ID 543: Day 4

Introduction to R

Homework review

Today's goals

- Make a simple frequency tables
- Make a more complex "Table 1"
- Run regressions in R and show the results in a table
- Introduce the framework of `ggplot2` and make some simple figures

Tables

We saw that we could compute summary statistics using `summarize()`:

```
nlsy |>
  group_by(sex_cat) |>
  summarize(prop_glasses = mean(glasses),
            mean_age_bir = mean(age_bir),
            n_vg_eye = sum(eyesight_cat == "Very Good"),
            prop_vg_eye = mean(eyesight_cat == "Very Good"),
            n_g_eye = sum(eyesight_cat == "Good"),
            prop_g_eye = mean(eyesight_cat == "Good")
  )
```

```
# A tibble: 2 × 7
  sex_cat prop_glasses mean_age_bir n_vg_eye prop_vg_eye n_g_eye prop_g_eye
  <fct>          <dbl>        <dbl>    <int>       <dbl>   <int>      <dbl>
1 Male           0.441         25.1      162       0.323      85      0.170
2 Female         0.572         22.2      223       0.317     164      0.233
```

Easier way to make frequency tables

The `{janitor}` package has nice functionality to create tables:

```r
# install.packages("janitor")
library(janitor)
tabyl(nlsy, sex_cat)
```

```
sex_cat   n   percent
   Male 501 0.4157676
 Female 704 0.5842324
```

Tip

Just like many of our other functions, it takes a dataset as its first argument so can pipe

Cross-tabulations with `tabyl()`

It's easy to make a 2 x 2 (or n x m) table with `tabyl()`

```
tabyl(nlsy, sex_cat, eyesight_cat)
```

```
sex_cat Excellent Very Good Good Fair Poor
   Male       228       162   85   21    5
 Female       246       223  164   57   14
```

`tabyl()`

We might want to see the percentages instead of counts

```r
nlsy |>
  tabyl(sex_cat, glasses_cat) |>
  adorn_percentages() |>
  adorn_pct_formatting()
```

```
sex_cat Doesn't wear glasses Wears glasses/contacts
   Male                55.9%                  44.1%
 Female                42.8%                  57.2%
```

Tip

By default, `adorn_percentages()` will give you row percentages (sums to 100% across the row)

Other percentages

```r
nlsy |>
    tabyl(sex_cat, glasses_cat) |>
    adorn_percentages("col") |>
    adorn_pct_formatting()
```

```
sex_cat Doesn't wear glasses Wears glasses/contacts
   Male                48.2%                  35.4%
 Female                51.8%                  64.6%
```

```r
nlsy |>
    tabyl(sex_cat, glasses_cat) |>
    adorn_percentages("all") |>
    adorn_pct_formatting()
```

```
sex_cat Doesn't wear glasses Wears glasses/contacts
   Male                23.2%                  18.3%
 Female                25.0%                  33.4%
```

`tabyl()`

These are just dataframes, so we can save as csv, etc.

```r
two_by_two <- nlsy |>
    tabyl(sex_cat, glasses_cat)

write_csv(two_by_two,
          here::here("results", "sex-eyesight-table.csv"))
```

Warning

If you don't have a "results" folder in your project, that code won't work until you make one!

We can easily do a chi-squared test

```r
tabyl(nlsy, sex_cat, eyesight_cat) |>
  chisq.test()
```

```

	Pearson's Chi-squared test

data:  tabyl(nlsy, sex_cat, eyesight_cat)
X-squared = 22.738, df = 4, p-value = 0.0001428
```

Tip

There are a lot of other helpful functions in the `janitor` package. My favorite is `clean_names()`. Check out the documentation for more!

Exercise

Making more complex tables

There are lots of packages for making tables in R

One of my favorites is `{gtsummary}`

```r
# install.packages("gtsummary")
library(gtsummary)
```

`gtsummary::tbl_summary()`

```r
tbl_summary(
  nlsy,
  by = sex_cat,
  include = c(race_eth_cat,
              eyesight_cat,
              glasses,
              age_bir))
```

Characteristic	**Male**, N = 501[1]	**Female**, N = 704[1]
race_eth_cat		
Hispanic	81 (16%)	130 (18%)
Black	138 (28%)	169 (24%)
Non-Black, Non-Hispanic	282 (56%)	405 (58%)
eyesight_cat		
Excellent	228 (46%)	246 (35%)
Very Good	162 (32%)	223 (32%)
Good	85 (17%)	164 (23%)
Fair	21 (4.2%)	57 (8.1%)
Poor	5 (1.0%)	14 (2.0%)
glasses	221 (44%)	403 (57%)
age_bir	24.0 (21.0, 29.0)	21.0 (18.0, 26.0)

[1] n (%); Median (IQR)

```r
tbl_summary(
  nlsy,
  by = sex_cat,
  include = c(race_eth_cat, eyesight_cat,
              glasses, age_bir),
  label = list(
    race_eth_cat ~ "Race/ethnicity",
    eyesight_cat ~ "Eyesight",
    glasses ~ "Wears glasses",
    age_bir ~ "Age at first birth"
  ))
```

Characteristic	**Male**, N = 501[1]	**Female**, N = 704[1]
Race/ethnicity		
Hispanic	81 (16%)	130 (18%)
Black	138 (28%)	169 (24%)
Non-Black, Non-Hispanic	282 (56%)	405 (58%)
Eyesight		
Excellent	228 (46%)	246 (35%)
Very Good	162 (32%)	223 (32%)
Good	85 (17%)	164 (23%)
Fair	21 (4.2%)	57 (8.1%)
Poor	5 (1.0%)	14 (2.0%)
Wears glasses	221 (44%)	403 (57%)
Age at first birth	24.0 (21.0, 29.0)	21.0 (18.0, 26.0)

[1] n (%); Median (IQR)

```r
tbl_summary(
  nlsy,
  by = sex_cat,
  include = c(race_eth_cat, eyesight_cat,
              glasses, age_bir),
  label = list(
    race_eth_cat ~ "Race/ethnicity",
    eyesight_cat ~ "Eyesight",
    glasses ~ "Wears glasses",
    age_bir ~ "Age at first birth"
  )) |>
  add_p(test = list(
        all_continuous() ~ "t.test",
        all_categorical() ~ "chisq.test")) |
  add_overall(col_label = "**Total**") |>
  bold_labels() |>
  modify_footnote(update = everything() ~ NA
  modify_header(label = "**Variable**",
                p.value = "**P**")
```

Variable	**Total**	**Male**, N = 501	**Female**, N = 704	**P**
Race/ethnicity				0.3
Hispanic	211 (18%)	81 (16%)	130 (18%)	
Black	307 (25%)	138 (28%)	169 (24%)	
Non-Black, Non-Hispanic	687 (57%)	282 (56%)	405 (58%)	
Eyesight				<0.001
Excellent	474 (39%)	228 (46%)	246 (35%)	
Very Good	385 (32%)	162 (32%)	223 (32%)	
Good	249 (21%)	85 (17%)	164 (23%)	
Fair	78 (6.5%)	21 (4.2%)	57 (8.1%)	
Poor	19 (1.6%)	5 (1.0%)	14 (2.0%)	
Wears glasses	624 (52%)	221 (44%)	403 (57%)	<0.001
Age at first birth	22.0 (19.0, 27.0)	24.0 (21.0, 29.0)	21.0 (18.0, 26.0)	<0.001

`tbl_summary()`

- Incredibly customizeable
- Really helpful with Table 1
- I often just view in the web browser and copy and paste into a Word document
- Can also be used within quarto/R Markdown
- If output is Word, I use `as_flex_table()` to output using the `flextable` package
- Make even more customizeable with the `gt` package with `as_gt()`

Exercise

Regression

Regressions take a formula: `y ~ x1 + x2 + x3`

- Include interaction terms between `x1` and `x2` with `y ~ x1*x2 + x3`
- Main effects of `x1` and `x2` will be included too
- Indicator ("dummy") variables will automatically be created for factors
- The first level will be the reference level
- If you want to include a squared term (for example), you can make the squared variable first, or wrap in `I()`: `y ~ x1 + I(x1^2)`

Regression

To fit a linear regression (by ordinary least squares), use the `lm()` function

To fit a generalized linear model (e.g., logistic regression, Poisson regression) use `glm()` and specify the `family =` argument

- `family = gaussian()` is the default: another way of fitting a linear regresion
- `family = binomial()` gives you logistic regression
- `family = poisson()` is Poisson regression

We tell R what dataset to pull the variables from with `data =`

Regression

```r
linear_model <- lm(income ~ sex_cat*age_bir + race_eth_cat,
                   data = nlsy)
```

```r
logistic_model <- glm(glasses ~ eyesight_cat + sex_cat + income,
                      data = nlsy, family = binomial())
```

Regression

```
1 coef(linear_model)
```

```
                         (Intercept)                     sex_catFemale 
                           1029.3339                        -4681.2484 
                             age_bir                 race_eth_catBlack 
                            482.4650                         -299.6490 
race_eth_catNon-Black, Non-Hispanic             sex_catFemale:age_bir 
                           6418.4568                          161.5008 
```

```
1 confint(linear_model)
```

```
                                          2.5 %    97.5 %
(Intercept)                         -3746.58543 5805.2531
sex_catFemale                      -10553.32090 1190.8242
age_bir                               303.50836  661.4217
race_eth_catBlack                   -2448.39108 1849.0932
race_eth_catNon-Black, Non-Hispanic  4500.91244 8336.0012
sex_catFemale:age_bir                 -77.30931  400.3109
```

Regression

```
1  exp(coef(logistic_model))
```

```
        (Intercept) eyesight_catVery Good      eyesight_catGood 
          0.6338301             0.9172091             0.8608297 
      eyesight_catFair      eyesight_catPoor         sex_catFemale 
          0.5798278             1.1624355             1.8362460 
             income 
          1.0000174 
```

```
1  summary(logistic_model)
```

```

Call:
glm(formula = glasses ~ eyesight_cat + sex_cat + income, family = binomial(), 
    data = nlsy)

Coefficients:
                        Estimate Std. Error z value Pr(>|z|)    
(Intercept)           -4.560e-01  1.383e-01  -3.298 0.000975 ***
eyesight_catVery Good -8.642e-02  1.400e-01  -0.617 0.537031    
eyesight_catGood      -1.499e-01  1.604e-01  -0.934 0.350267    
eyesight_catFair      -5.450e-01  2.535e-01  -2.150 0.031585 *  
eyesight_catPoor       1.505e-01  4.786e-01   0.315 0.753121    
sex_catFemale          6.077e-01  1.210e-01   5.021 5.14e-07 ***
income                 1.745e-05  4.613e-06   3.782 0.000155 ***
---
```

Helpful regression packages

`{broom}` helps "tidy" regression results

```r
library(broom)
tidy(linear_model)
```

```
# A tibble: 6 × 5
  term                              estimate std.error statistic  p.value
  <chr>                                <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)                          1029.     2434.     0.423 6.72e- 1
2 sex_catFemale                       -4681.     2993.    -1.56  1.18e- 1
3 age_bir                               482.       91.2    5.29  1.46e- 7
4 race_eth_catBlack                    -300.     1095.    -0.274 7.84e- 1
5 race_eth_catNon-Black, Non-Hispanic  6418.      977.     6.57  7.63e-11
6 sex_catFemale:age_bir                 162.      122.     1.33  1.85e- 1
```

broom::tidy()

```
tidy(logistic_model, conf.int = TRUE, exponentiate = TRUE)
```

```
# A tibble: 7 × 7
  term                    estimate  std.error statistic p.value conf.low conf.high
  <chr>                      <dbl>      <dbl>     <dbl>   <dbl>    <dbl>     <dbl>
1 (Intercept)                0.634 0.138         -3.30  9.75e-4    0.483     0.830
2 eyesight_catVery Good      0.917 0.140         -0.617 5.37e-1    0.697     1.21
3 eyesight_catGood           0.861 0.160         -0.934 3.50e-1    0.628     1.18
4 eyesight_catFair           0.580 0.254         -2.15  3.16e-2    0.350     0.949
5 eyesight_catPoor           1.16  0.479          0.315 7.53e-1    0.458     3.08
6 sex_catFemale              1.84  0.121          5.02  5.14e-7    1.45      2.33
7 income                     1.00  0.00000461     3.78  1.55e-4    1.00      1.00
```

`broom::tidy()` can also help other statistics

```r
tabyl(nlsy, sex_cat, eyesight_cat) |>
  chisq.test() |>
  tidy()
```

```
# A tibble: 1 × 4
  statistic  p.value parameter method
      <dbl>    <dbl>     <int> <chr>
1      22.7 0.000143         4 Pearson's Chi-squared test
```

```r
t.test(income ~ sex_cat, data = nlsy) |>
  tidy()
```

```
# A tibble: 1 × 10
  estimate estimate1 estimate2 statistic p.value parameter conf.low conf.high
     <dbl>     <dbl>     <dbl>     <dbl>   <dbl>     <dbl>    <dbl>     <dbl>
1    2397.    16690.    14292.      3.00 0.00273      963.     831.     3963.
# ℹ 2 more variables: method <chr>, alternative <chr>
```

gtsummary::tbl_regression()

```r
tbl_regression(
  linear_model,
  intercept = TRUE,
  label = list(
    sex_cat ~ "Sex",
    race_eth_cat ~ "Race/ethnicity",
    age_bir ~ "Age at first birth",
    `sex_cat:age_bir` ~ "Sex/age interaction
  ))
```

Characteristic	**Beta**	**95% CI**[1]	**p-value**
(Intercept)	1,029	-3,747, 5,805	0.7
Sex			
Male	—	—	
Female	-4,681	-10,553, 1,191	0.12
Age at first birth	482	304, 661	<0.001
Race/ethnicity			
Hispanic	—	—	
Black	-300	-2,448, 1,849	0.8
Non-Black, Non-Hispanic	6,418	4,501, 8,336	<0.001
Sex/age interaction			
Female * Age at first birth	162	-77, 400	0.2

[1] CI = Confidence Interval

gtsummary::tbl_regression()

```r
tbl_regression(
  logistic_model,
  exponentiate = TRUE,
  label = list(
    sex_cat ~ "Sex",
    eyesight_cat ~ "Eyesight",
    income ~ "Income"
  ))
```

Characteristic	**OR**[1]	**95% CI**[1]	**p-value**
Eyesight			
Excellent	—	—	
Very Good	0.92	0.70, 1.21	0.5
Good	0.86	0.63, 1.18	0.4
Fair	0.58	0.35, 0.95	0.032
Poor	1.16	0.46, 3.08	0.8
Sex			
Male	—	—	
Female	1.84	1.45, 2.33	<0.001
Income	1.00	1.00, 1.00	<0.001

[1] OR = Odds Ratio, CI = Confidence Interval

You could put several together

```r
linear_model_no_int <- lm(income ~ sex_cat + age_bir + race_eth_ca

tbl_no_int <- tbl_regression(
  linear_model_no_int,
  intercept = TRUE,
  label = list(
    sex_cat ~ "Sex",
    race_eth_cat ~ "Race/ethnicity",
    age_bir ~ "Age at first birth"
  ))

tbl_int <- tbl_regression(
  linear_model,
  intercept = TRUE
```

You could put several together

```r
tbl_merge(list(tbl_no_int, tbl_int),
          tab_spanner = c("**Model 1**", "**Model 2**"))
```

	Model 1			Model 2		
Characteristic	**Beta**	**95% CI**[1]	**p-value**	**Beta**	**95% CI**[1]	**p-value**
(Intercept)	-1,201	-4,657, 2,256	0.5	1,029	-3,747, 5,805	0.7
Sex						
Male	—	—		—	—	
Female	-833	-2,283, 617	0.3	-4,681	-10,553, 1,191	0.12
Age at first birth	571	448, 693	<0.001	482	304, 661	<0.001
Race/ethnicity						
Hispanic	—	—		—	—	
Black	-287	-2,436, 1,863	0.8	-300	-2,448, 1,849	0.8
Non-Black, Non-Hispanic	6,434	4,516, 8,352	<0.001	6,418	4,501, 8,336	<0.001
Sex/age interaction						
Female * Age at first birth				162	-77, 400	0.2

[1] CI = Confidence Interval

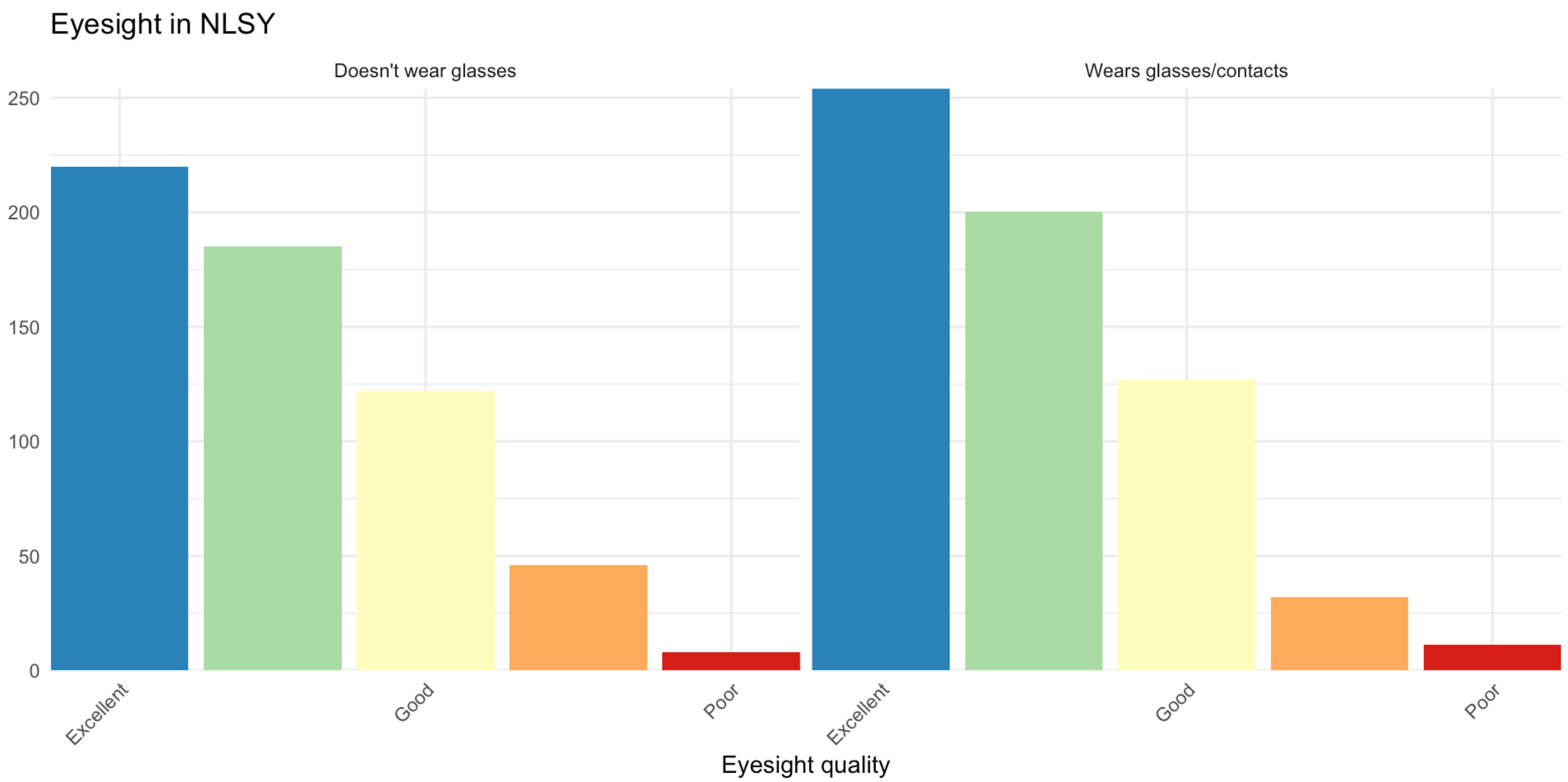
Regression package recommendation from Xiyue: autoReg

```r
# install.packages("autoReg")
library(autoReg)
autoReg(linear_model) |>
  myft()
```

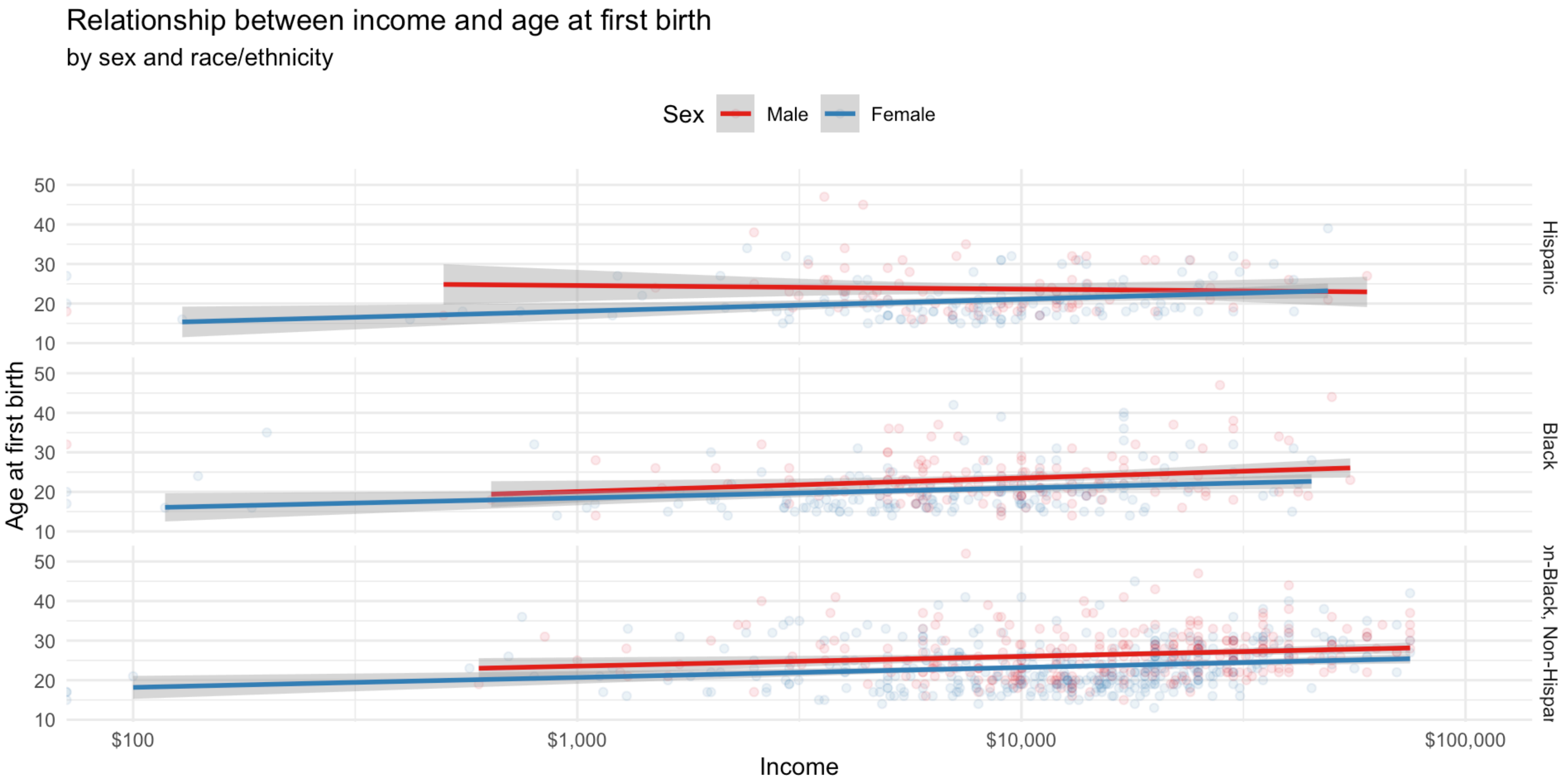
Dependent: income		unit	value	Coefficient (multivariable)
sex_cat	Female (N=704)	Mean ± SD	16689.6 ± 14526.5	
	Male (N=501)	Mean ± SD	14292.3 ± 12321.0	-4681.25 (-10553.32 to 1190.82, p=.118)
age_bir	[13,52]	Mean ± SD	23.4 ± 6.0	482.47 (303.51 to 661.42, p<.001)
race_eth_cat	Black (N=307)	Mean ± SD	10794.8 ± 9468.8	
	Hispanic (N=211)	Mean ± SD	10489.7 ± 9209.5	-299.65 (-2448.39 to 1849.09, p=.784)
	Non-Black, Non-Hispanic (N=687)	Mean ± SD	18814.0 ± 14750.7	6418.46 (4500.91 to 8336.00, p<.001)
sex_cat:age_bir	Male:			
sex_cat:age_bir	Female:			161.50 (-77.31 to 400.31, p=.185)

Exercise

Figures in R using `ggplot()`



Figures in R using `ggplot()`



Why ggplot?

- Powerful and flexible: create complex and customized visualizations easily
- Reproducibility and efficiency: promotes reproducibility by offering consistent syntax and saves time through automation of plot creation
- Layered approach: incrementally build visualizations with multiple layers, exploring different aspects of data
- Extensive customization: essentially infinite options to tailor visualizations

ggplot builds figures by adding on pieces via a particular "*g*rammar of *g*raphics"

A Layered Grammar of Graphics

Hadley WICKHAM

A grammar of graphics is a tool that enables us to concisely describe the components of a graphic. Such a grammar allows us to move beyond named graphics (e.g., the "scatterplot") and gain insight into the deep structure that underlies statistical graphics. This article builds on Wilkinson, Anand, and Grossman (2005), describing extensions and refinements developed while building an open source implementation of the grammar of graphics for R, `ggplot2`.

The topics in this article include an introduction to the grammar by working through the process of creating a plot, and discussing the components that we need. The grammar is then presented formally and compared to Wilkinson's grammar, highlighting the hierarchy of defaults, and the implications of embedding a graphical grammar into a programming language. The power of the grammar is illustrated with a selection of examples that explore different components and their interactions, in more detail. The article concludes by discussing some perceptual issues, and thinking about how we can

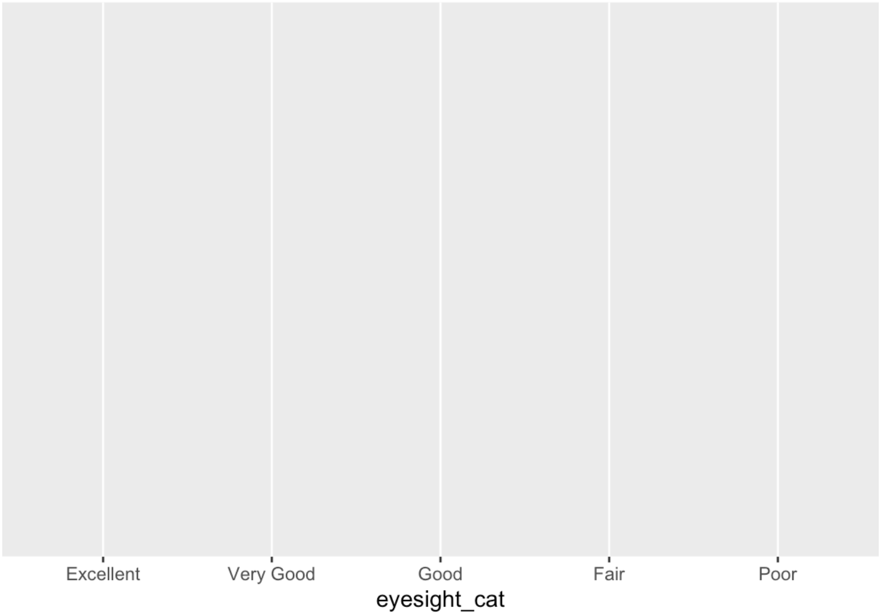
Basic structure of a ggplot

```
ggplot(data = {data},
       aes(x = {xvar}, y = {yvar}, <characteristic> = {othvar}, ..
      <geom>(<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
  - some graphs may not require both, or may require other parameters
- `{othvar}` is some other unquoted variable name that defines a grouping or other characteristic you want to map to an aesthetic
- `<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.
- `<geom>`: the geometric feature you want to use; e.g., point (scatterplot), line, histogram, bar, etc.
- `"value"`: a fixed value that defines some characteristic of the figure; e.g., “red”, 10, “dashed”
- … : there are numerous other options to discover!

Let's walk through the creation of a figure

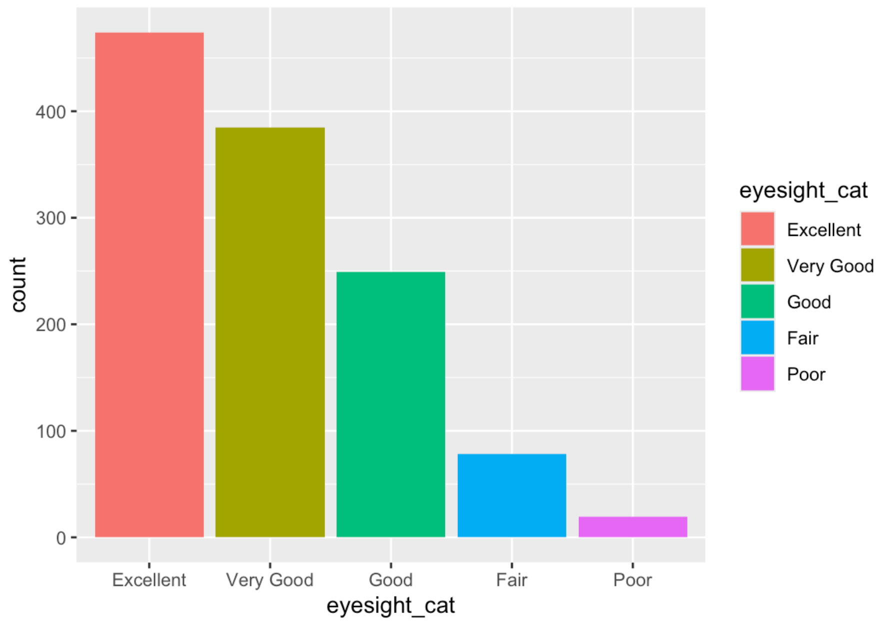
```
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat))
```



- `ggplot()` doesn't plot any data itself, it just sets up the data and variables

Let’s walk through the creation of a figure

```
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar()
```



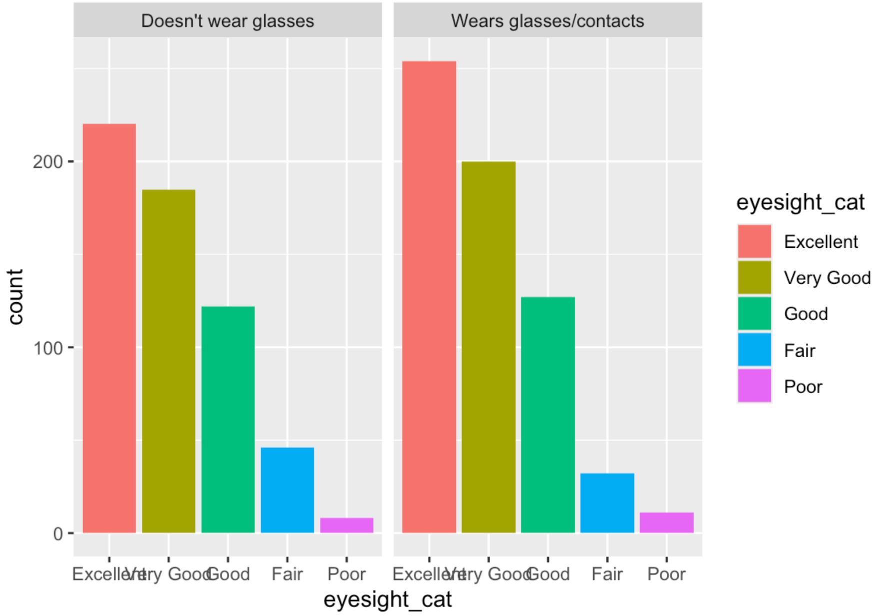
- `geom_bar()` creates a bar graph for the number of observations with a certain value of the `x` variable
  - does not need a `y` variable

Tip

use `geom_col()` if you have a `y` variable that you want to use as the height of the bars

Let’s walk through the creation of a figure

```r
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar() +
  facet_grid(cols = vars(glasses_cat))
```



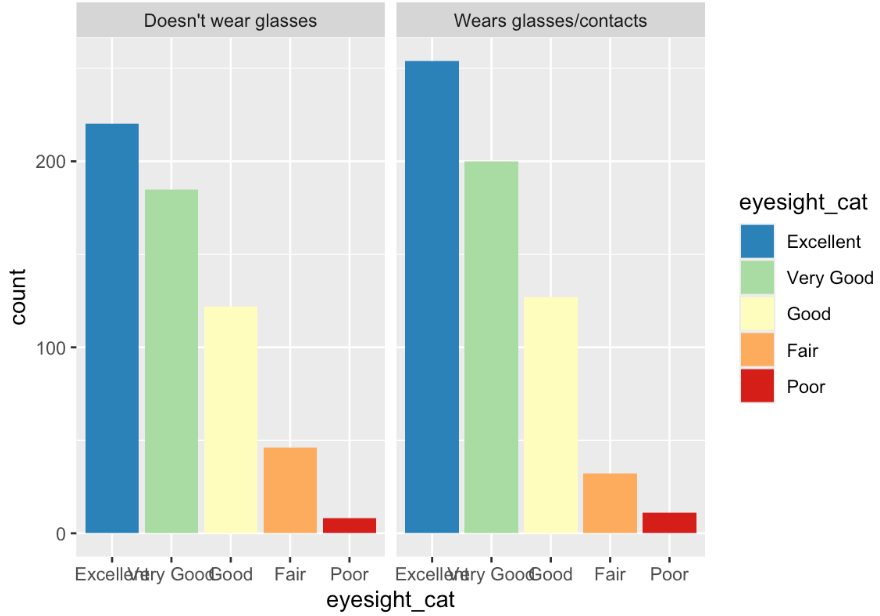
- `facet_grid()` creates a panel for each value of another variable
  - can also do `rows =`
  - variable name should be within `vars()` (you can use helpers like `starts_with()`)

Tip

use `facet_wrap()` if you want to create panels that expand along rows and columns (e.g., to facet by many countries)

Let’s walk through the creation of a figure

```
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar() +
  facet_grid(cols = vars(glasses_cat)) +
  scale_fill_brewer(palette = "Spectral",
                    direction = -1)
```



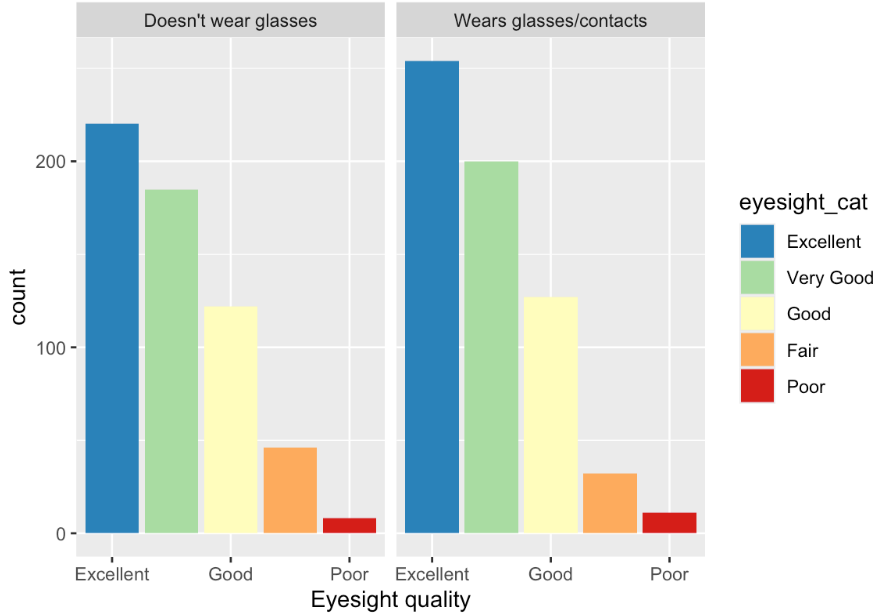
- `scale_{fill/color}_{...}()` functions change the color palette
  - some are appropriate for continuous variables, others discrete

Tip

`scale_fill_viridis_d()` good color-blind and black & white-friendly options

Let's walk through the creation of a figure

```
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar() +
  facet_grid(cols = vars(glasses_cat)) +
  scale_fill_brewer(palette = "Spectral",
                    direction = -1) +
  scale_x_discrete(breaks = c("Excellent",
                              "Good", "Poor"),
                   name = "Eyesight quality")
```



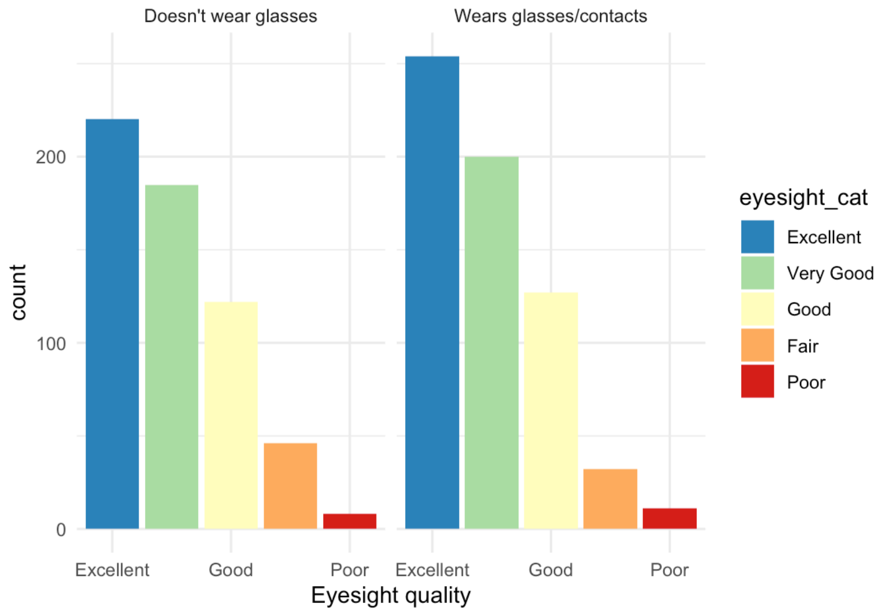
- `scale_{x/y}_{...}()` functions change the axis scale and/or labeling

Tip

`scale_y_log10()` is helpful when plotting odds or risk ratios

Let's walk through the creation of a figure

```
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar() +
  facet_grid(cols = vars(glasses_cat)) +
  scale_fill_brewer(palette = "Spectral",
                    direction = -1) +
  scale_x_discrete(breaks = c("Excellent",
                              "Good", "Poor"),
                   name = "Eyesight quality") +
  theme_minimal()
```

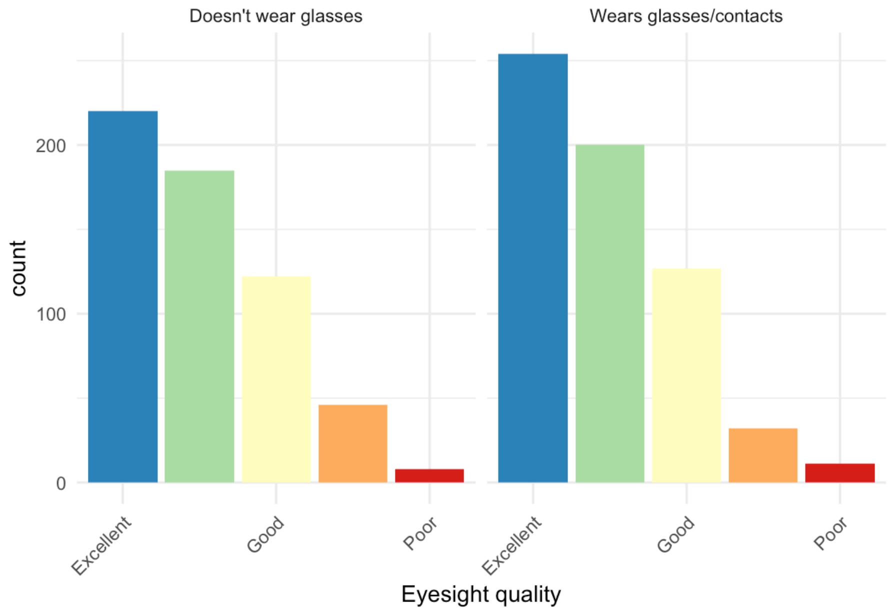


- `theme_{...}()` changes the "look" of the plot
  - but not the data color palette

> **Tip**
>
> find lots of themes and color palettes at https://yutannihilation.github.io/allYourFigureAreBelongToUs/ggthemes/

Let’s walk through the creation of a figure

```
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar() +
  facet_grid(cols = vars(glasses_cat)) +
  scale_fill_brewer(palette = "Spectral",
                    direction = -1) +
  scale_x_discrete(breaks = c("Excellent",
                              "Good", "Poor"),
                   name = "Eyesight quality") +
  theme_minimal() +
  theme(legend.position = "none",
        axis.text.x = element_text(
          angle = 45, vjust = 1, hjust = 1))
```



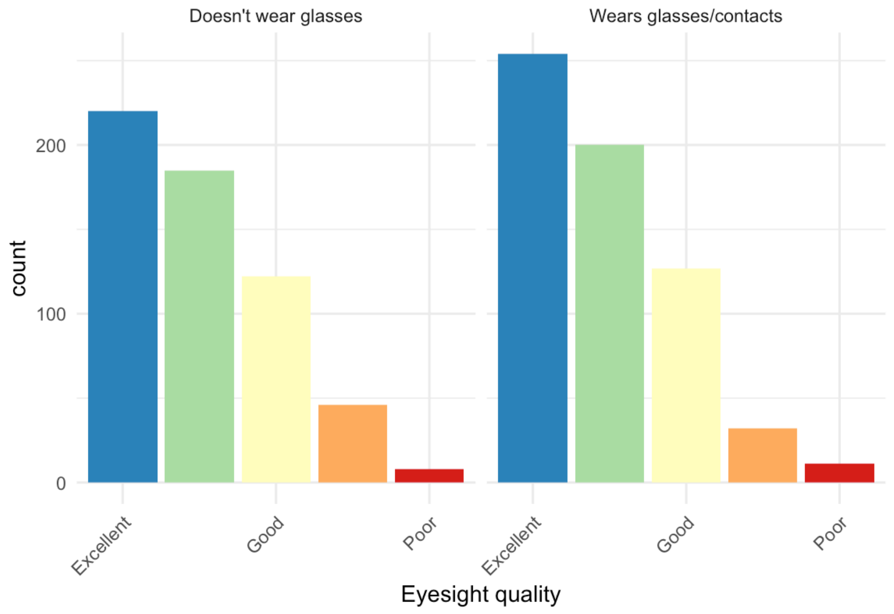
- you can also specify any component of the theme directly

Tip

lots of arguments can be set to `element_blank()` to get rid of them

Let’s walk through the creation of a figure

```r
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar() +
  facet_grid(cols = vars(glasses_cat)) +
  scale_fill_brewer(palette = "Spectral",
                    direction = -1) +
  scale_x_discrete(breaks = c("Excellent",
                              "Good", "Poor"),
                   name = "Eyesight quality") +
  theme_minimal() +
  theme(legend.position = "none",
        axis.text.x = element_text(
          angle = 45, vjust = 1, hjust = 1))
```

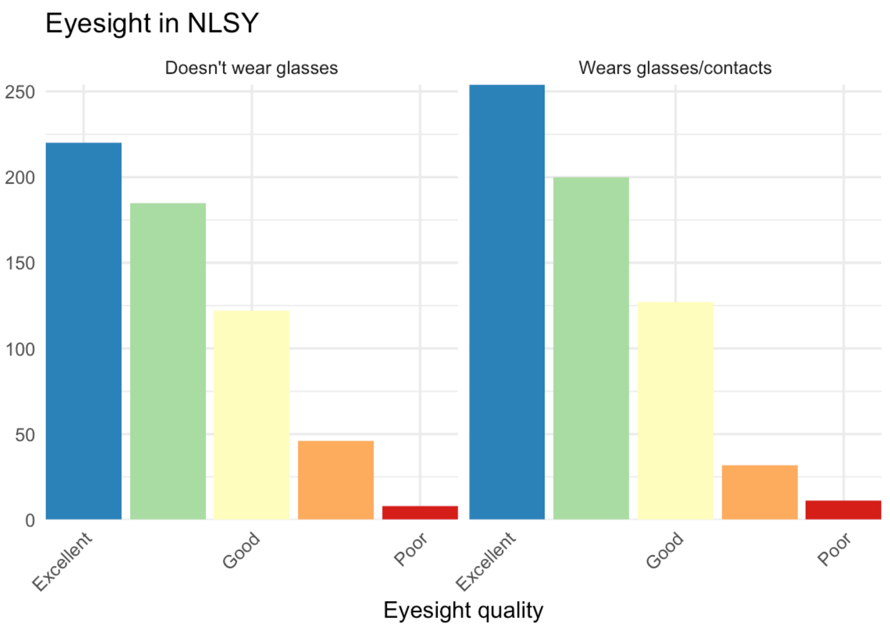


- `labs()` can add subtitles, caption, alt text, as well as label any aesthetics (fill, color, etc.)

> **Tip**
>
> there’s a lot of redundancy… we could have specified `x = "Eyesight quality"` here instead.

Let's walk through the creation of a figure

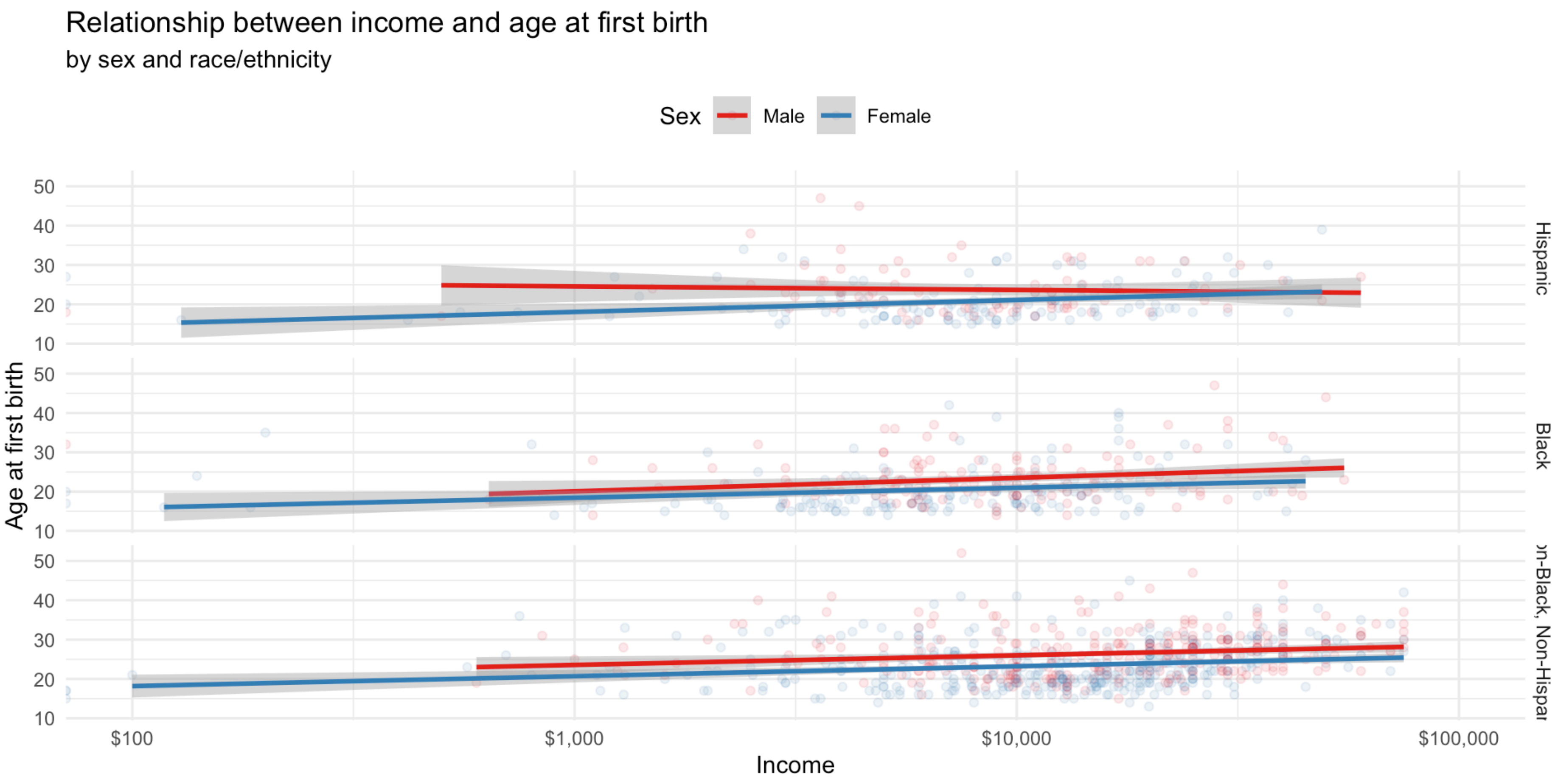
```r
ggplot(data = nlsy,
       aes(x = eyesight_cat,
           fill = eyesight_cat)) +
  geom_bar() +
  facet_grid(cols = vars(glasses_cat)) +
  scale_fill_brewer(palette = "Spectral",
                    direction = -1) +
  scale_x_discrete(breaks = c("Excellent",
                              "Good", "Poor"),
                   name = "Eyesight quality") +
  theme_minimal() +
  theme(legend.position = "none",
        axis.text.x = element_text(
          angle = 45, vjust = 1, hjust = 1)) +
  labs(title = "Eyesight in NLSY",
       y = NULL) +
  coord_cartesian(expand = FALSE)
```



- `coord_{...}()` functions change the coordinate system
  - cartesian is already the default, but `expand = FALSE` means there is no extra space beyond the axis limits

Tip

What are some of the layers we may need for this one?



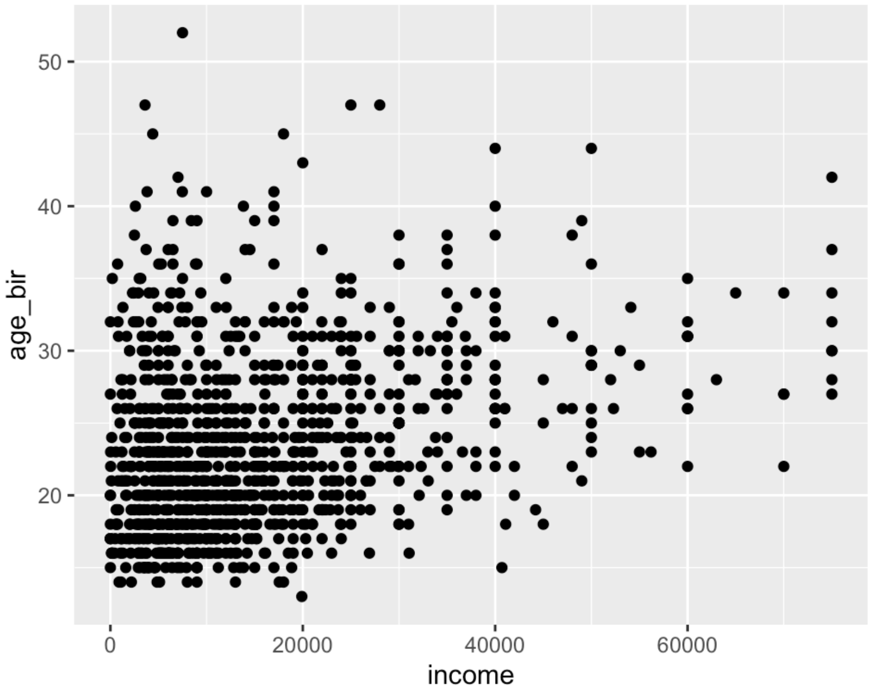
Returning to our basic structure

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

Let’s walk through some more examples in depth

Scatterplot: `geom_point()`

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

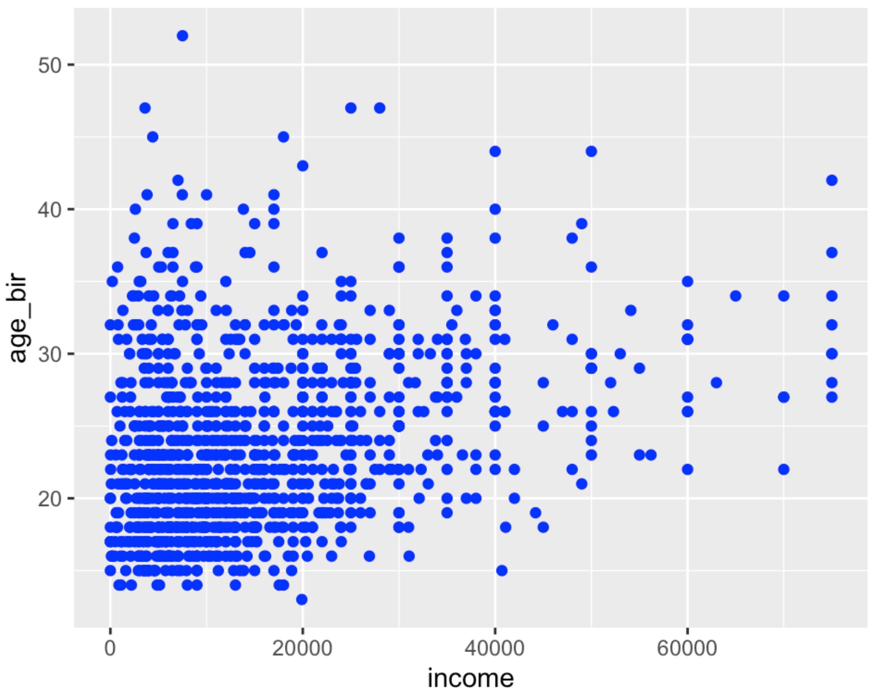


Question

How are we specifying the type of plot (scatterplot)? How are we specifying the variables to plot? How are we specifying the data used to plot it?

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

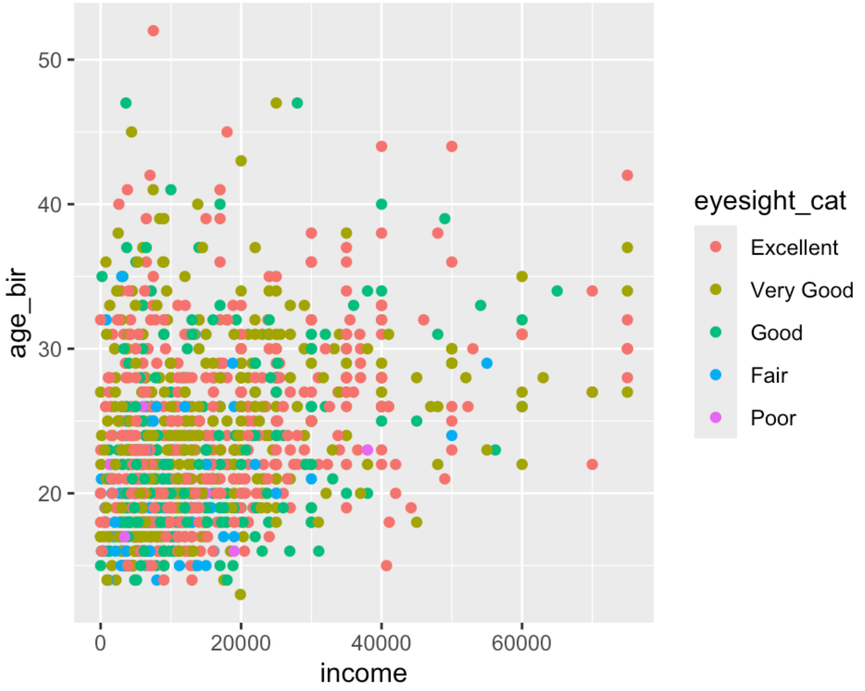
```r
ggplot(data = nlsy,
       aes(x = income, y = age_bir)) +
  geom_point(color = "blue")
```



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

What if we want the color to correspond to values of a variable?

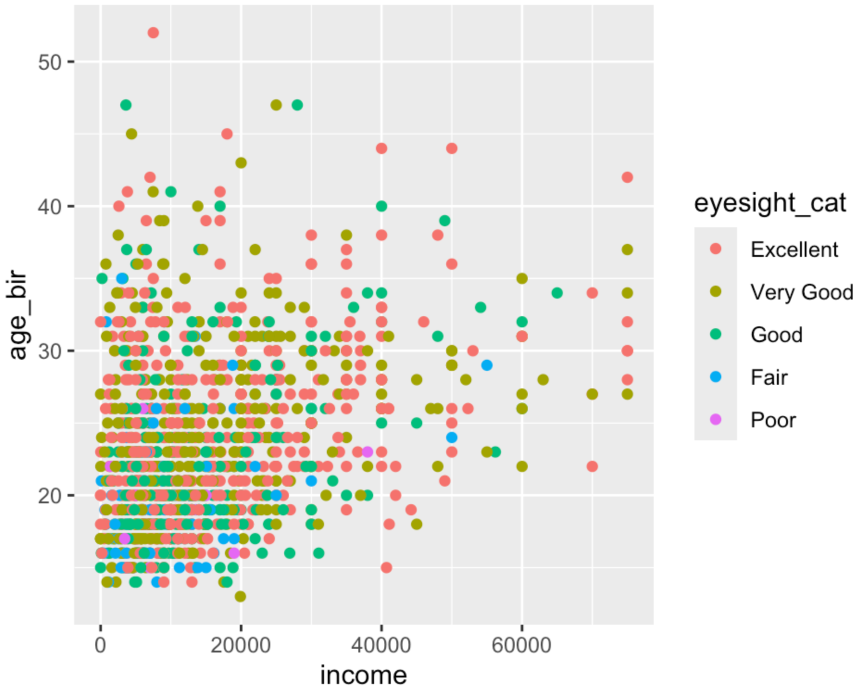
```
ggplot(data = nlsy,
       aes(x = income, y = age_bir,
           color = eyesight_cat)) +
  geom_point()
```



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

Alternative specification

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



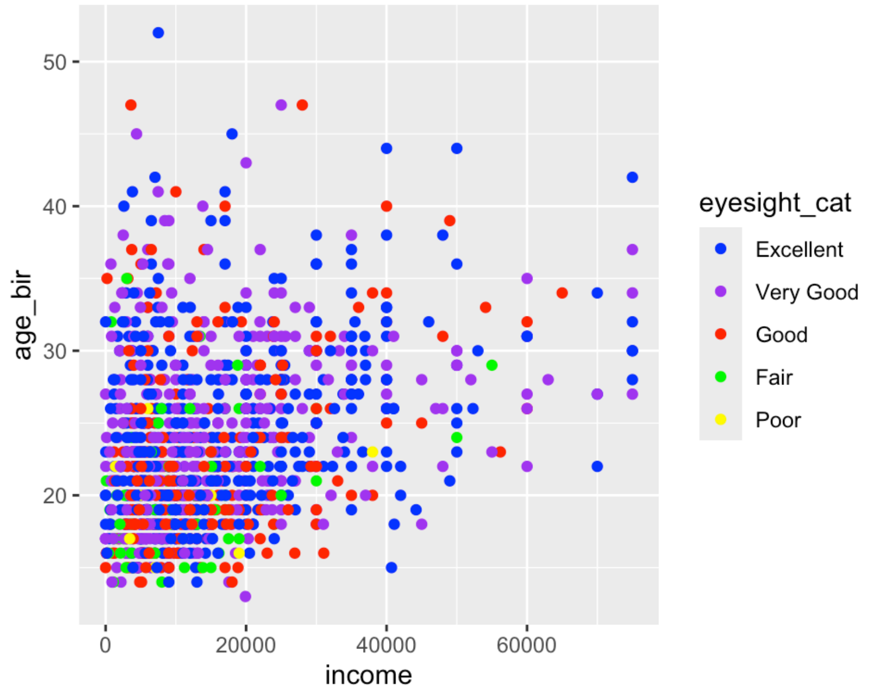
Note that we could also put the `aes()` (aesthetics) in the `geom_()` itself.

> ⚠ **Warning**
>
> If within `geom_point()`, it will only apply to that geom. Here it doesn't matter

Exercise

Let's change the colors

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



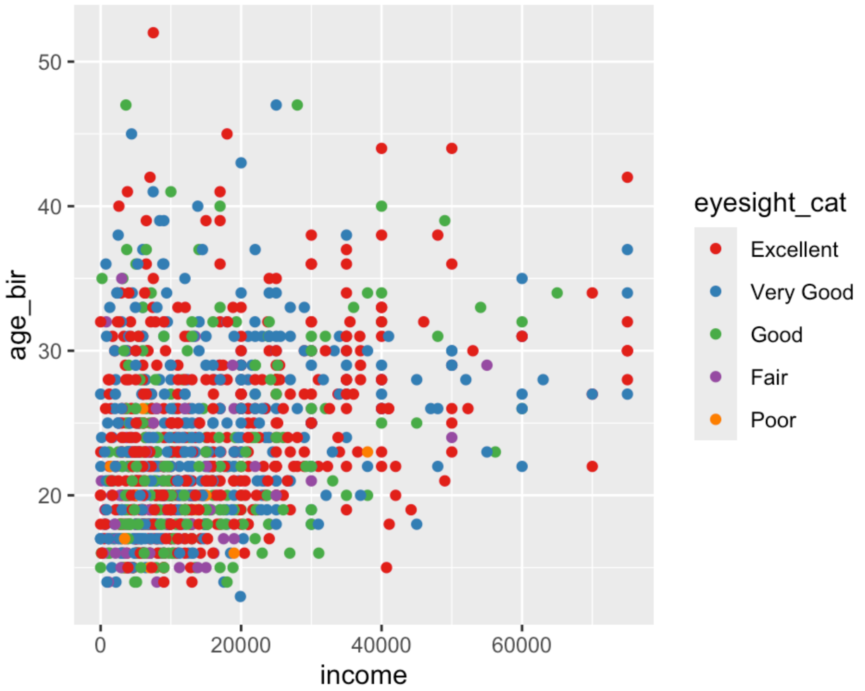
We add on another layer to specify the colors we want.

Tip

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

Color palettes

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



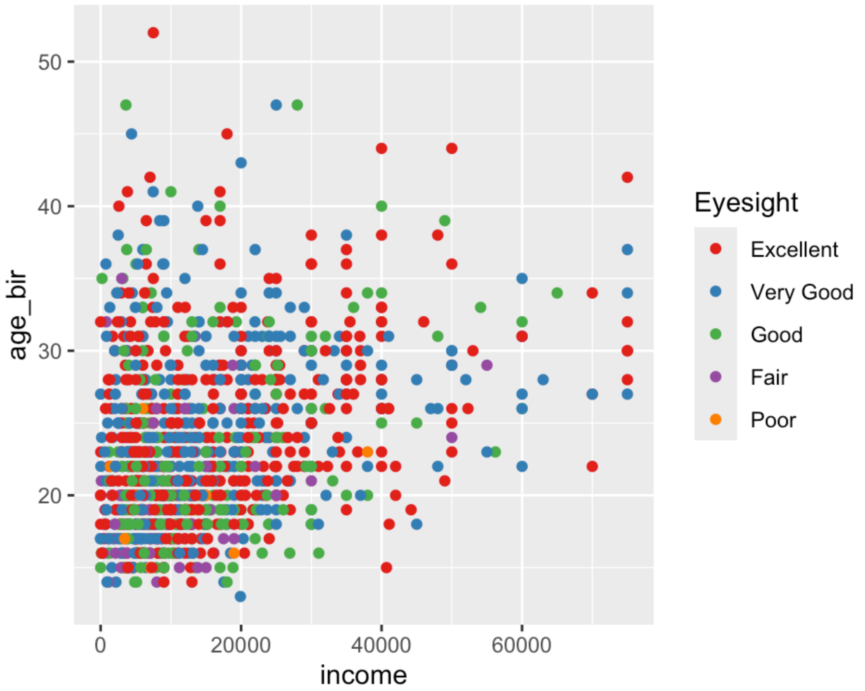
There are tons of different options in R for color palettes.

You can play around with those in the `RColorBrewer` package here.

You can access the scales in that package with

Change the title on the legend

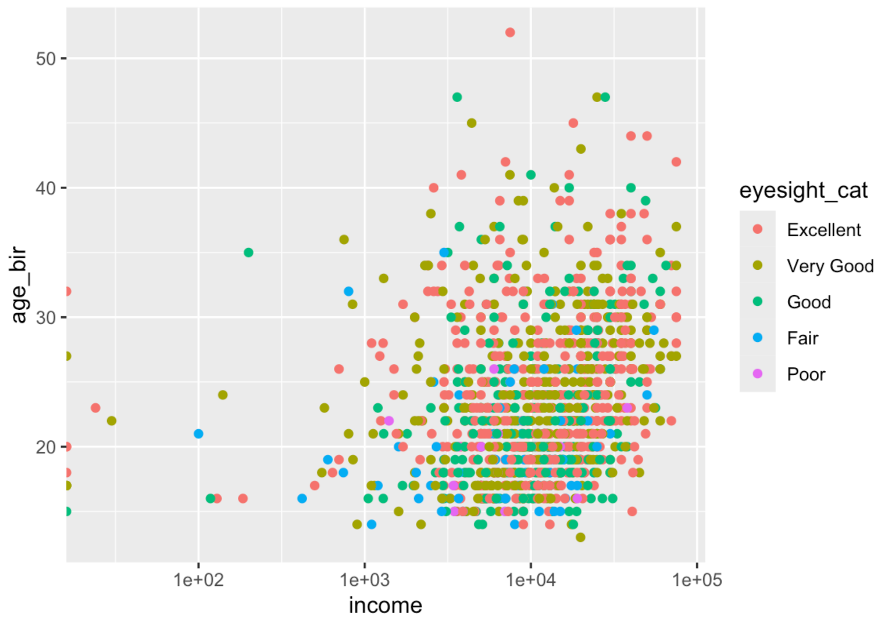
```r
ggplot(data = nlsy,
       aes(x = income, y = age_bir,
           color = eyesight_cat)) +
  geom_point()  +
  scale_color_brewer(palette = "Set1",
                     name = "Eyesight")
```



> **Tip**
>
> Each of the `scale_color_x()` functions has a lot of the same arguments. For a lot more info visit the ggplot2 book

Change the axis scale

```r
ggplot(data = nlsy,
       aes(x = income, y = age_bir,
           color = eyesight_cat)) +
  geom_point()  +
  scale_x_log10()
```



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

Tip

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

We can label the axis better

```
ggplot(nlsy,
       aes(x = income, y = age_bir,
           color = eyesight_cat)) +
  geom_point()  +
  scale_x_log10(labels = scales::dollar,
                limits = c(100, 100000),
                breaks = c(100, 1000, 10000,
                name = "Income (USD)")
```

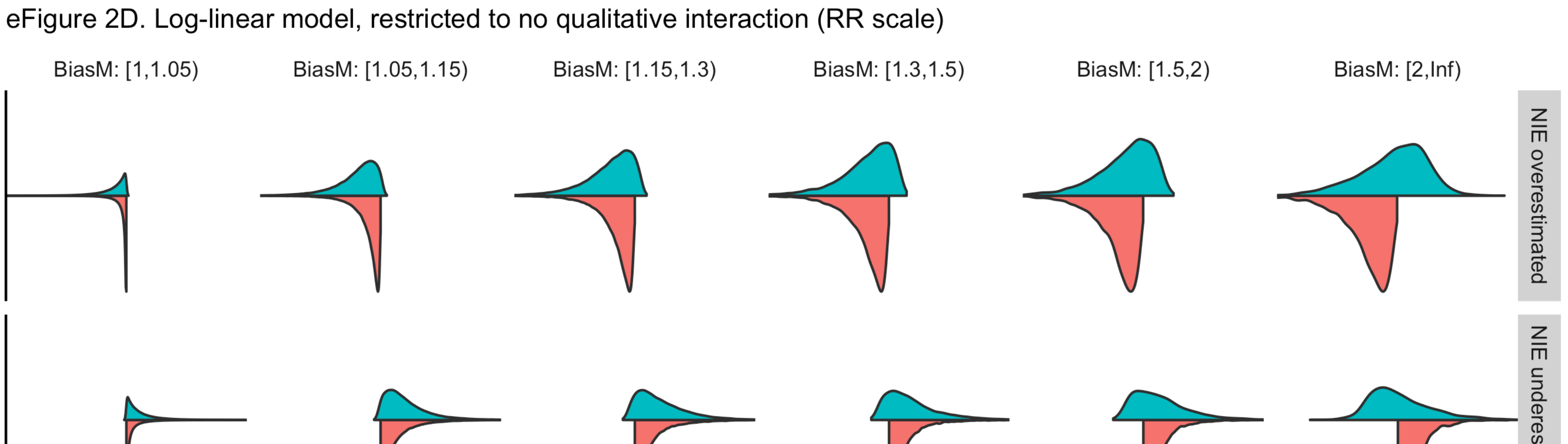
The `{scales}` packages contains lots of helpful number formatting functions

See the the examples in `help(scale_x_log10)` for some of them

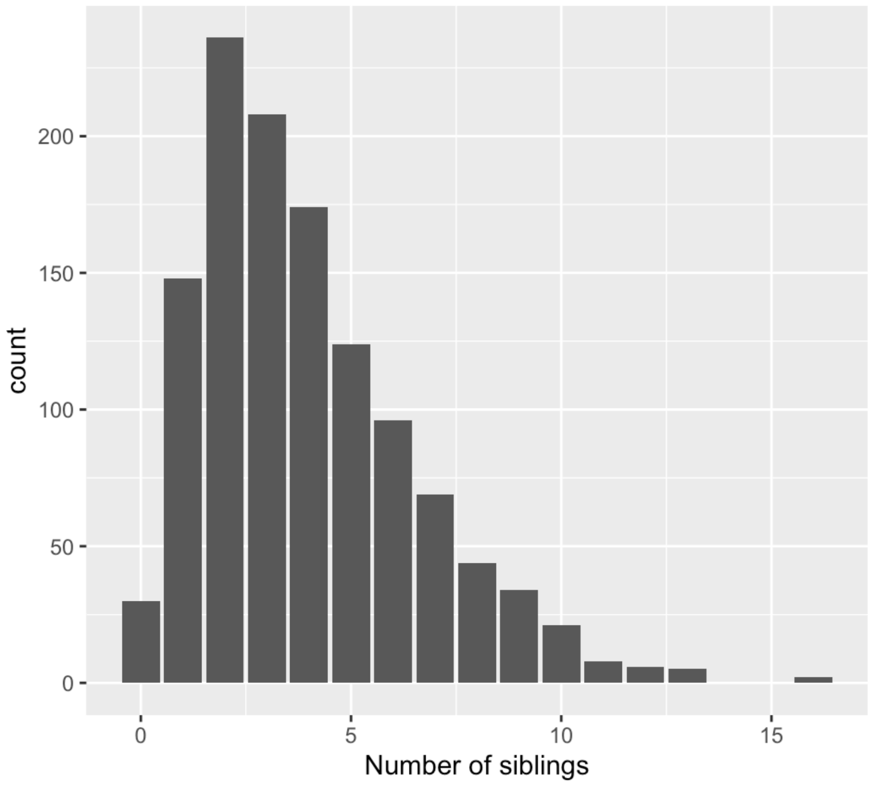
Exercise

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



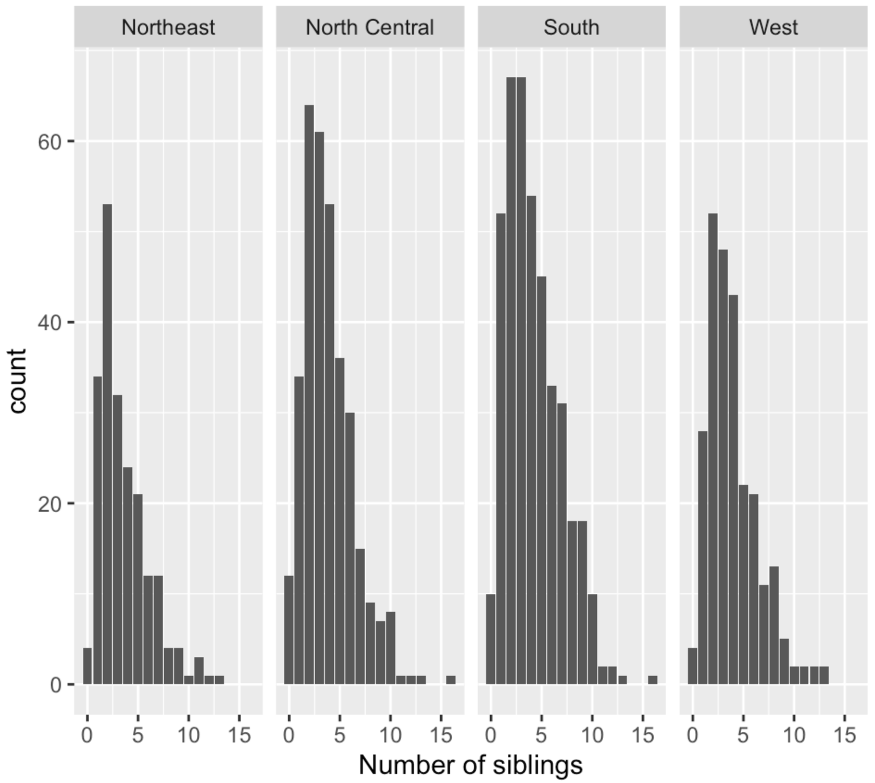
```r
ggplot(data = nlsy, aes(x = nsibs)) +
  geom_bar() +
  labs(x = "Number of siblings")
```



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.

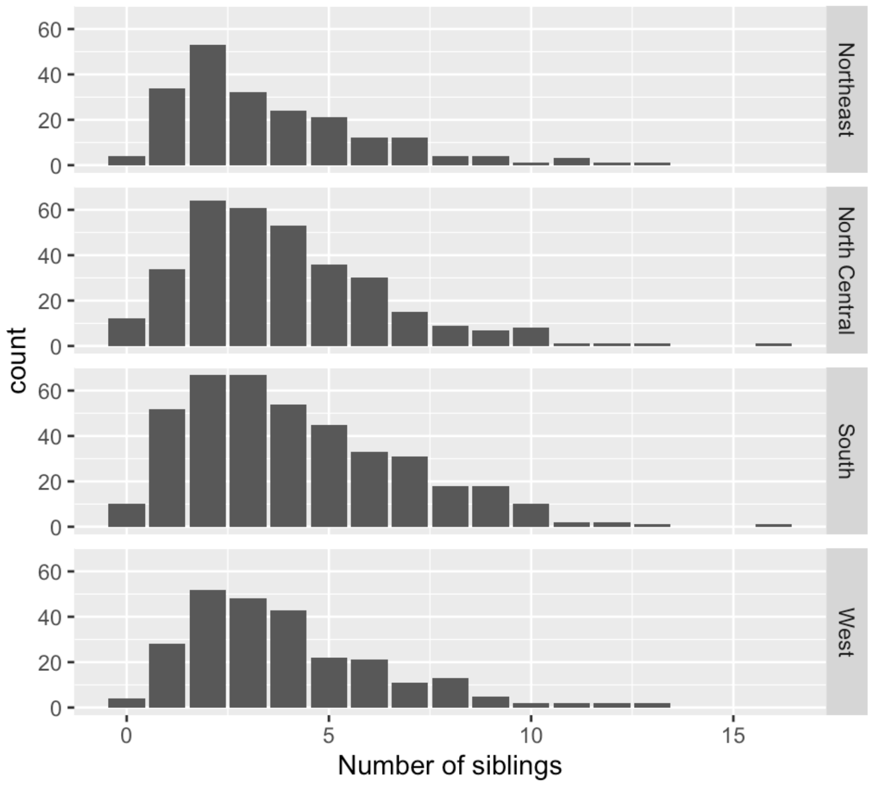
```r
ggplot(data = nlsy, aes(x = nsibs)) +
  geom_bar() +
  labs(x = "Number of siblings") +
  facet_grid(cols = vars(region_cat))
```



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

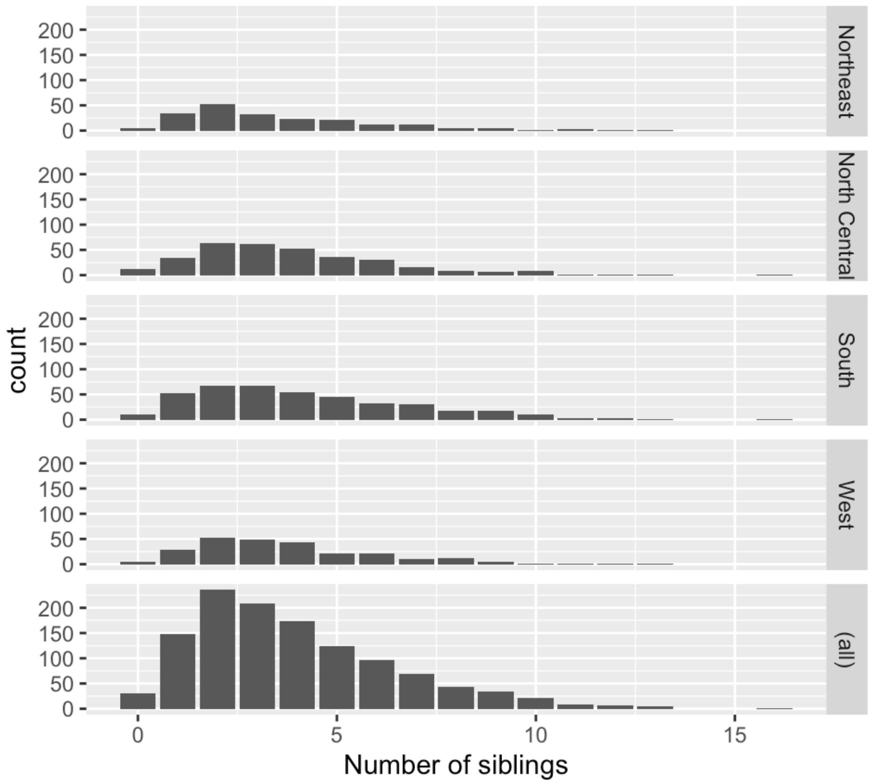
Here we've split it by region into columns.

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



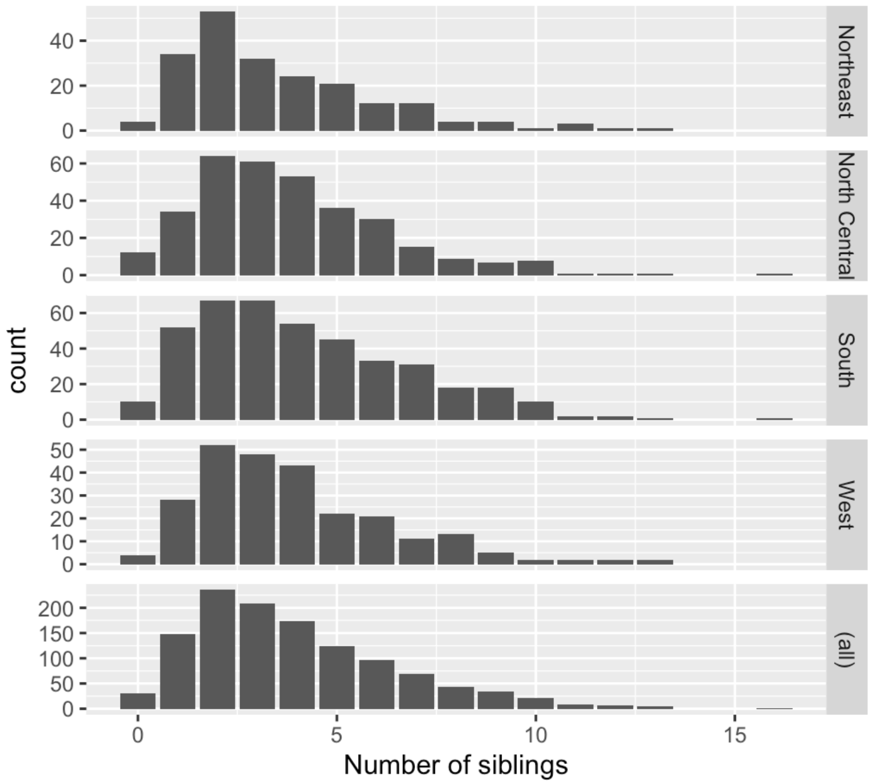
Since that was hard to read, we'll probably want to split by rows instead.

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



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

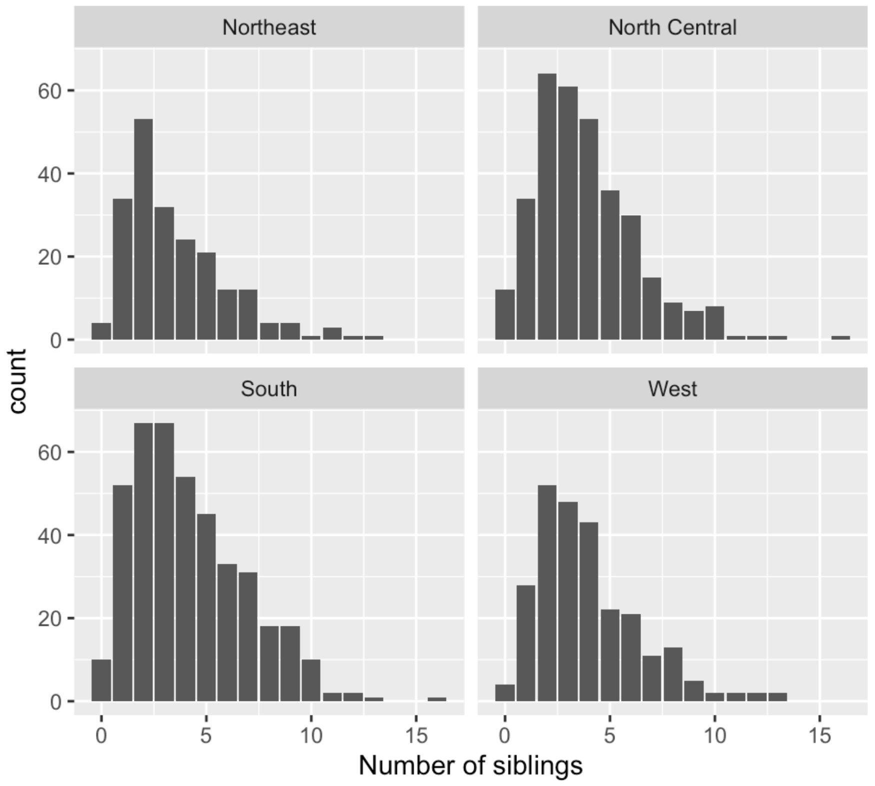
```r
ggplot(data = nlsy, aes(x = nsibs)) +
  geom_bar() +
  labs(x = "Number of siblings") +
  facet_grid(rows = vars(region_cat),
             margins = TRUE,
             scales = "free_y")
```



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

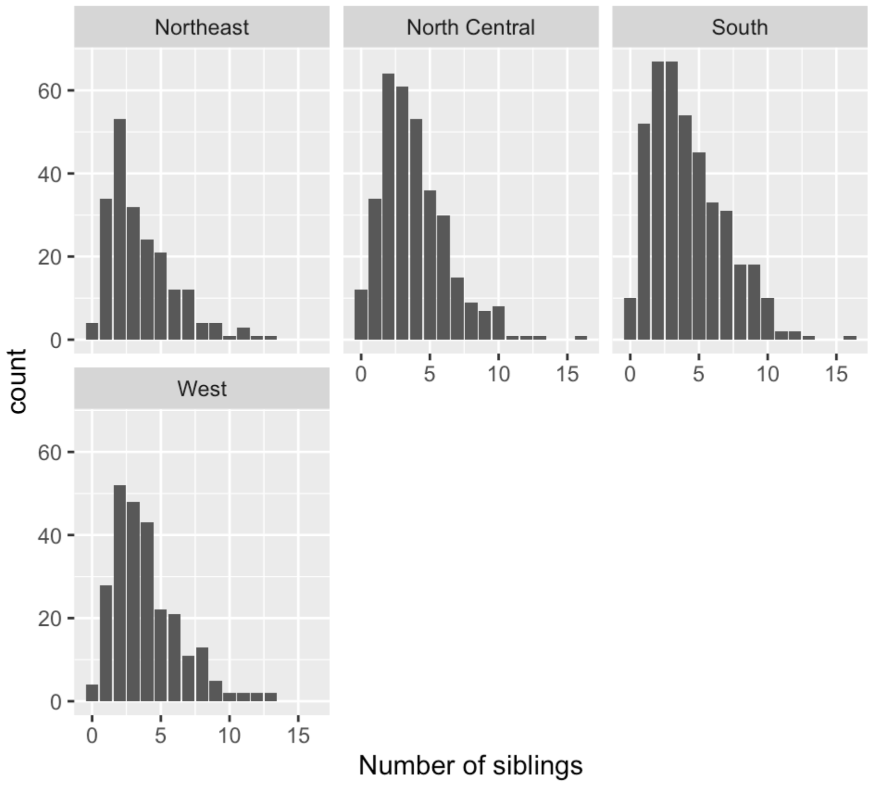
Other options are “free_x” if we want to allow the x-axis scale to vary, or just “free” to allow for both.

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



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, aes(x = nsibs)) +
  geom_bar() +
  labs(x = "Number of siblings") +
  facet_wrap(vars(region_cat),
             ncol = 3)
```

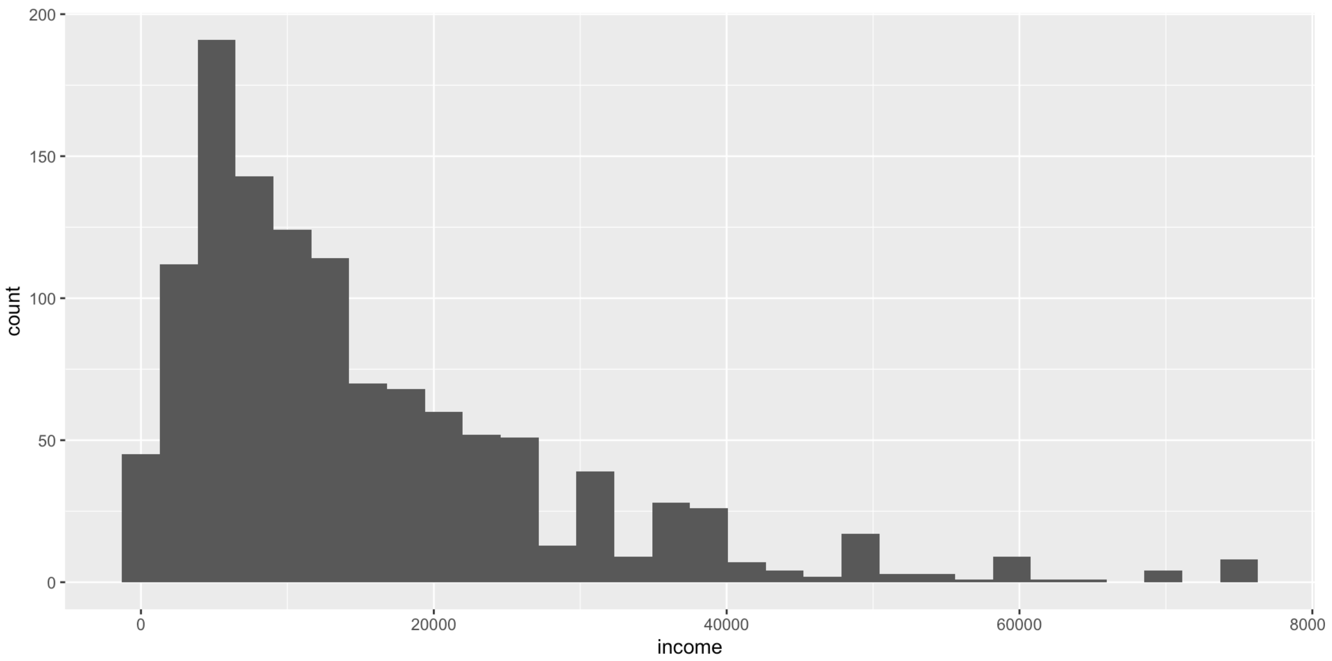


It tries to make a good decision, but you can override how many columns you want!

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

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

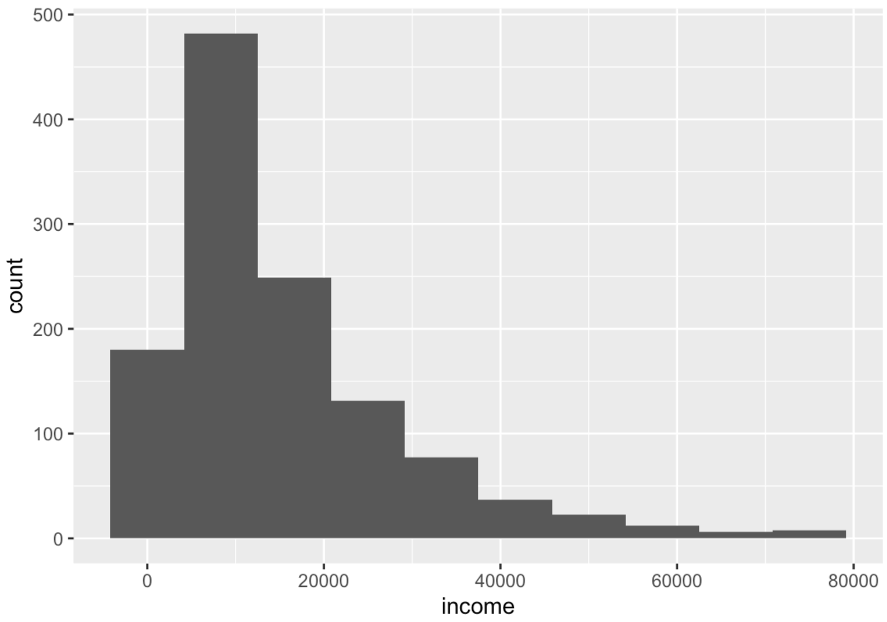


`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

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!

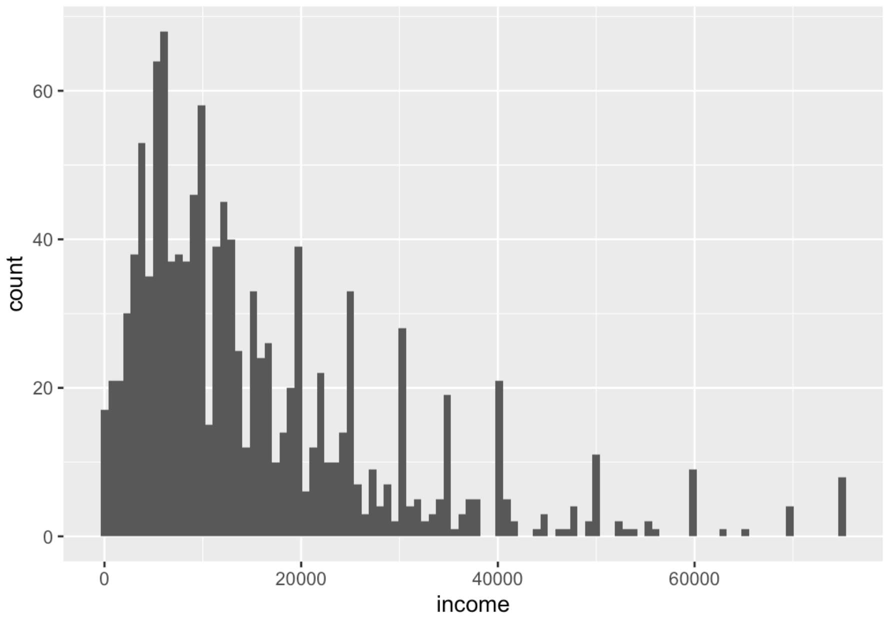
```r
ggplot(nlsy, aes(x = income)) +
  geom_histogram(bins = 10)
```



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

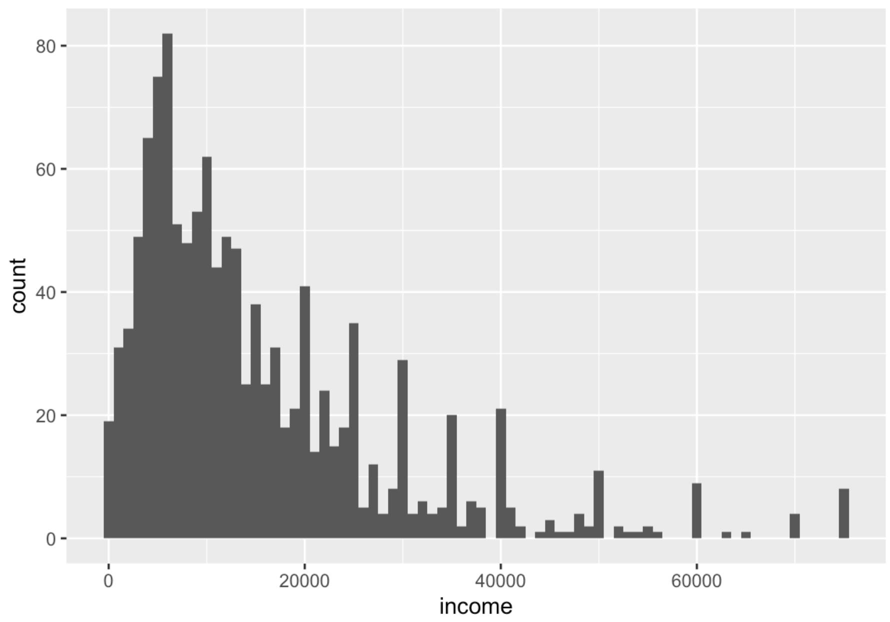
> **Note**
>
> Note how this fits into the `<characteristic> = "value"` structure

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



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

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

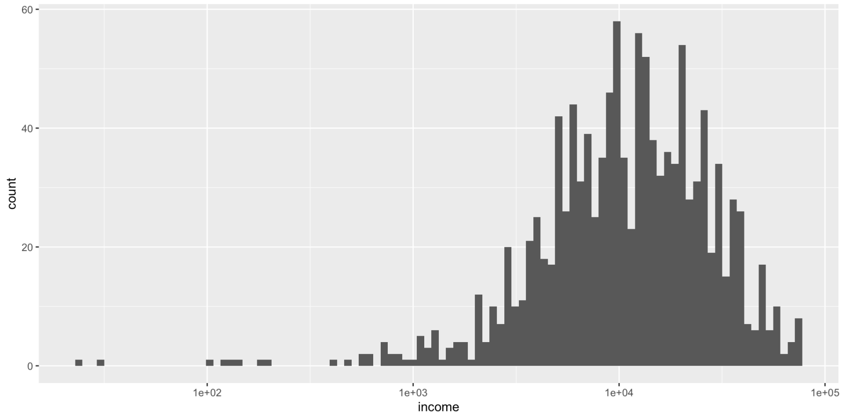


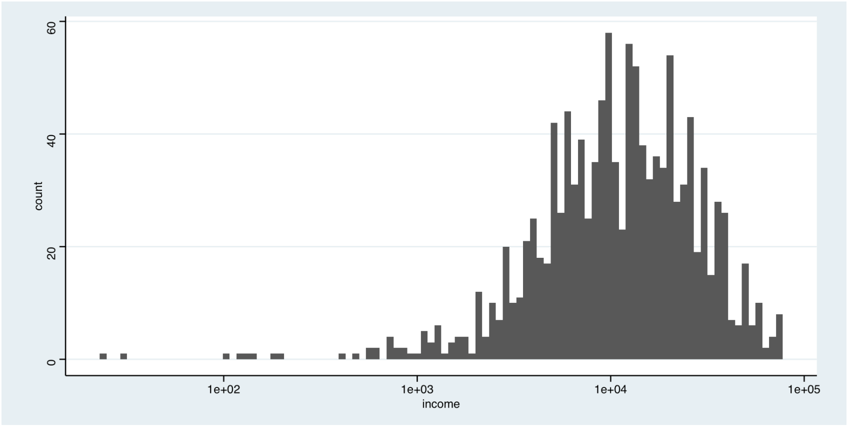
Sometimes the bin width actually has some meaning so we want to specify that

Here, each bin is $1000 – you can see the $5000 and $10000 increments

Themes to make our plots prettier

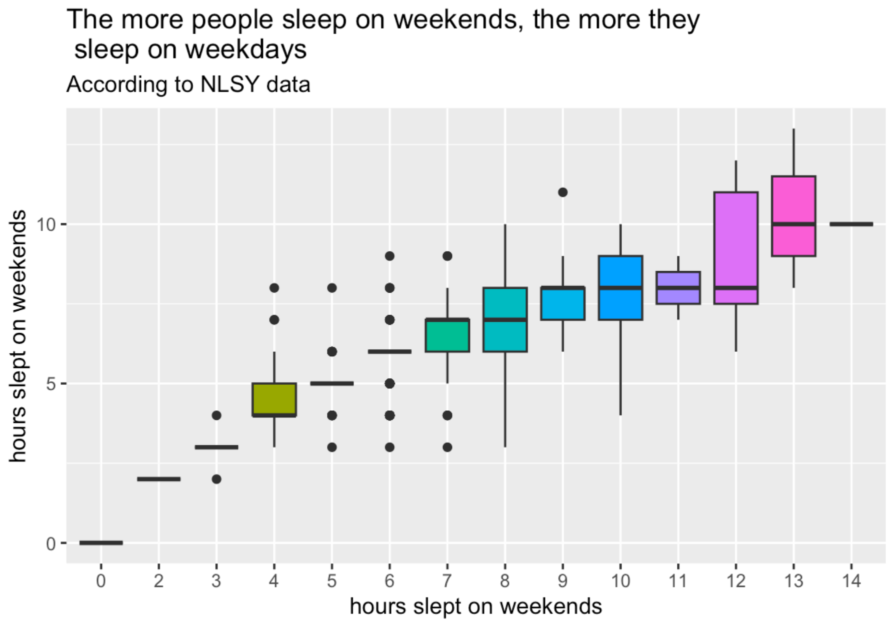
You probably recognize the ggplot theme. But did you know you can trick people into thinking you made your figures in Stata?





```r
p <- ggplot(nlsy,
            aes(x = factor(sleep_wknd),
                y = sleep_wkdy,
                fill = factor(sleep_wknd)))
  geom_boxplot() +
  scale_fill_discrete(guide = "none") +
  labs(x = "hours slept on weekends",
       y = "hours slept on weekends",
       title = "The more people sleep on wee
       subtitle = "According to NLSY data")

p
```



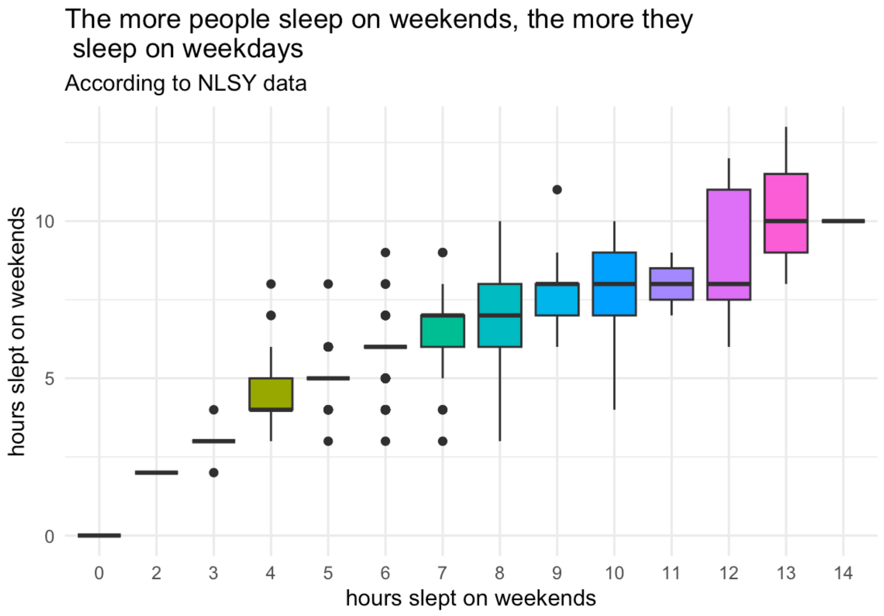
Let’s store our plot first.

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

Question

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

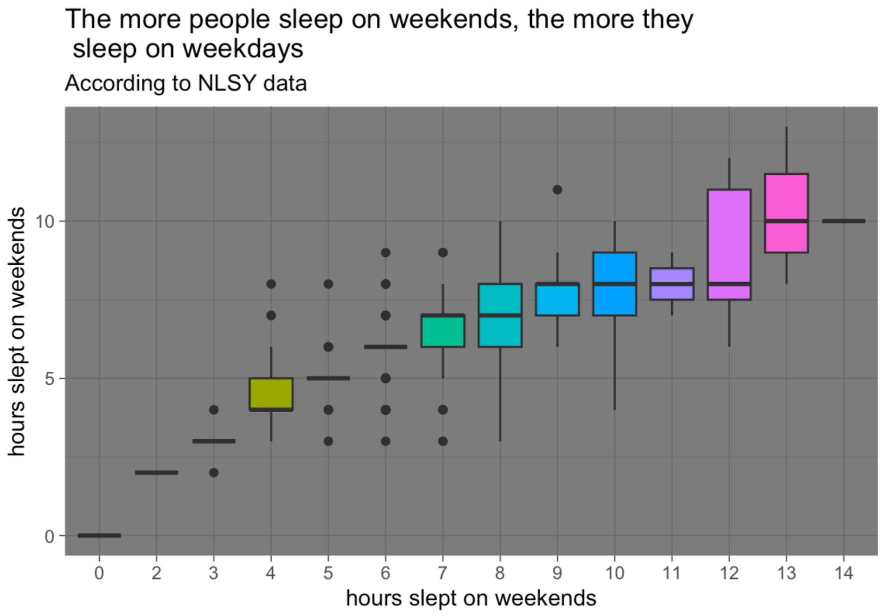
```r
p +
  theme_minimal()
```



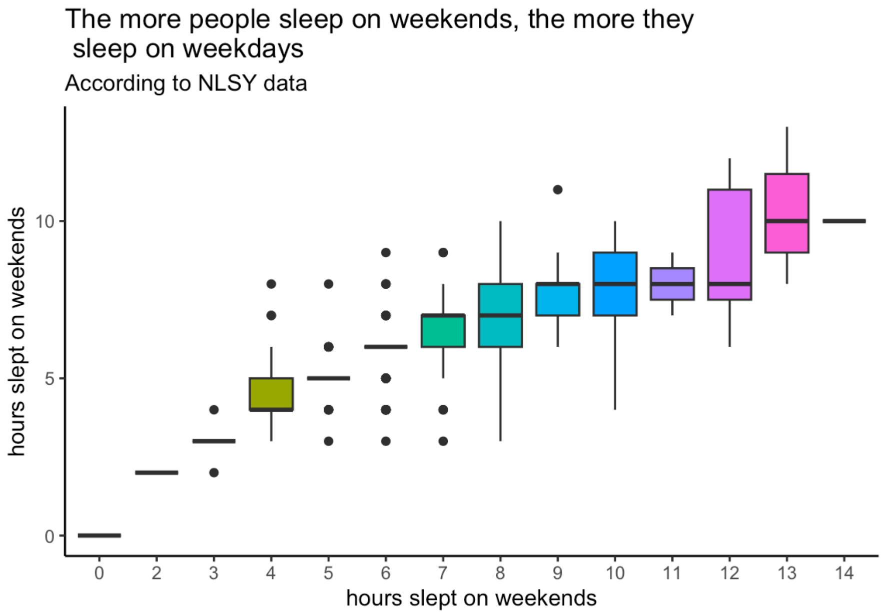
We can change the overall theme

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

```r
p +
  theme_dark()
```

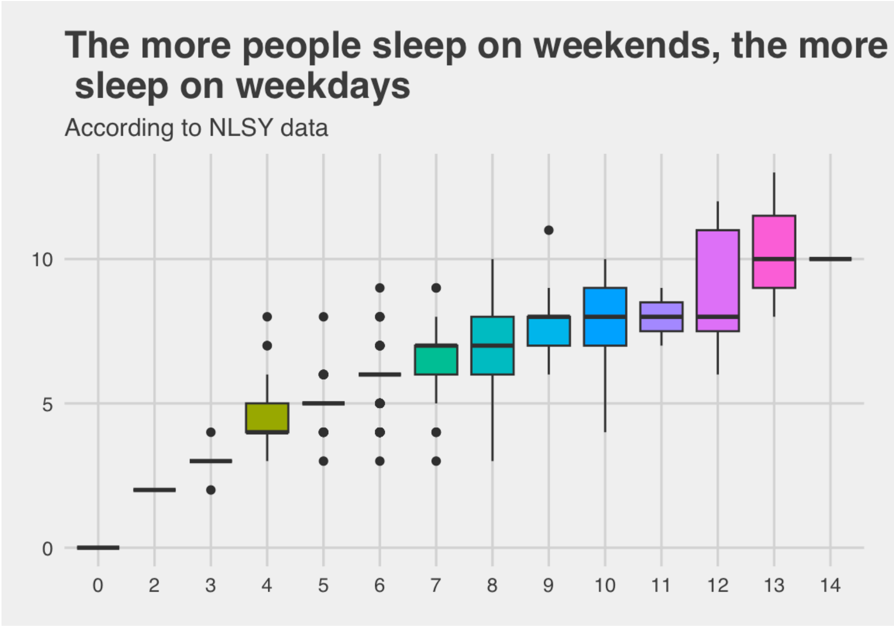


```r
p +
  theme_classic()
```



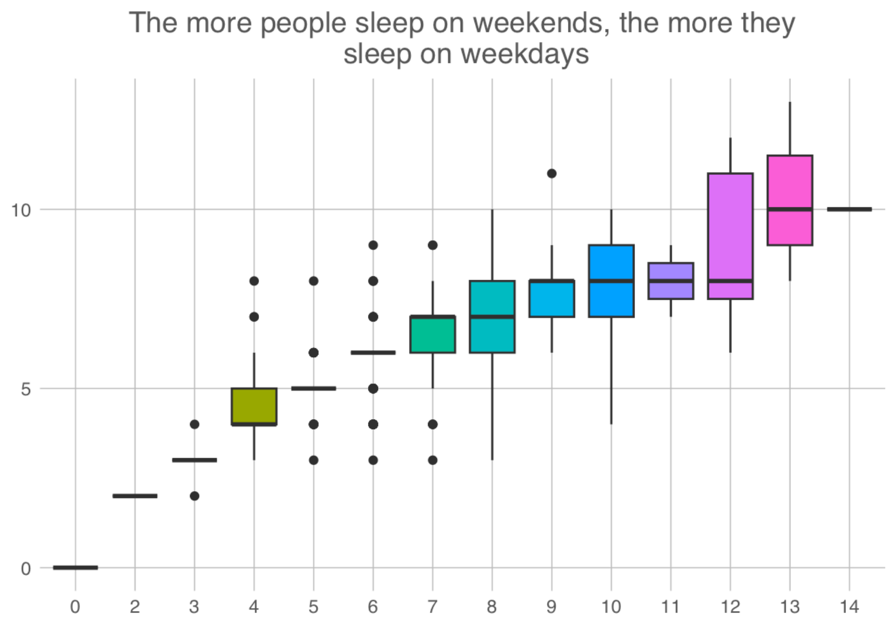
```r
p +
  theme_void()
```

```r
p +
  ggthemes::theme_fivethirtyei
```



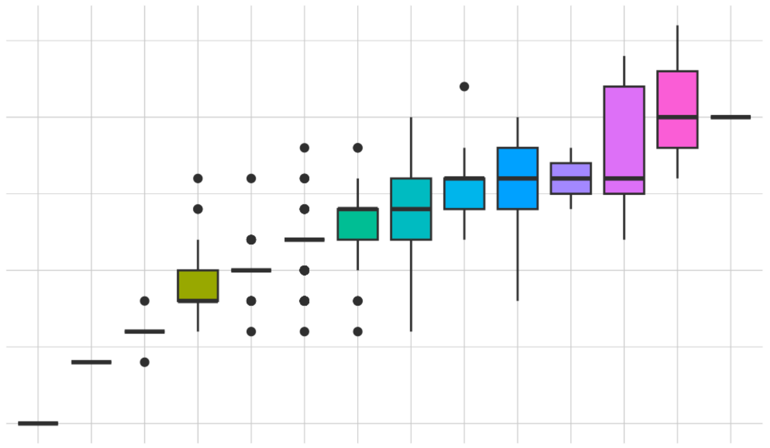
Other packages contain themes, too.

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



In case you miss Excel....

```r
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:

```r
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

```r
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/
- Book (includes theory behind ggplot): https://ggplot2-book.org/

Today's summary

- We learned how to use `janitor` and `gtsummary` packages to make tables
- We learned how to fit linear regressions and generlized linear models
- We learned how to use `broom` to tidy regression results
- We learned the basics of `ggplot2`

Today's functions

- `tabyl()`: create frequency tables
- `tbl_summary()`: create summary tables with a lot of covariates
- `lm()`: fit linear regression models
- `glm()`: fit generalized linear models
- `tbl_regression()`: create regression tables
- `tidy()`: tidy regression results
- `ggplot()`: create figures
- `geom_...()`: add a geometric feature to a figure
- `scale_...()`: change the scale of an axis or aesthetic
- `facet_...()`: split a figure into panels