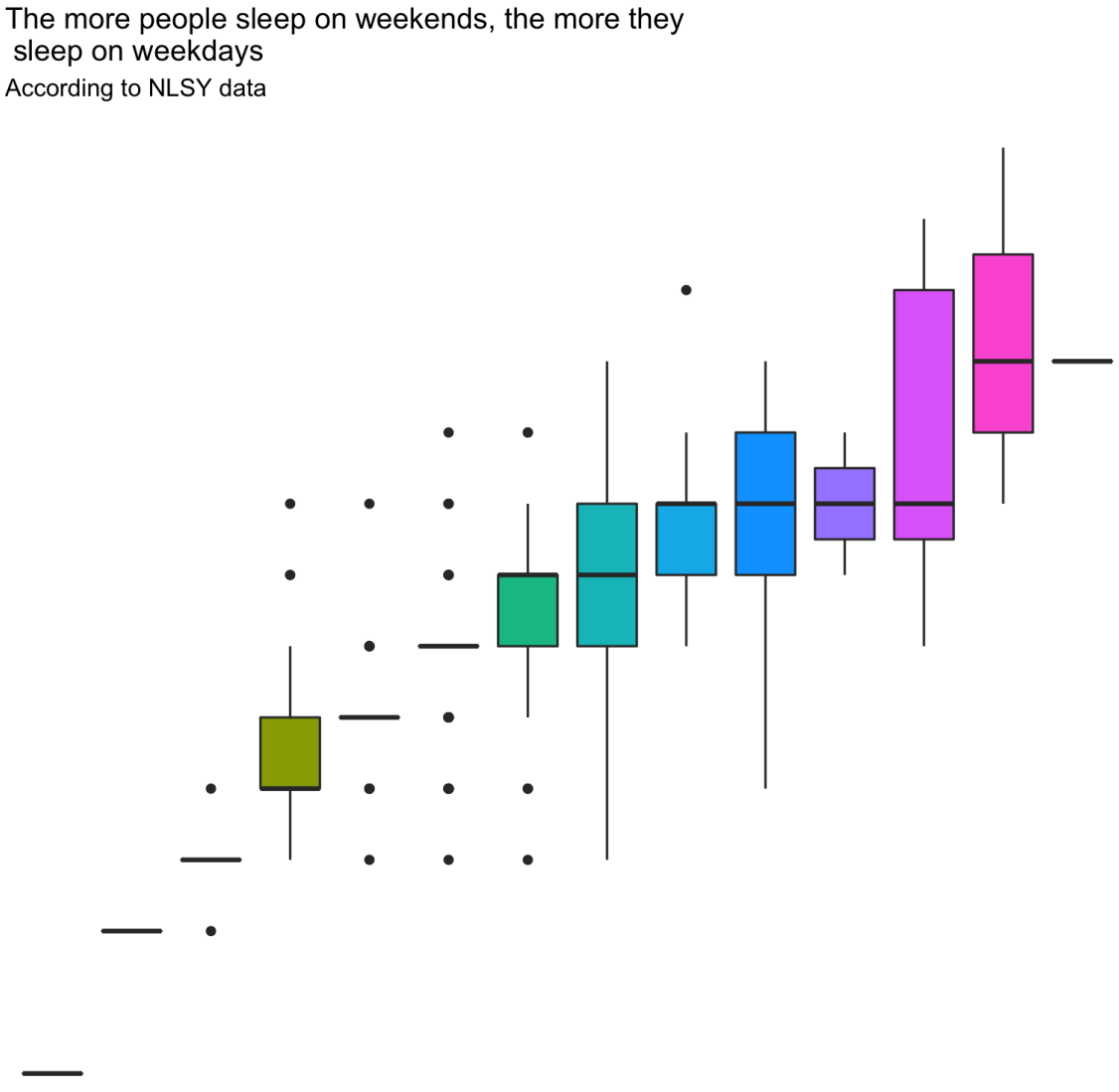
```
p +
  theme_dark()
```


The more people sleep on weekends, the more they sleep on weekdays
According to NLSY data
hours slept on weekends
0
5
10
0
2
3
4
5
6
7
8
9
10
11
12
13
14
hours slept on weekends

```
p +
  theme_classic()
```

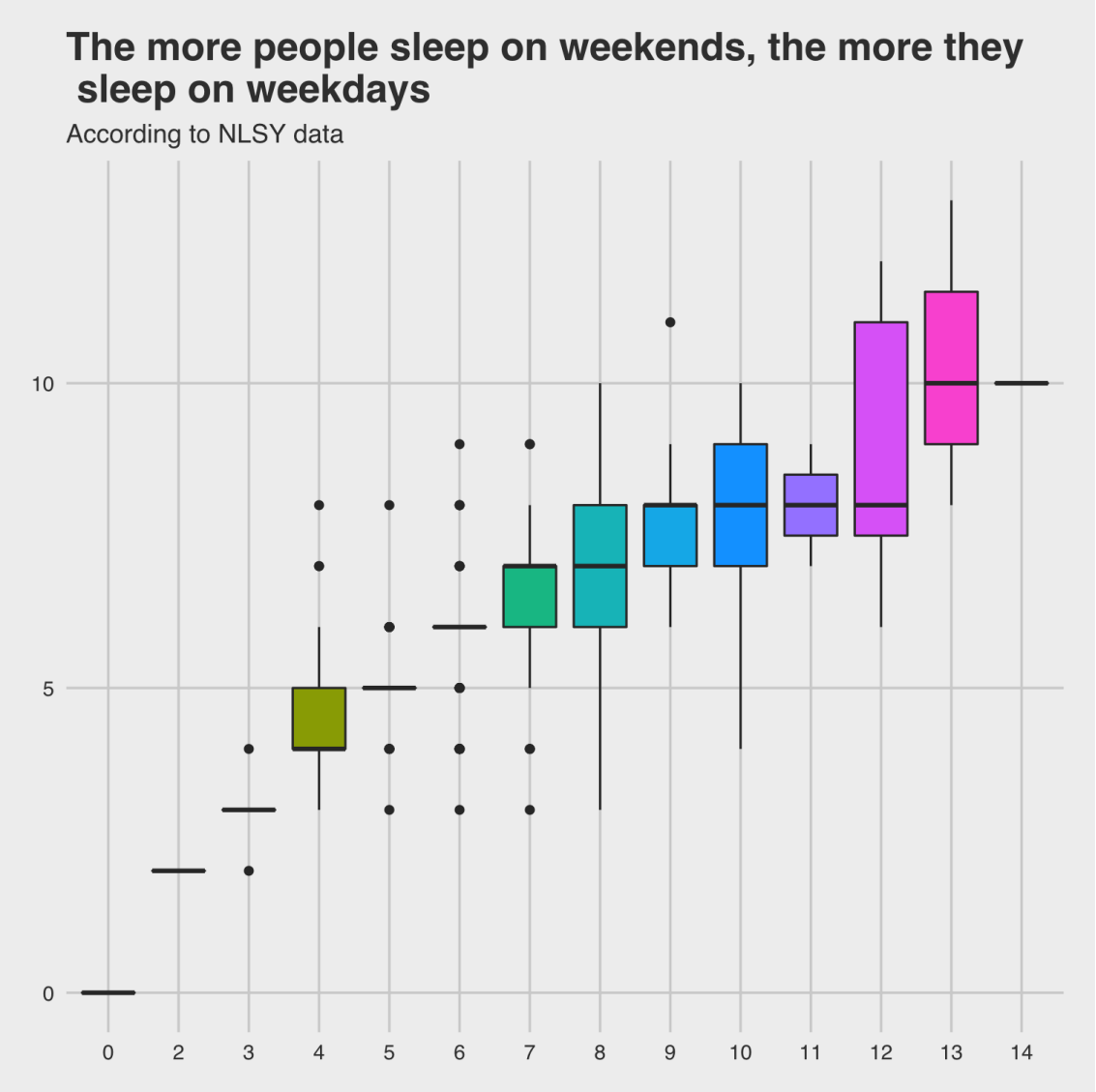


```
p +
  theme_void()
```


The more people sleep on weekends, the more they sleep on weekdays
According to NLSY data

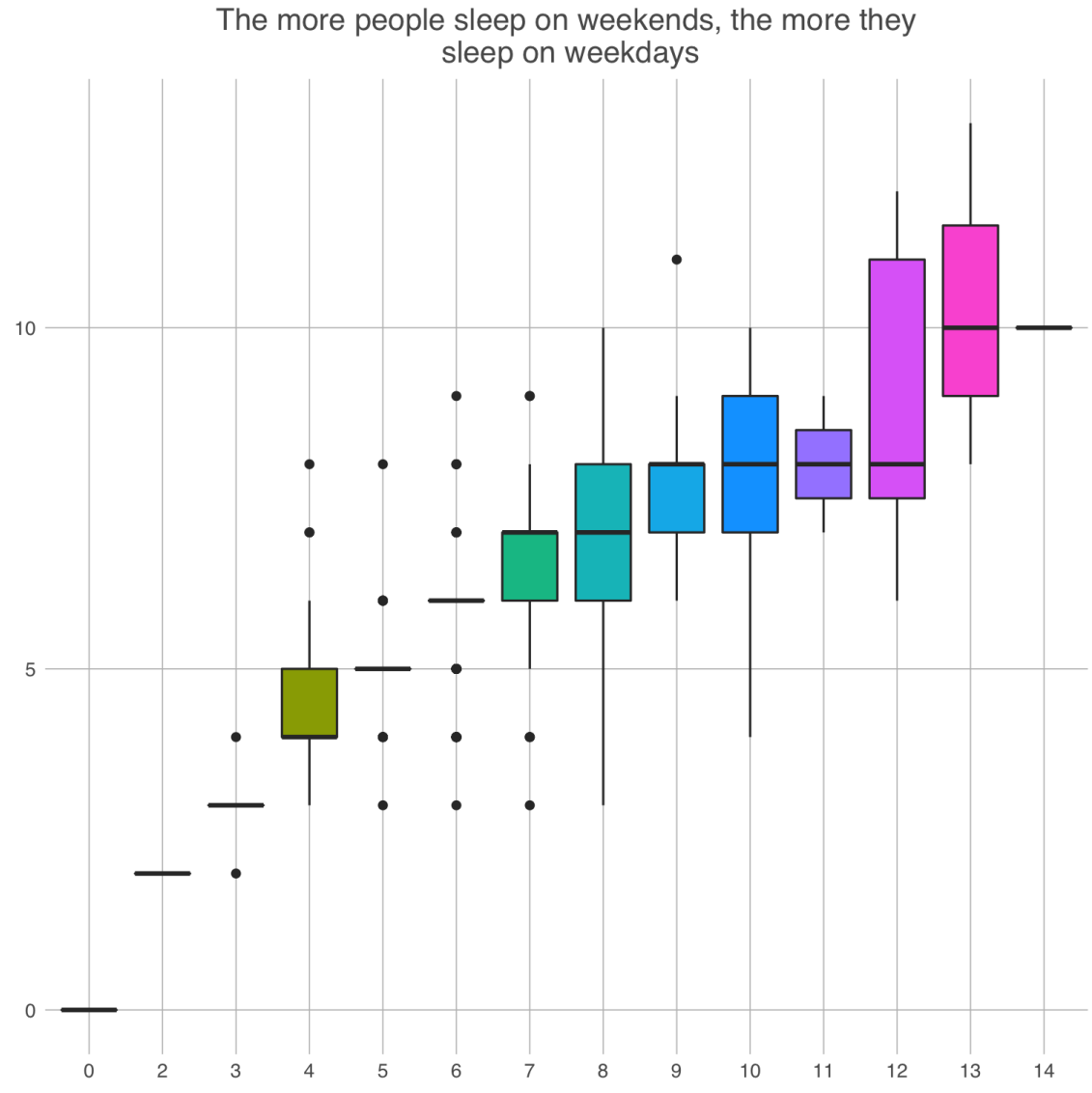
```
p +
  ggthemes::theme_fivethirtyeight()
```

Other packages may contain themes.

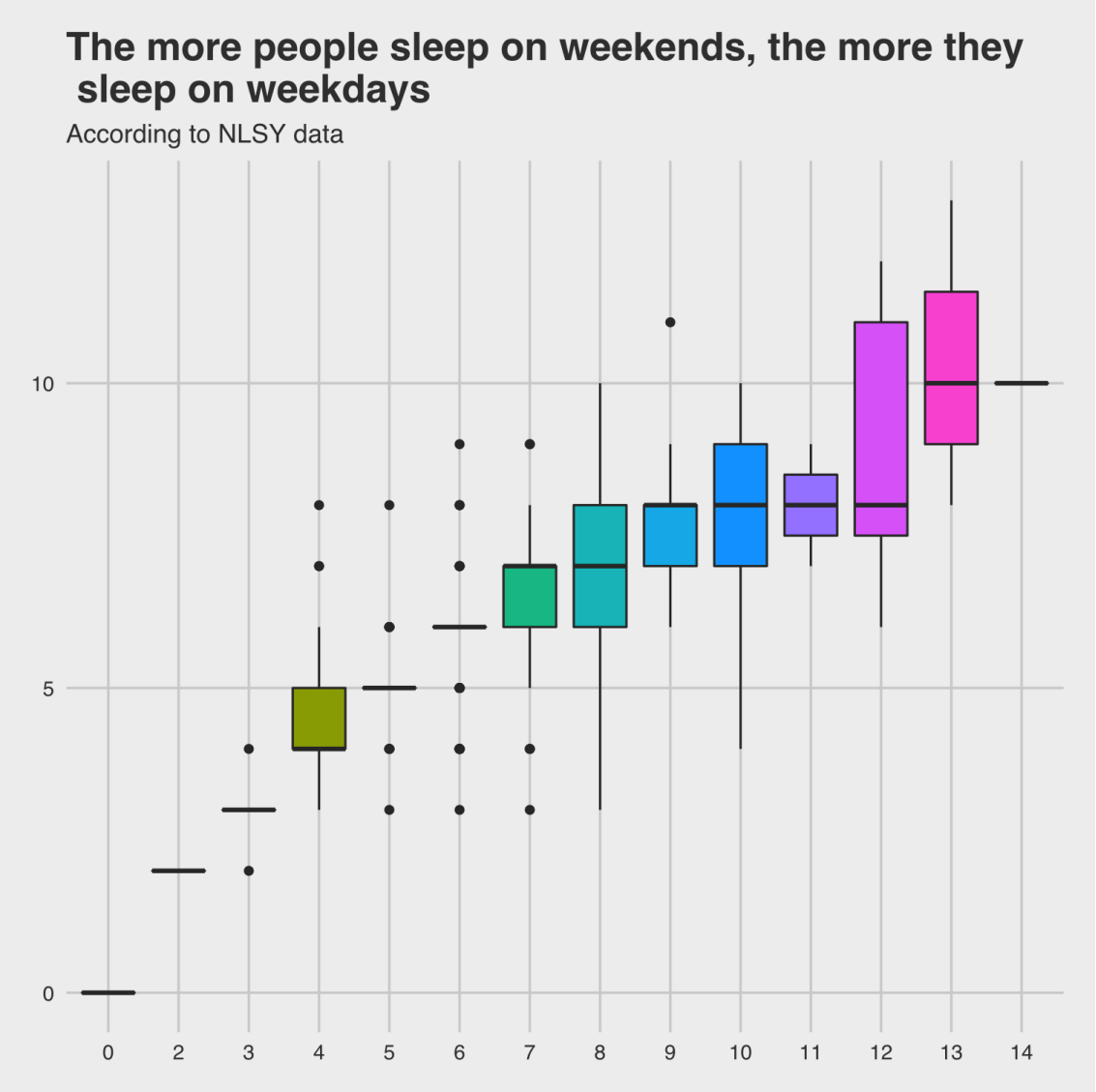


```
p +
  ggthemes::theme_excel_new()
```

In case you miss Excel....

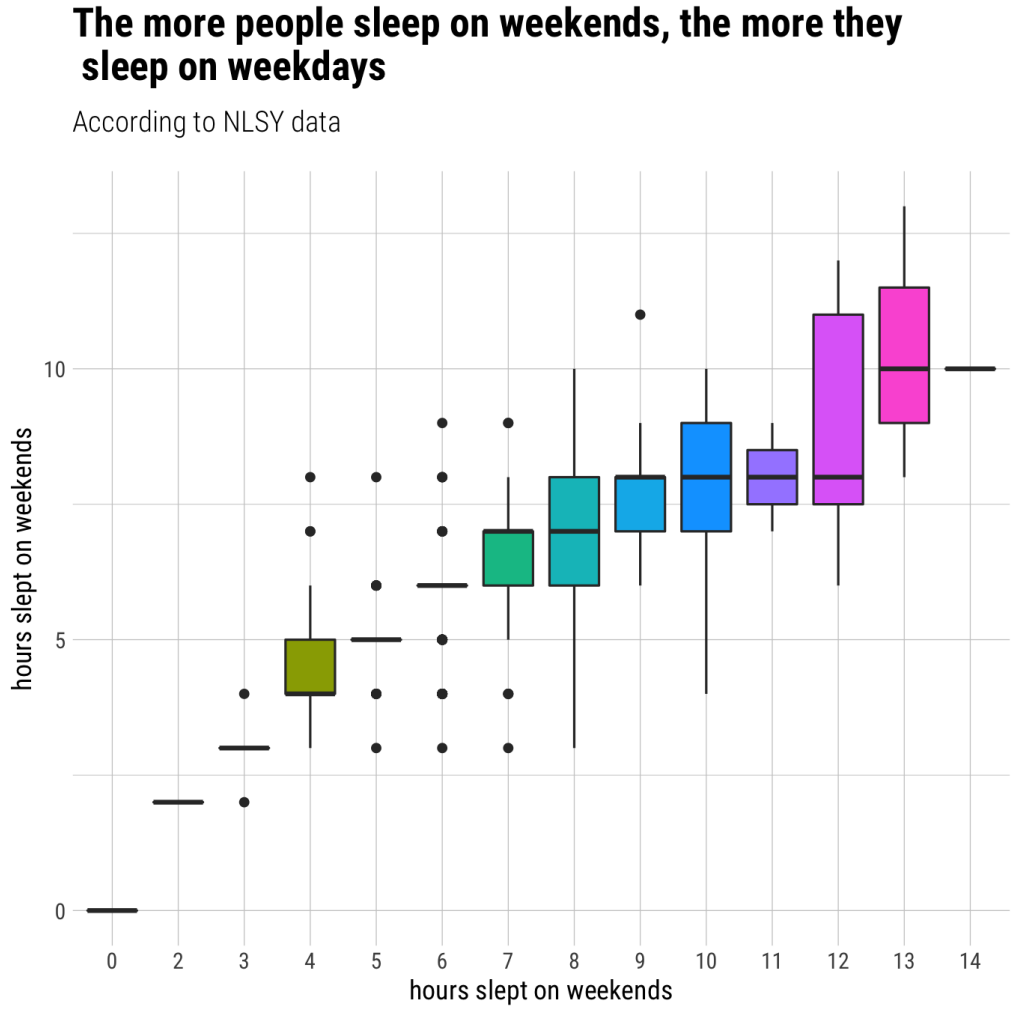


```
p +
  ggthemes::theme_gdocs()
```



```
p +
  louisahstuff::my_theme()
```

You can even make your own!



Finally, save it!

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

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

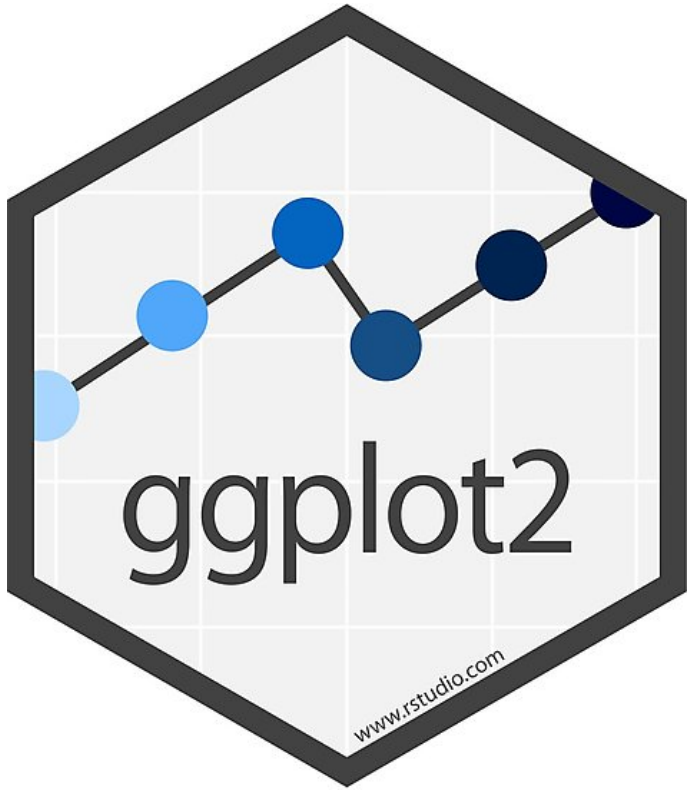
The `ggsave()` function will automatically save the most recent plot in your output.

To be safe, you can store your plot, e.g., `p <- ggplot(...) + ...` and then

```
ggsave(filename = "scatterplot.pdf", plot = p)
```

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
- List of themes and instructions to make your own: https://www.datanovia.com/en/blog/ggplot-themes-gallery/



Your turn...

3

Exercises 2.3: Recreate this plot!