Finer control over statistics

We fit a series of univariate regressions

```
income_table <- tbl_uv
  nlsy,
  y = income,
  include = c(
    sex_cat, race_eth_
    eyesight_cat, inco
  ),
  method = lm
)
income_table
```

Characteristic	N	Beta	95% CI[1]	p-value
sex_cat	10,195			
Male		—	—	
Female		-358	-844, 128	0.15
race_eth_cat	10,195			
Hispanic		—	—	
Black		-1,747	-2,507, -988	<0.001
Non-Black, Non-Hispanic		3,863	3,195, 4,530	<0.001
eyesight_cat	6,789			
Excellent		—	—	
Very good		-578	-1,319, 162	0.13
Good		-1,863	-2,719, -1,006	<0.001
Fair		-4,674	-5,910, -3,439	<0.001
Poor		-6,647	-9,154,	<0.001

[1] CI = Confidence Interval

Characteristic	N	Beta	95% CI [1]	p-value
			-4,140	
age_bir	4,773	595	538, 652	<0.001

[1] CI = Confidence Interval

But a table is a limited form of output

We might want to dig in a little more to those regressions

- One helpful option from `{gtsummary}` is to extract data from the table directly
- This can be reported in a manuscript (rather than copying and pasting from the table)

```r
inline_text(income_table, variable = "age_bir")
```

```
[1] "595 (95% CI 538, 652; p<0.001)"
```

We'll look at this again later!

What if we want *all* the numbers, say to create a figure?

- Under the hood, `{gtsummary}` is using the `{broom}` package to extract the statistics from the various models
- We can also use that package directly!

Statistical models in R can be messy

```r
mod_sex_cat <- lm(income ~ sex_cat, data = nlsy)
```

We could look at the model summary:

```r
summary(mod_sex_cat)
```

```

Call:
lm(formula = income ~ sex_cat, data = nlsy)

Residuals:
   Min      1Q Median      3Q     Max 
-14880  -8880  -3943   5477  60478 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)    14880.3      172.6  86.237   <2e-16 ***
sex_catFemale   -357.8      247.8  -1.444    0.149    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12510 on 10193 degrees of freedom
  (2491 observations deleted due to missingness)
```

Statistical models in R can be messy

If we want to do something with the various values, we could extract each statistic individually:

```r
coef(mod_sex_cat)
```

```
(Intercept) sex_catFemale 
 14880.3152     -357.8029 
```

```r
confint(mod_sex_cat)
```

```
                  2.5 %     97.5 %
(Intercept)   14542.079 15218.5512
sex_catFemale  -843.608   128.0022
```

```r
summary(mod_sex_cat)$r.squared
```

```
[1] 0.0002044429
```

```r
summary(mod_sex_cat)$coefficients
```

```
                Estimate Std. Error   t value  Pr(>|t|)
(Intercept)   14880.3152   172.5521 86.236672 0.0000000
sex_catFemale  -357.8029   247.8349 -1.443715 0.1488499
```

`{broom}` has three main functions: `augment()`, `glance()`, `tidy()`

`augment()` adds fitted values, residuals, and other statistics to the original data

```r
library(broom)
augment(mod_sex_cat)
```

```
# A tibble: 10,195 × 9
   .rownames income sex_cat .fitted  .resid      .hat .sigma   .cooksd .std.resid
   <chr>      <dbl> <fct>     <dbl>   <dbl>     <dbl>  <dbl>     <dbl>      <dbl>
 1 1          30000 Female   14523.  15477. 0.000202 12506.   1.55e-4      1.24
 2 2          20000 Female   14523.   5477. 0.000202 12507.   1.94e-5      0.438
 3 3          22390 Female   14523.   7867. 0.000202 12507.   4.01e-5      0.629
 4 4          22390 Female   14523.   7867. 0.000202 12507.   4.01e-5      0.629
 5 5          36000 Male     14880.  21120. 0.000190 12505.   2.72e-4      1.69
 6 6          35000 Male     14880.  20120. 0.000190 12505.   2.46e-4      1.61
 7 7           8502 Male     14880.  -6378. 0.000190 12507.   2.48e-5     -0.510
 8 8           7227 Female   14523.  -7296. 0.000202 12507.   3.44e-5     -0.583
 9 9          17000 Male     14880.   2120. 0.000190 12507.   2.74e-6      0.170
10 10          3548 Female   14523. -10975. 0.000202 12506.   7.79e-5     -0.878
# ℹ 10,185 more rows
```

`{broom}` has three main functions: `augment()`, `glance()`, `tidy()`

`glance()` creates a table of statistics that pertain to the entire model

```
glance(mod_sex_cat)
```

```
# A tibble: 1 × 12
  r.squared adj.r.squared  sigma statistic p.value    df   logLik     AIC     BIC
      <dbl>         <dbl>  <dbl>     <dbl>   <dbl> <dbl>    <dbl>   <dbl>   <dbl>
1  0.000204      0.000106 12506.      2.08   0.149     1 -110644. 221295. 2.21e5
# ℹ 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

`{broom}` has three main functions: `augment()`, `glance()`, `tidy()`

`tidy()` is the most useful to me and probably you!

It extracts coefficients and confidence intervals from models

```r
tidy(mod_sex_cat, conf.int = TRUE)
```

```
# A tibble: 2 × 7
  term          estimate std.error statistic p.value conf.low conf.high
  <chr>            <dbl>     <dbl>     <dbl>   <dbl>    <dbl>     <dbl>
1 (Intercept)     14880.      173.     86.2    0       14542.    15219.
2 sex_catFemale    -358.      248.     -1.44   0.149    -844.      128.
```

`tidy()` works on over 100 statistical methods in R!

Anova, ARIMA, Cox, factor analysis, fixed effects, GAM, GEE, IV, kappa, kmeans, multinomial, proportional odds, principal components, survey methods, ...

- See the full list here
- All the output shares column names
- This makes it really easy to work with the output and reuse code across analyses

Some models have additional arguments

For example, we might want exponentiated coefficients:

```r
logistic_model <- glm(glasses ~ eyesight_cat + sex_cat + i
                      data = nlsy, family = binomial())
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)              0.499   5.96e-2   -11.7   1.74e-31    0.444     0.560
2 eyesight_catVery good    0.920   5.96e-2    -1.39  1.64e- 1    0.819     1.03
3 eyesight_catGood         0.916   6.91e-2    -1.27  2.04e- 1    0.800     1.05
4 eyesight_catFair         0.802   1.00e-1    -2.20  2.77e- 2    0.658     0.976
5 eyesight_catPoor         1.03    2.01e-1     0.147 8.83e- 1    0.694     1.53
6 sex_catFemale            2.04    5.00e-2    14.2   5.46e-46    1.85      2.25
7 income                   1.00    1.93e-6     7.49  6.95e-14    1.00      1.00
```

We can also combine the results of lots of regressions

```
# we already made mod_sex_cat
mod_race_eth_cat <- lm(income ~ race_eth_cat, data = nlsy)
mod_eyesight_cat <- lm(income ~ eyesight_cat, data = nlsy)
mod_age_bir <- lm(income ~ age_bir, data = nlsy)

tidy_sex_cat <- tidy(mod_sex_cat, conf.int = TRUE)
tidy_race_eth_cat <- tidy(mod_race_eth_cat, conf.int = TRU
tidy_eyesight_cat <- tidy(mod_eyesight_cat, conf.int = TRU
tidy_age_bir <- tidy(mod_age_bir, conf.int = TRUE)
```

There are of course more efficient ways to do this instead of copy/pasting 4 times…

With a little finagling, we have the same data as in the original univartiate regression table...

```r
dplyr::bind_rows(
  sex_cat = tidy_sex_cat,
  race_eth_cat = tidy_race_eth_cat,
  eyesight_cat = tidy_eyesight_cat,
  age_bir = tidy_age_bir, .id = "model") |>
  dplyr::mutate(
    term = stringr::str_remove(term, model),
    term = ifelse(term == "", model, term))
```

With a little finagling, we have the same data as in the original univartiate regression table…

```
# A tibble: 12 × 8
   model        term     estimate std.error statistic  p.value conf.low conf.high
   <chr>        <chr>       <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
 1 sex_cat      (Inter…    14880.     173.      86.2  0           14542.    15219.
 2 sex_cat      Female      -358.     248.      -1.44 1.49e- 1     -844.      128.
 3 race_eth_cat (Inter…    12867.     302.      42.7  0           12276.    13459.
 4 race_eth_cat Black      -1747.     387.      -4.51 6.58e- 6    -2507.     -988.
 5 race_eth_cat Non-Bl…     3863.     341.      11.3  1.20e-29     3195.     4530.
 6 eyesight_cat (Inter…    17683.     270.      65.6  0           17155.    18212.
 7 eyesight_cat Very g…     -578.     378.      -1.53 1.26e- 1    -1319.      162.
 8 eyesight_cat Good       -1863.     437.      -4.26 2.05e- 5    -2719.    -1006.
 9 eyesight_cat Fair       -4674.     630.      -7.42 1.35e-13    -5910.    -3439.
10 eyesight_cat Poor       -6647.    1279.      -5.20 2.07e- 7    -9154.    -4140.
11 age_bir      (Inter…     1707.     733.       2.33 1.99e- 2      270.     3143.
12 age_bir      age_bir      595.      29.1     20.4  3.71e-89      538.      652.
```

Even easier cleanup!

We could instead clean up the names and add reference rows with the `{tidycat}` package:

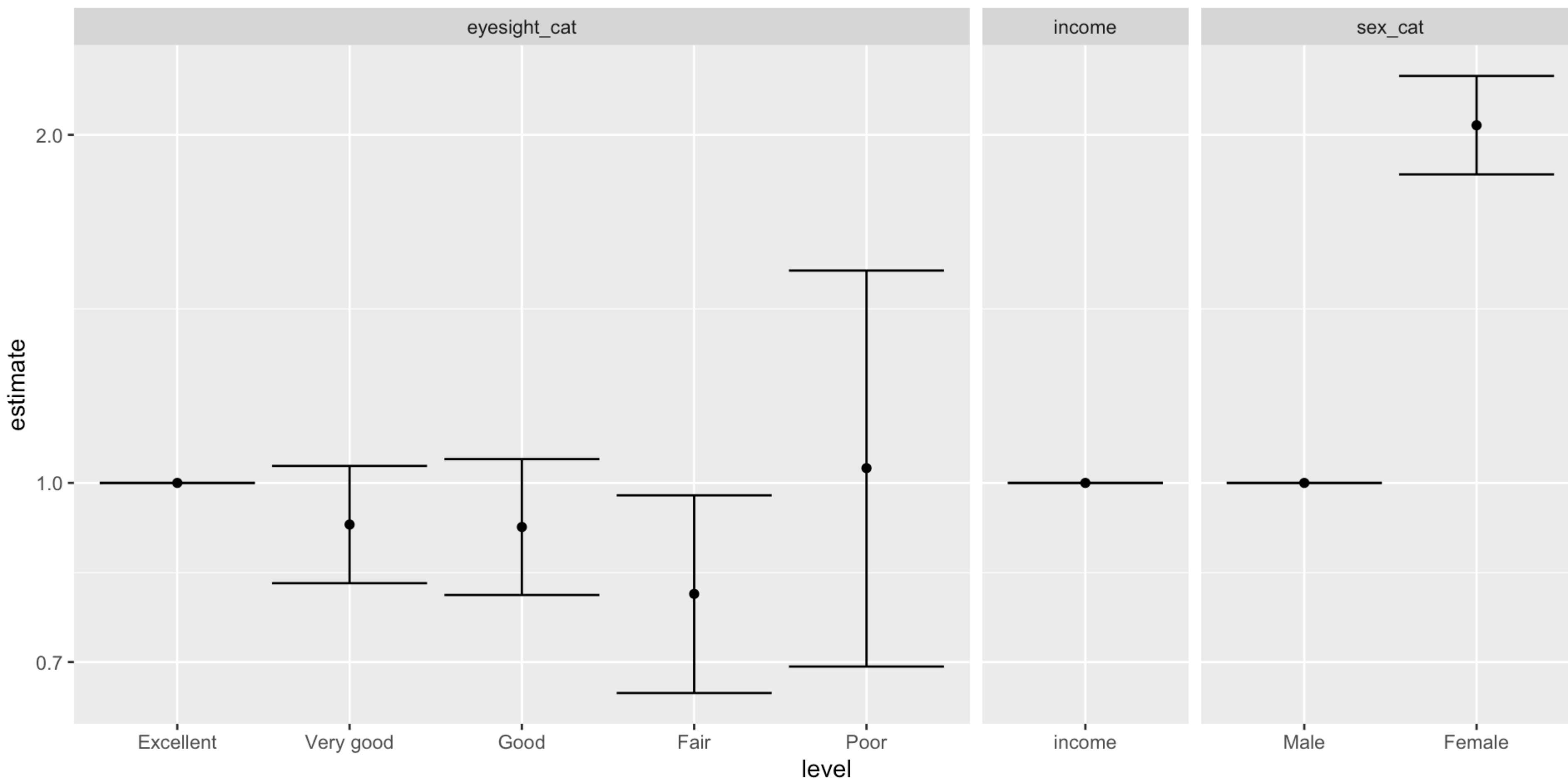
```r
tidy(logistic_model, conf.int = TRUE, exponentiate = TRUE)
  tidycat::tidy_categorical(logistic_model, exponentiate =
  dplyr::select(-c(3:5))
```

```
# A tibble: 9 × 8
  term                estimate conf.low conf.high variable level effect reference
  <chr>                  <dbl>    <dbl>     <dbl> <chr>    <fct> <chr>  <chr>
1 (Intercept)            0.499    0.444     0.560 (Interc… (Int… main   Non-Base…
2 <NA>                   1        1         1     eyesigh… Exce… main   Baseline…
3 eyesight_catVery …     0.920    0.819     1.03  eyesigh… Very… main   Non-Base…
4 eyesight_catGood       0.916    0.800     1.05  eyesigh… Good  main   Non-Base…
5 eyesight_catFair       0.802    0.658     0.976 eyesigh… Fair  main   Non-Base…
6 eyesight_catPoor       1.03     0.694     1.53  eyesigh… Poor  main   Non-Base…
7 <NA>                   1        1         1     sex_cat  Male  main   Baseline…
8 sex_catFemale          2.04     1.85      2.25  sex_cat  Fema… main   Non-Base…
9 income                 1.00     1.00      1.00  income   inco… main   Non-Base…
```

This makes it easy to make forest plots, for example

```
library(ggplot2)
tidy(logistic_model, conf.int = TRUE, exponentiate = TRUE)
  tidycat::tidy_categorical(logistic_model, exponentiate =
  dplyr::slice(-1) |> # remove intercept
  ggplot(mapping = aes(x = level, y = estimate,
                       ymin = conf.low, ymax = conf.high))
  geom_point() +
  geom_errorbar() +
  facet_grid(cols = vars(variable), scales = "free", space
  scale_y_log10()
```

This makes it easy to make forest plots, for example



Exercises

1. Open the script with these examples.
2. Run it.
3. Teach yourself to use `broom::tidy()` to extract the results of the Poisson regression with robust standard errors and combine them with the results of the log-binomial regression.
4. Start creating some tables for your final project!

15:00