

# Work with Models

Data Visualization: Session 6  
Kieran Healy  
Code Horizons, April 2023

# Load our libraries

```
library(here)      # manage file paths
library(socviz)    # data and some useful functions
library(tidyverse) # your friend and mine
library(gapminder) # Everyone's favorite dataset
library(broom)     # Tidy model output
library(marginaleffects) # Tidy marginal effects
library(modelsummary) # Tidy summary tables and graphs
library(scales)    # Format our axes and guides

## 
## Attaching package: 'scales'
## The following object is masked from 'package:purrr':
##   discard
## The following object is masked from 'package:readr':
##   col_factor
```

# We know `ggplot` can work with models

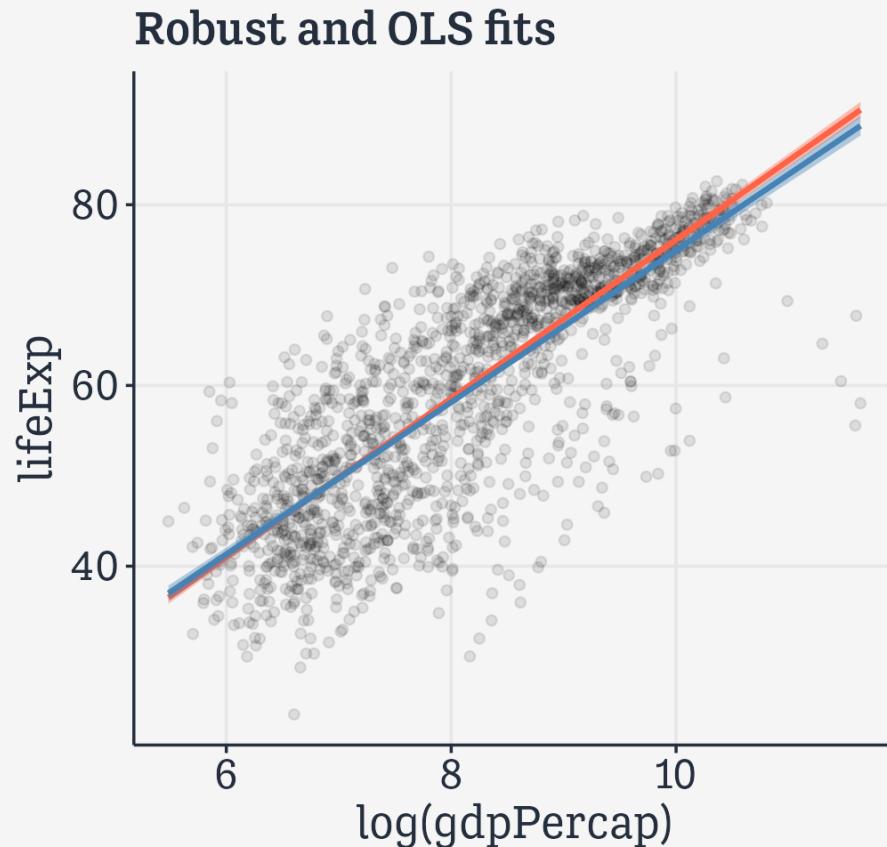
```
p ← gapminder ▷  
  ggplot(mapping = aes(x = log(gdpPercap),  
                        y = lifeExp))  
  
p + geom_point(alpha=0.1) +  
  geom_smooth(color = "tomato",  
              fill="tomato",  
              method = MASS::rlm) +  
  geom_smooth(color = "steelblue",  
              fill="steelblue",  
              method = "lm") +  
  labs(title = "Robust and OLS fits")
```

We know because geoms often do calculations in the background, via their `stat` functions.

# We know `ggplot` can work with models

```
p ← gapminder %>%  
  ggplot(mapping = aes(x = log(gdpPercap),  
                      y = lifeExp))  
  
p + geom_point(alpha=0.1) +  
  geom_smooth(color = "tomato",  
              fill="tomato",  
              method = MASS::rlm) +  
  geom_smooth(color = "steelblue",  
              fill="steelblue",  
              method = "lm") +  
  labs(title = "Robust and OLS fits")
```

We know because geoms often do calculations in the background, via their `stat` functions.

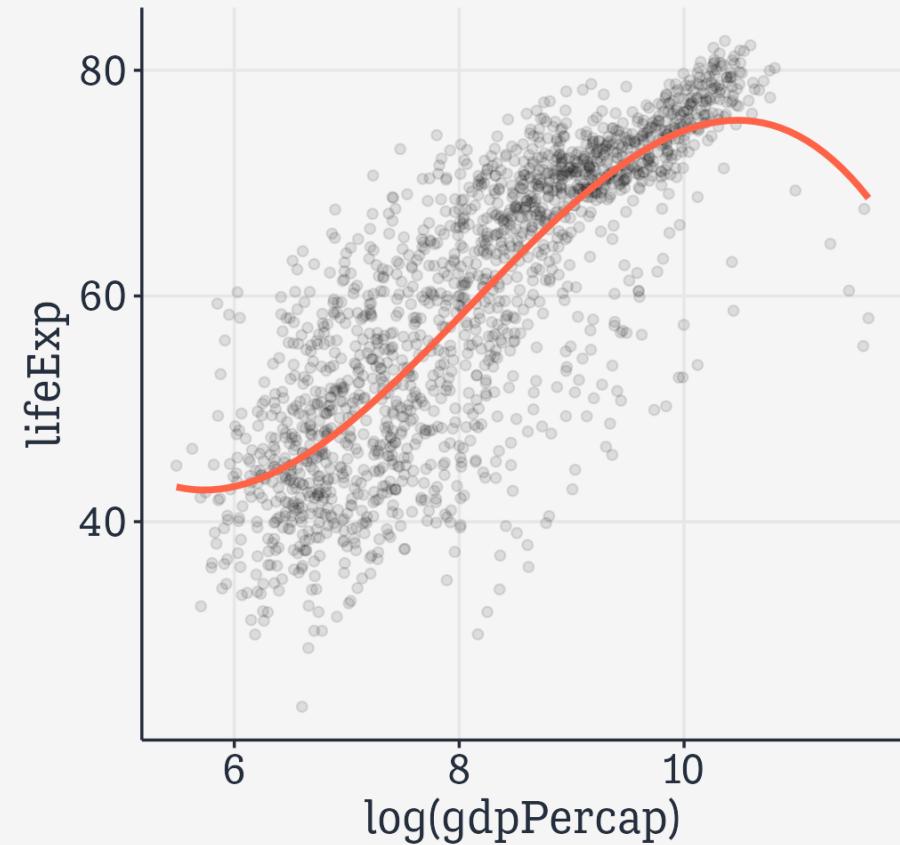


# And these can be complex ...

```
p + geom_point(alpha=0.1) +  
  geom_smooth(color = "tomato",  
              method = "lm",  
              size = 1.2,  
              formula = y ~ splines::bs(x, 3),  
              se = FALSE)
```

# And these can be complex ...

```
p + geom_point(alpha=0.1) +  
  geom_smooth(color = "tomato",  
              method = "lm",  
              size = 1.2,  
              formula = y ~ splines::bs(x, 3),  
              se = FALSE)
```

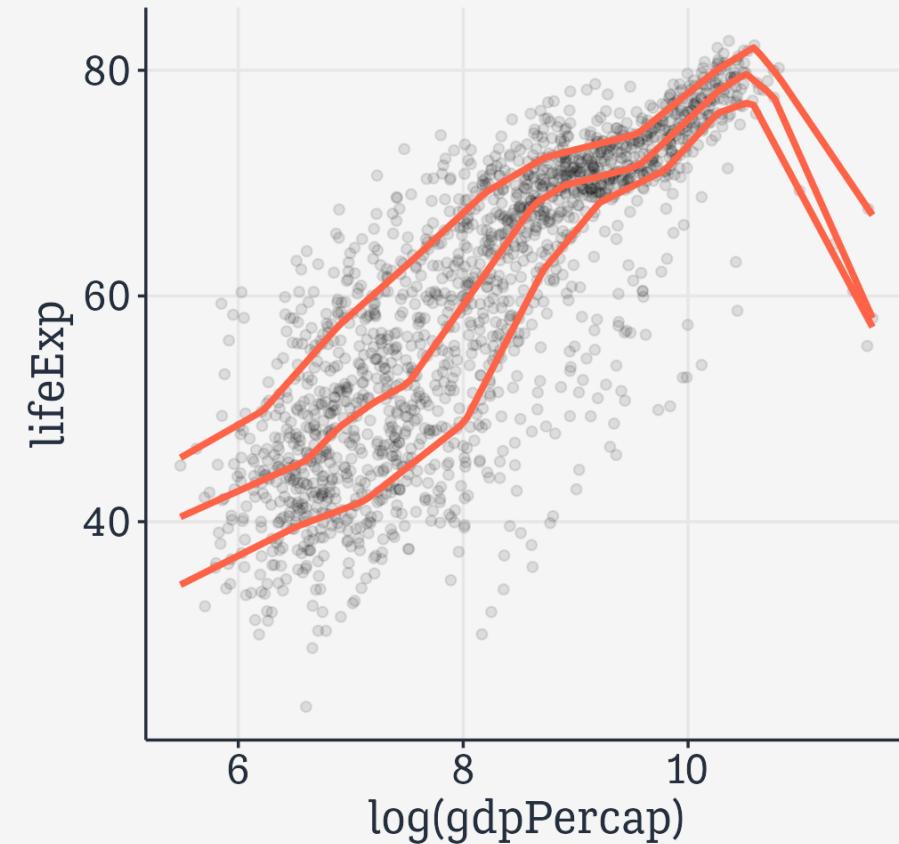


# ... but we usually won't do this inside ggplot

```
p + geom_point(alpha=0.1) +  
  geom_quantile(color = "tomato",  
    size = 1.2,  
    method = "rqss",  
    lambda = 1,  
    quantiles = c(0.20, 0.5, 0.85))
```

# ... but we usually won't do this inside ggplot

```
p + geom_point(alpha=0.1) +  
  geom_quantile(color = "tomato",  
    size = 1.2,  
    method = "rqss",  
    lambda = 1,  
    quantiles = c(0.20, 0.5, 0.85))
```



**Transform and  
summarize first.**

**Then send your clean  
tables to ggplot.**

**Look inside  
the box**

# Again, objects are Bento Boxes of To-Do lists

gapminder

```
## # A tibble: 1,704 × 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>  <dbl>     <int>     <dbl>
## 1 Afghanistan Asia      1952    28.8  8425333    779.
## 2 Afghanistan Asia      1957    30.3  9240934    821.
## 3 Afghanistan Asia      1962    32.0  10267083   853.
## 4 Afghanistan Asia      1967    34.0  11537966   836.
## 5 Afghanistan Asia      1972    36.1  13079460   740.
## 6 Afghanistan Asia      1977    38.4  14880372   786.
## 7 Afghanistan Asia      1982    39.9  12881816   978.
## 8 Afghanistan Asia      1987    40.8  13867957   852.
## 9 Afghanistan Asia      1992    41.7  16317921   649.
## 10 Afghanistan Asia     1997    41.8  22227415   635.
## # i 1,694 more rows
```

# Fit a model

```
out ← lm(formula = lifeExp ~ gdpPercap + log(pop) + continent,
         data = gapminder)

summary(out)

##
## Call:
## lm(formula = lifeExp ~ gdpPercap + log(pop) + continent, data = gapminder)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -47.490  -4.614   0.250   5.293  26.094 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3.816e+01  2.050e+00 18.618 < 2e-16 ***  
## gdpPercap   4.557e-04  2.345e-05 19.435 < 2e-16 ***  
## log(pop)    6.394e-01  1.329e-01  4.810 1.64e-06 ***  
## continentAmericas 1.308e+01  6.063e-01 21.579 < 2e-16 ***  
## continentAsia    7.784e+00  5.810e-01 13.398 < 2e-16 ***  
## continentEurope   1.695e+01  6.350e-01 26.691 < 2e-16 ***  
## continentOceania  1.764e+01  1.779e+00  9.916 < 2e-16 ***  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.336 on 1697 degrees of freedom
## Multiple R-squared:  0.585,    Adjusted R-squared:  0.5835 
## F-statistic: 398.7 on 6 and 1697 DF,  p-value: < 2.2e-16
```

# Poke around inside

Use the Object Inspector to take a look

`out`

```
  └── coefficients
  └── residuals
  └── effects
  └── rank
  └── qr
      └── qr
      └── pivot
      └── qraux
      └── tol
      └── rank
  └── df.residual
  └── contrasts
  └── xlevels
  └── call
  └── terms
  └── model.frame
```

# **Predict from models: DIY method**

# Behind the curtain

**predict()** and its methods do a lot of work behind the scenes

We won't usually need to do this stuff manually. But the idea is that the generic **predict()** function has *methods* for specific sorts of models. Give it a model and some new data and it will produce predicted values for the new data. Here's an example.

# The labor-intensive way

```
min_gdp ← min(gapminder$gdpPercap)
max_gdp ← max(gapminder$gdpPercap)
med_pop ← median(gapminder$pop)

# Make a grid of predictor values
pred_df ← expand_grid(gdpPercap = seq(from = min_gdp,
                                         to = max_gdp,
                                         length.out = 100)),
            pop = med_pop,
            continent = c("Africa", "Americas",
                          "Asia", "Europe", "Oceania"))

pred_df

## # A tibble: 500 × 3
##   gdpPercap      pop continent
##       <dbl>    <dbl> <chr>
## 1     241.  7023596. Africa
## 2     241.  7023596. Americas
## 3     241.  7023596. Asia
## 4     241.  7023596. Europe
## 5     241.  7023596. Oceania
## 6    1385.  7023596. Africa
## 7    1385.  7023596. Americas
## 8    1385.  7023596. Asia
## 9    1385.  7023596. Europe
## 10   1385.  7023596. Oceania
## # i 490 more rows
```

# The labor-intensive way

```
# Get the predicted values
pred_out ← predict(object = out,
                     newdata = pred_df,
                     interval = "confidence")
head(pred_out)

##      fit     lwr     upr
## 1 48.35388 47.67735 49.03041
## 2 61.43646 60.43917 62.43375
## 3 56.13821 55.22045 57.05597
## 4 65.30361 64.21794 66.38927
## 5 65.99517 62.55277 69.43757
## 6 48.87530 48.20261 49.54799
```

# The labor-intensive way

```
# Bind them into one data frame. We can do this safely
# here because we know the row order by construction.
# But this is not a safe approach in general.

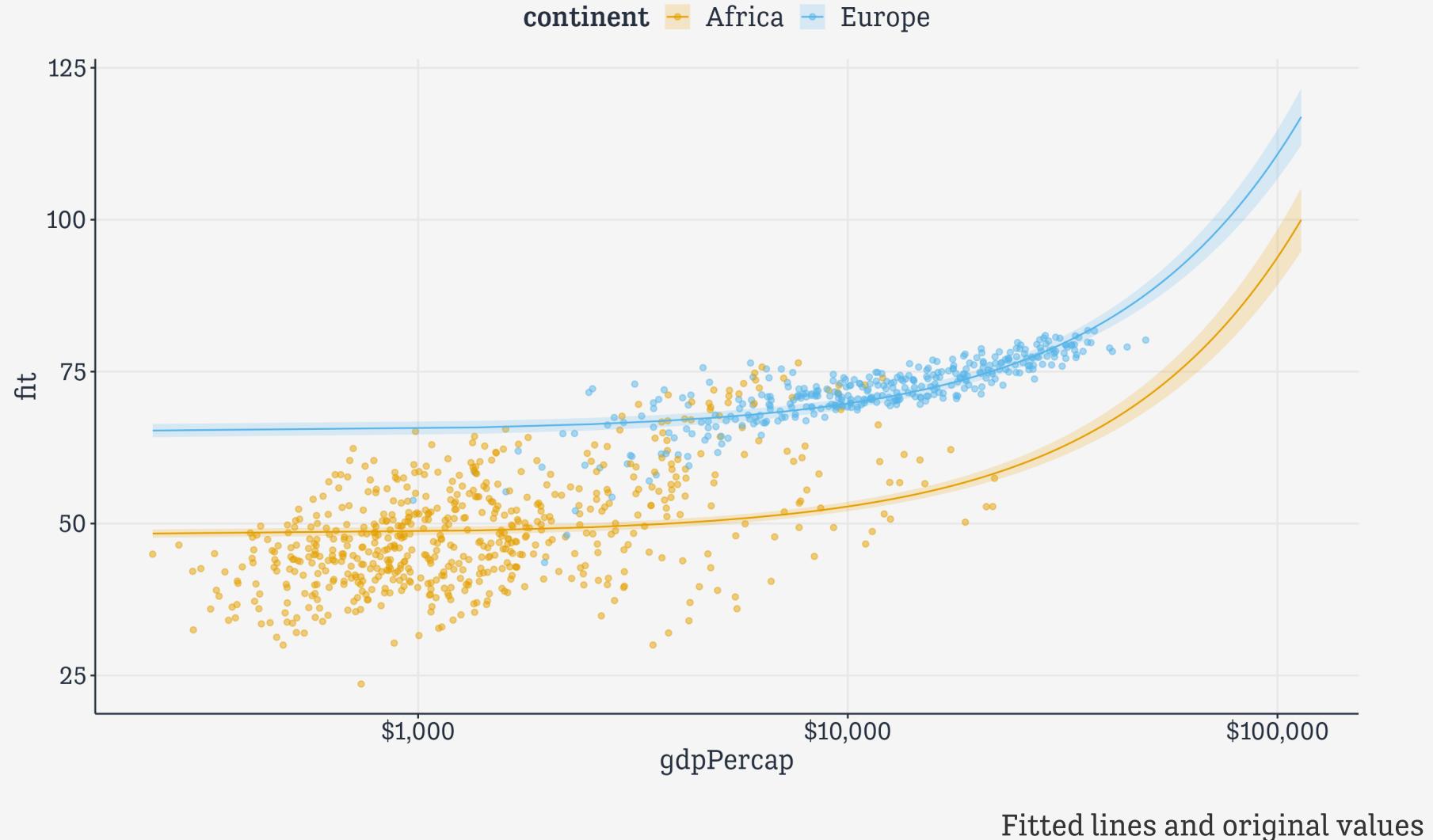
pred_df ← cbind(pred_df, pred_out)
head(pred_df)

##   gdpPercap     pop continent      fit      lwr      upr
## 1 241.1659 7023596    Africa 48.35388 47.67735 49.03041
## 2 241.1659 7023596   Americas 61.43646 60.43917 62.43375
## 3 241.1659 7023596      Asia 56.13821 55.22045 57.05597
## 4 241.1659 7023596   Europe 65.30361 64.21794 66.38927
## 5 241.1659 7023596  Oceania 65.99517 62.55277 69.43757
## 6 1385.4282 7023596    Africa 48.87530 48.20261 49.54799
```

# The labor-intensive way

```
p <- ggplot(data = subset(pred_df, continent %in% c("Europe", "Africa")),
  aes(x = gdpPercap,
      y = fit,
      ymin = lwr,
      ymax = upr,
      color = continent,
      fill = continent,
      group = continent))

# Use the original data as the point layer
p_out <- p + geom_point(data = subset(gapminder,
  continent %in% c("Europe", "Africa")),
  mapping = aes(x = gdpPercap, y = lifeExp,
    color = continent),
  alpha = 0.5,
  inherit.aes = FALSE) +
# And the predicted values to draw the lines
  geom_line() +
  geom_ribbon(alpha = 0.2, color = FALSE) +
  scale_x_log10(labels = scales::label_dollar())
```



**Use broom to tidy  
models**

# We can't do anything with this

```
out ← lm(formula = lifeExp ~ gdpPercap + log(pop) + continent,
         data = gapminder)

summary(out)

##
## Call:
## lm(formula = lifeExp ~ gdpPercap + log(pop) + continent, data = gapminder)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -47.490  -4.614   0.250   5.293  26.094 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3.816e+01  2.050e+00 18.618 < 2e-16 ***  
## gdpPercap   4.557e-04  2.345e-05 19.435 < 2e-16 ***  
## log(pop)    6.394e-01  1.329e-01  4.810 1.64e-06 ***  
## continentAmericas 1.308e+01  6.063e-01 21.579 < 2e-16 ***  
## continentAsia    7.784e+00  5.810e-01 13.398 < 2e-16 ***  
## continentEurope   1.695e+01  6.350e-01 26.691 < 2e-16 ***  
## continentOceania  1.764e+01  1.779e+00  9.916 < 2e-16 ***  
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
##
## Residual standard error: 8.336 on 1697 degrees of freedom
## Multiple R-squared:  0.585,    Adjusted R-squared:  0.5835 
## F-statistic: 398.7 on 6 and 1697 DF,  p-value: < 2.2e-16
```

# Tidy regression output with **broom**

```
library(broom)

tidy(out)

## # A tibble: 7 × 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 38.2      2.05     18.6  1.50e- 70
## 2 gdpPercap    0.000456  0.0000234  19.4  3.98e- 76
## 3 log(pop)     0.639     0.133     4.81  1.64e-  6
## 4 continentAmericas 13.1     0.606    21.6  1.85e- 91
## 5 continentAsia    7.78     0.581     13.4  5.52e- 39
## 6 continentEurope   16.9     0.635    26.7  2.43e-131
## 7 continentOceania  17.6     1.78     9.92  1.43e- 22
```

That's a *lot* nicer. Now it's just a tibble. We know those.

# Tidy regression output with broom

```
out_conf ← tidy(out, conf.int = TRUE)
out_conf

## # A tibble: 7 × 7
##   term          estimate std.error statistic  p.value conf.low conf.high
##   <chr>        <dbl>     <dbl>      <dbl>    <dbl>    <dbl>     <dbl>
## 1 (Intercept) 38.2      2.05      18.6  1.50e- 70 34.1      42.2
## 2 gdpPercap    0.000456 0.0000234  19.4  3.98e- 76 0.000410  0.000502
## 3 log(pop)     0.639     0.133      4.81  1.64e- 6  0.379      0.900
## 4 continentAmericas 13.1     0.606      21.6  1.85e- 91 11.9      14.3
## 5 continentAsia  7.78      0.581      13.4  5.52e- 39 6.64      8.92
## 6 continentEurope 16.9      0.635      26.7  2.43e-131 15.7      18.2
## 7 continentOceania 17.6      1.78       9.92  1.43e- 22 14.2      21.1
```

# Tidy regression output with broom

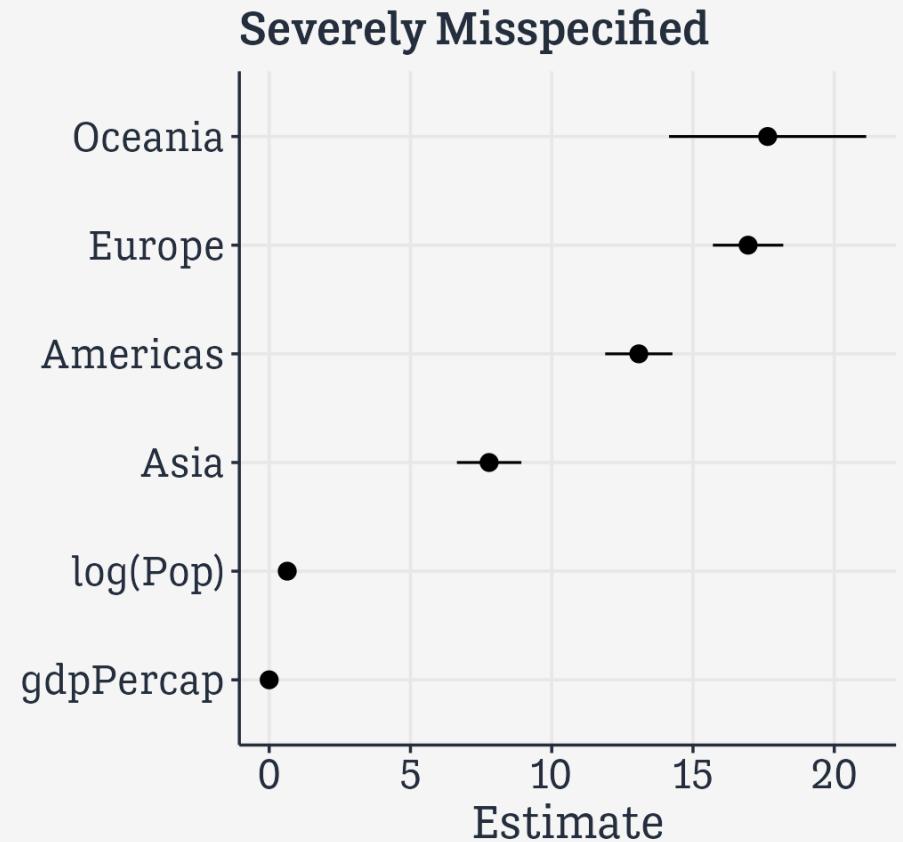
```
out_conf %>  
  filter(term %nin% "(Intercept)") %>  
  mutate(nicelabs = prefix_strip(term, "continent")) %>  
  relocate(nicelabs)  
  
## # A tibble: 6 × 8  
##   nicelabs term      estimate std.error statistic p.value conf.low conf.high  
##   <chr>     <chr>      <dbl>     <dbl>      <dbl>    <dbl>     <dbl>     <dbl>  
## 1 gdpPercap gdpPercap  4.56e-4  0.0000234     19.4  3.98e- 76  4.10e-4  0.000502  
## 2 log(Pop)   log(pop)   6.39e-1  0.133        4.81  1.64e- 6  3.79e-1  0.900  
## 3 Americas   continent... 1.31e+1  0.606       21.6  1.85e- 91  1.19e+1  14.3  
## 4 Asia       continent... 7.78e+0  0.581       13.4  5.52e- 39  6.64e+0  8.92  
## 5 Europe     continent... 1.69e+1  0.635       26.7  2.43e-131 1.57e+1  18.2  
## 6 Oceania    continent... 1.76e+1  1.78        9.92 1.43e- 22  1.42e+1  21.1
```

# Tidy regression output with broom

```
out_conf %>  
  filter(term %nin% "(Intercept)") %>  
  mutate(nicelabs = prefix_strip(term, "continent")) %>  
  ggplot(mapping = aes(x = estimate,  
                        xmin = conf.low,  
                        xmax = conf.high,  
                        y = reorder(nicelabs,  
                                    estimate))) +  
  geom_pointrange() +  
  labs(x = "Estimate",  
       y = NULL,  
       title = "Severely Misspecified")
```

# Tidy regression output with **broom**

```
out_conf %>  
  filter(term %nin% "(Intercept)") %>  
  mutate(nicelabs = prefix_strip(term, "continent")) %>  
  ggplot(mapping = aes(x = estimate,  
                        xmin = conf.low,  
                        xmax = conf.high,  
                        y = reorder(nicelabs,  
                                    estimate))) +  
  
  geom_pointrange() +  
  labs(x = "Estimate",  
       y = NULL,  
       title = "Severely Misspecified")
```



# Three ways to tidy

Component level: `tidy()`

Observation level: `augment()`

Model level: `glance()`

# Component level

```
> summary(out)

Call:
lm(formula = lifeExp ~ gdpPercap + log(pop) + continent, data = gapminder)

Residuals:
    Min      1Q  Median      3Q     Max 
-47.490 -4.614  0.250  5.293 26.094 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 3.816e+01 2.050e+00 18.618 < 2e-16 ***  
gdpPercap   4.557e-04 2.345e-05 19.435 < 2e-16 ***  
log(pop)    6.394e-01 1.329e-01  4.810 1.64e-06 ***  
continentAmericas 1.308e+01 6.063e-01 21.579 < 2e-16 ***  
continentAsia  7.784e+00 5.810e-01 13.398 < 2e-16 ***  
continentEurope 1.695e+01 6.350e-01 26.691 < 2e-16 ***  
continentOceania 1.764e+01 1.779e+00  9.916 < 2e-16 ***  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.336 on 1697 degrees of freedom
Multiple R-squared:  0.585,    Adjusted R-squared:  0.5835 
F-statistic: 398.7 on 6 and 1697 DF,  p-value: < 2.2e-16
```

# Observation level

```
> summary(out)

Call:
lm(formula = lifeExp ~ gdpPercap + log(pop) + continent, data = gapminder)

Residuals:
    Min      1Q  Median      3Q     Max 
-47.490 -4.614  0.250  5.293 26.094 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 3.816e+01 2.050e+00 18.618 < 2e-16 ***  
gdpPercap   4.557e-04 2.345e-05 19.435 < 2e-16 ***  
log(pop)    6.394e-01 1.329e-01  4.810 1.64e-06 ***  
continentAmericas 1.308e+01 6.063e-01 21.579 < 2e-16 ***  
continentAsia    7.784e+00 5.810e-01 13.398 < 2e-16 ***  
continentEurope   1.695e+01 6.350e-01 26.691 < 2e-16 ***  
continentOceania   1.764e+01 1.779e+00  9.916 < 2e-16 ***  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.336 on 1697 degrees of freedom
Multiple R-squared:  0.585,    Adjusted R-squared:  0.5835 
F-statistic: 398.7 on 6 and 1697 DF,  p-value: < 2.2e-16
```

# Observation level

```
augment(out)

## # A tibble: 1,704 × 10
##   lifeExp gdpPercap `log(pop)` continent .fitted .resid     .hat .sigma .cooksdi
##   <dbl>      <dbl>      <dbl> <fct>       <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 28.8       779.     15.9 Asia       56.5   -27.7  0.00302   8.31  0.00479
## 2 30.3       821.     16.0 Asia       56.6   -26.2  0.00299   8.31  0.00426
## 3 32.0       853.     16.1 Asia       56.7   -24.7  0.00296   8.32  0.00372
## 4 34.0       836.     16.3 Asia       56.7   -22.7  0.00294   8.32  0.00313
## 5 36.1       740.     16.4 Asia       56.8   -20.7  0.00294   8.32  0.00259
## 6 38.4       786.     16.5 Asia       56.9   -18.4  0.00292   8.33  0.00205
## 7 39.9       978.     16.4 Asia       56.9   -17.0  0.00291   8.33  0.00174
## 8 40.8       852.     16.4 Asia       56.9   -16.0  0.00292   8.33  0.00155
## 9 41.7       649.     16.6 Asia       56.9   -15.2  0.00294   8.33  0.00140
## 10 41.8      635.     16.9 Asia      57.1   -15.3  0.00297   8.33  0.00144
## # i 1,694 more rows
## # i 1 more variable: .std.resid <dbl>
```

# Observation level

For OLS models:

`.fitted` – The fitted values of the model.

`.se.fit` – The standard errors of the fitted values.

`.resid` – The residuals.

`.hat` – The diagonal of the hat matrix.

`.sigma` – An estimate of the residual standard deviation when the corresponding observation is dropped from the model.

`.cooksdist` – Cook's distance, a common regression diagnostic.

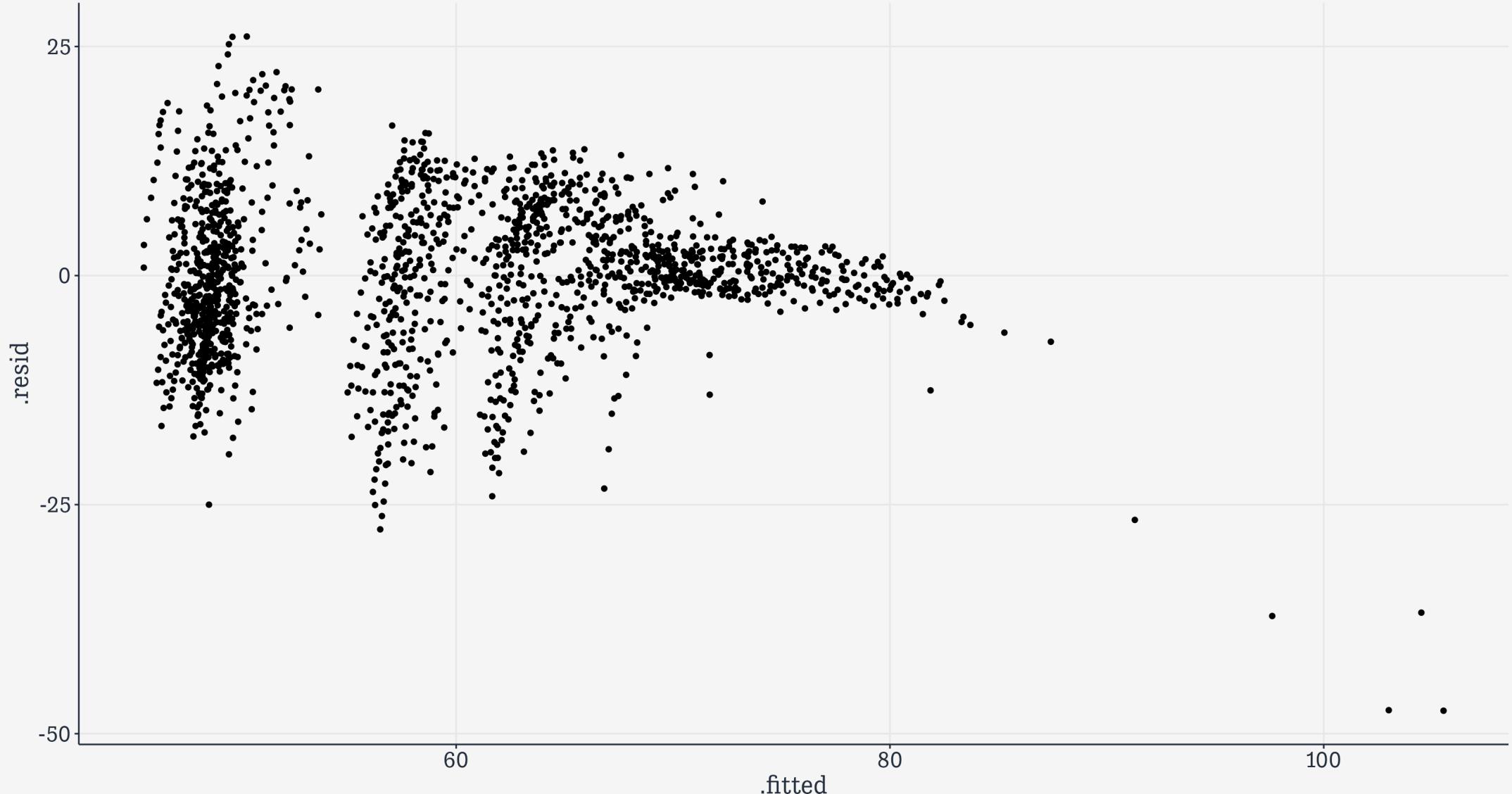
`.std.resid` – The standardized residuals.

# Observation level

```
# Adding the data argument puts back any additional columns from the original
# tibble
out_aug ← augment(out, data = gapminder)
head(out_aug)

## # A tibble: 6 × 12
##   country continent  year lifeExp     pop gdpPercap .fitted .resid     .hat .sigma
##   <fct>    <fct>    <int>   <dbl>   <int>      <dbl>   <dbl>   <dbl>   <dbl>
## 1 Afghan... Asia      1952    28.8 8.43e6     779.    56.5  -27.7  0.00302  8.31
## 2 Afghan... Asia      1957    30.3 9.24e6     821.    56.6  -26.2  0.00299  8.31
## 3 Afghan... Asia      1962    32.0 1.03e7     853.    56.7  -24.7  0.00296  8.32
## 4 Afghan... Asia      1967    34.0 1.15e7     836.    56.7  -22.7  0.00294  8.32
## 5 Afghan... Asia      1972    36.1 1.31e7     740.    56.8  -20.7  0.00294  8.32
## 6 Afghan... Asia      1977    38.4 1.49e7     786.    56.9  -18.4  0.00292  8.33
## # i 2 more variables: .cooksdf <dbl>, .std.resid <dbl>

## Residuals vs Fitted Values
p ← ggplot(data = out_aug,
            mapping = aes(x = .fitted, y = .resid))
p_out ← p + geom_point()
```



I told you it was misspecified

# Model level

```
> summary(out)

Call:
lm(formula = lifeExp ~ gdpPercap + log(pop) + continent, data = gapminder)

Residuals:
    Min      1Q  Median      3Q     Max 
-47.490 -4.614   0.250   5.293  26.094 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 3.816e+01  2.050e+00 18.618 < 2e-16 ***  
gdpPercap   4.557e-04  2.345e-05 19.435 < 2e-16 ***  
log(pop)    6.394e-01  1.329e-01  4.810 1.64e-06 ***  
continentAmericas 1.308e+01  6.063e-01 21.579 < 2e-16 ***  
continentAsia    7.784e+00  5.810e-01 13.398 < 2e-16 ***  
continentEurope   1.695e+01  6.350e-01 26.691 < 2e-16 ***  
continentOceania   1.764e+01  1.779e+00  9.916 < 2e-16 ***  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.336 on 1697 degrees of freedom
Multiple R-squared:  0.585,    Adjusted R-squared:  0.5835 
F-statistic: 398.7 on 6 and 1697 DF,  p-value: < 2.2e-16
```

# Model level

```
glance(out)

## # 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>
## 1     0.585        0.584  8.34     399. 1.01e-319      6 -6028. 12072. 12115.
## # i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

The usefulness of `glance()` becomes clearer when dealing with ensembles of models.

# Another example

# A Kaplan-Meier Curve

```
library(survival)
```

```
head(lung)
```

```
##   inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
## 1    3  306      2  74    1       1      90     100    1175      NA
## 2    3  455      2  68    1       0      90      90    1225      15
## 3    3 1010      1  56    1       0      90      90      NA      15
## 4    5  210      2  57    1       1      90      60    1150      11
## 5    1  883      2  60    1       0     100      90      NA      0
## 6   12 1022      1  74    1       1      50      80     513      0
```

```
tail(lung)
```

```
##   inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
## 223    1  116      2  76    1       1      80      80      NA      0
## 224    1  188      1  77    1       1      80      60      NA      3
## 225   13  191      1  39    1       0      90      90    2350     -5
## 226   32  105      1  75    2       2      60      70    1025      5
## 227    6  174      1  66    1       1      90     100    1075      1
## 228   22  177      1  58    2       1      80      90    1060      0
```

# A Kaplan-Meier Curve

First we fit the model:

```
## Hazard model
out_cph <- coxph(Surv(time, status) ~ age + sex, data = lung)

summary(out_cph)

## Call:
## coxph(formula = Surv(time, status) ~ age + sex, data = lung)
##
## n= 228, number of events= 165
##
##          coef exp(coef)    se(coef)      z Pr(>|z|)
## age   0.017045  1.017191  0.009223  1.848  0.06459 .
## sex  -0.513219  0.598566  0.167458 -3.065  0.00218 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##          exp(coef) exp(-coef) lower .95 upper .95
## age     1.0172     0.9831    0.9990    1.0357
## sex     0.5986     1.6707    0.4311    0.8311
##
## Concordance= 0.603  (se = 0.025 )
## Likelihood ratio test= 14.12  on 2 df,  p=9e-04
## Wald test            = 13.47  on 2 df,  p=0.001
## Score (logrank) test = 13.72  on 2 df,  p=0.001
```

# A Kaplan-Meier Curve

Then we create the survival curve, which is *nearly* tidy out of the box:

```
## Hazard model
out_surv ← survfit(out_cph)

## See how this is just a print method,
## not a tibble, or even a data frame.
## So it just runs off the end of the slide.
summary(out_surv)

## Call: survfit(formula = out_cph)
##
##   time n.risk n.event survival std.err lower 95% CI upper 95% CI
##      5    228      1 0.9958 0.00417    0.9877 1.000
##     11    227      3 0.9833 0.00831    0.9671 1.000
##     12    224      1 0.9791 0.00928    0.9611 0.997
##     13    223      2 0.9706 0.01096    0.9494 0.992
##     15    221      1 0.9664 0.01170    0.9438 0.990
##     26    220      1 0.9622 0.01240    0.9382 0.987
##     30    219      1 0.9579 0.01305    0.9327 0.984
##     31    218      1 0.9537 0.01368    0.9273 0.981
##     53    217      2 0.9452 0.01484    0.9165 0.975
##     54    215      1 0.9409 0.01538    0.9112 0.972
##     59    214      1 0.9366 0.01590    0.9060 0.968
##     60    213      2 0.9280 0.01689    0.8955 0.962
##     61    211      1 0.9237 0.01735    0.8903 0.958
##     62    210      1 0.9194 0.01780    0.8852 0.955
##     65    209      2 0.9109 0.01865    0.8751 0.948
##     71    207      1 0.9066 0.01906    0.8700 0.945
##     79    206      1 0.9023 0.01945    0.8650 0.941
```

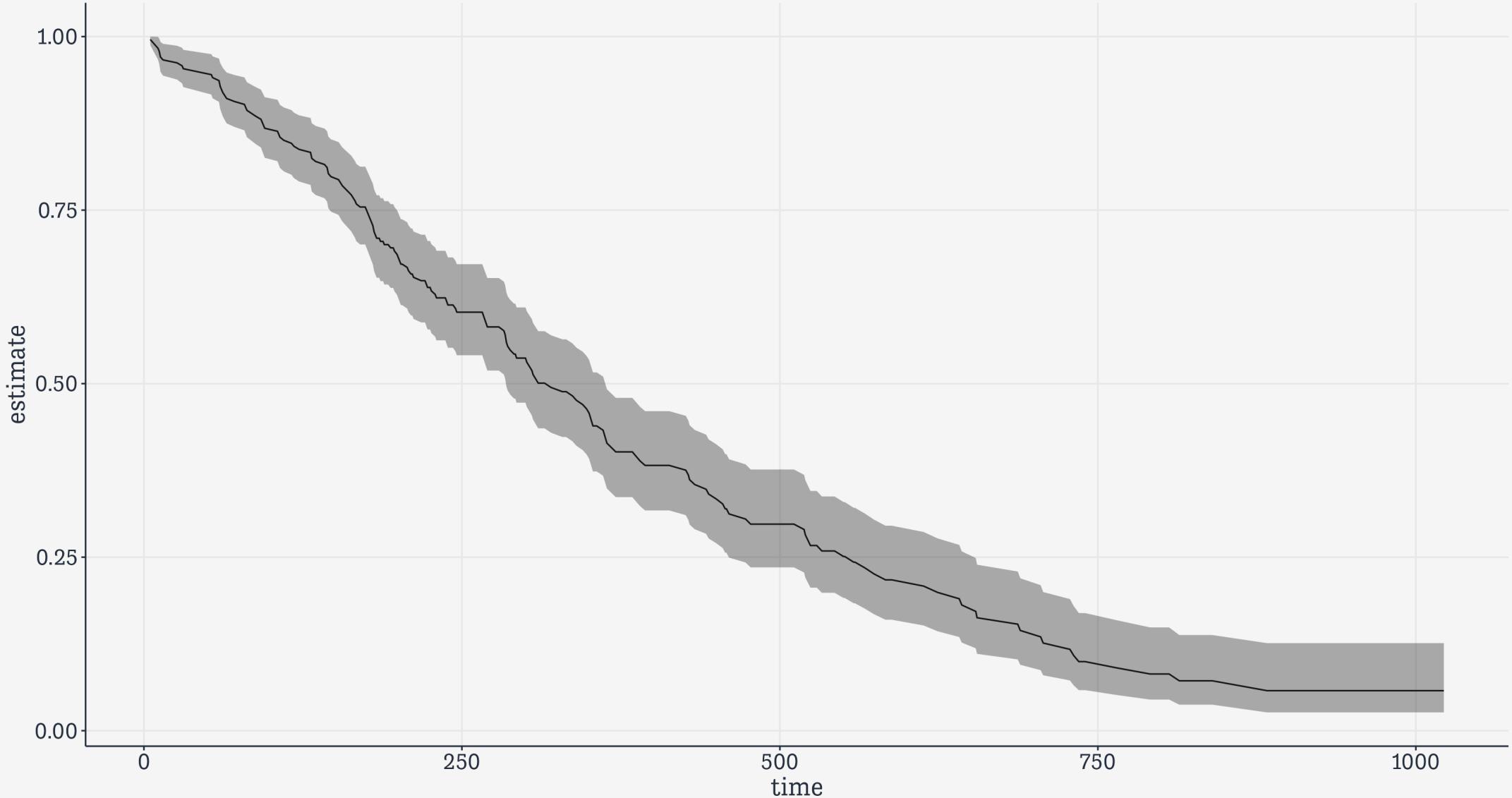
# A Kaplan-Meier Curve

Then we tidy it and draw the plot.

```
## Much nicer. (See how the column headers have been regularized, too.)
out_tidy <- tidy(out_surv)
out_tidy

## # A tibble: 186 × 8
##   time n.risk n.event n.censor estimate std.error conf.high conf.low
##   <dbl>   <dbl>   <dbl>   <dbl>     <dbl>     <dbl>       <dbl>     <dbl>
## 1     5     228      1      0     0.996    0.00419      1     0.988
## 2    11     227      3      0     0.983    0.00845     1.00    0.967
## 3    12     224      1      0     0.979    0.00947     0.997    0.961
## 4    13     223      2      0     0.971    0.0113      0.992    0.949
## 5    15     221      1      0     0.966    0.0121      0.990    0.944
## 6    26     220      1      0     0.962    0.0129      0.987    0.938
## 7    30     219      1      0     0.958    0.0136      0.984    0.933
## 8    31     218      1      0     0.954    0.0143      0.981    0.927
## 9    53     217      2      0     0.945    0.0157      0.975    0.917
## 10   54     215      1      0     0.941    0.0163      0.972    0.911
## # i 176 more rows

p_out <- out_tidy %>
  ggplot(mapping = aes(x = time, y = estimate)) +
  geom_line() +
  geom_ribbon(mapping = aes(ymin = conf.low, ymax = conf.high),
              alpha = 0.4)
```



Kaplan-Meier Plot

# Grouped Analysis with broom

Pipelines show  
their real power  
when used  
iteratively

# Iteration without tears (or explicit loops)

You might be familiar with code that looks like this:

```
x <- 10

for (i in 1:5) {
  print(x + i)
}

## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
```

This is one way to do something repeatedly.

# Iteration without tears (or explicit loops)

We can also write, e.g.,

```
x <- c(10, 20, 30, 40)

for (i in 1:length(x)) {
  # Add 5 to the ith element of x
  print(x[i] + 5)
}

## [1] 15
## [1] 25
## [1] 35
## [1] 45
```

This way we can refer to each element of x in turn, and do the same thing to it.

# Iteration without tears (or explicit loops)

The more complicated the thing we want to do, the more likely we are to use functions to help us out.

```
x <- 10  
  
for (i in 1:5) {  
  print(sqrt(x + i))  
}  
  
## [1] 3.316625  
## [1] 3.464102  
## [1] 3.605551  
## [1] 3.741657  
## [1] 3.872983
```

# Isn't this like ... Vectorized arithmetic?

The simplest cases are not that different from the vectorized arithmetic we saw before.

```
a <- c(1:10)  
b <- 1  
  
# You know what R will do here  
a + b  
  
## [1] 2 3 4 5 6 7 8 9 10 11
```

# Isn't this like ... Vectorized arithmetic?

The simplest cases are not that different from the vectorized arithmetic we saw before.

```
a <- c(1:10)  
b <- 1  
  
# You know what R will do here  
a + b  
  
## [1] 2 3 4 5 6 7 8 9 10 11
```

R's vectorized rules add b to every element of a. In a sense, the **+** operation can be thought of as a function that takes each element of a and does something with it. In this case "add b".

# Repeatedly applying a function

We can make this explicit by writing a function:

```
a <- c(1:10)

add_b <- function(x) {
  b <- 1
  x + b # for any x
}
```

Now:

```
add_b(x = a)

## [1]  2  3  4  5  6  7  8  9 10 11
```

In effect we take the vector **a** and feed it to the **add\_b()** function one element at a time.

# Repeatedly applying a function

Again, R's vectorized approach means it automatically applies `add_b()` to every element of the `x` we give it.

```
add_b(x = 10)
## [1] 11

add_b(x = c(1, 99, 1000))
## [1]    2 100 1001
```

# Iterating in a pipeline

Some operations can't directly be vectorized in this way, most often because the function we want to apply only knows what to do if it is handed, say, a vector. It doesn't understand what to do if it's handed a list of vectors or a tibble of them, etc. This is when we might find ourselves manually iterating---writing out every single step explicitly.

```
library(gapminder)
gapminder %>
  summarize(country_n = n_distinct(country),
           continent_n = n_distinct(continent),
           year_n = n_distinct(year),
           lifeExp_n = n_distinct(lifeExp),
           population_n = n_distinct(population))

## # A tibble: 1 × 5
##   country_n continent_n year_n lifeExp_n population_n
##       <int>      <int>    <int>      <int>
## 1        142          5     12       1626        4060
```

That's tedious to write! Computers are supposed to allow us to avoid that sort of thing.

# Iterating in a pipeline

So how would we iterate this? What we want is to apply the `n_distinct()` function to each column of `gapminder`. But we can't easily write a loop inside a pipeline. We want a way to iterate that lets us repeatedly apply a function without explicitly writing a loop.

```
library(gapminder)
gapminder %>
  summarize(n_distinct(country),
            n_distinct(continent),
            n_distinct(year),
            n_distinct(lifeExp),
            n_distinct(population))

## # A tibble: 1 × 5
##   `n_distinct(country)` `n_distinct(continent)` `n_distinct(year)` 
##                 <int>                  <int>          <int>
## 1                   142                      5              12
## # i 2 more variables: `n_distinct(lifeExp)` <int>,
## #   `n_distinct(population)` <int>
```

Using `n_distinct()` in this context is an idea I got from Rebecca Barter's discussion of `purrr`.

# Iterating in a pipeline

In real life, you'd use **across()**, like this:

```
gapminder %>  
  summarize(across(everything(), n_distinct))  
  
## # A tibble: 1 × 6  
##   country continent year lifeExp   pop gdpPercap  
##   <int>     <int> <int>   <int> <int>      <int>  
## 1     142         5    12     1626  1704       1704
```

# Iterating in a pipeline

But you could also say "Feed each column of `gapminder` in turn to the `n_distinct()` function".

This is what the `map()` function is for.

```
map(gapminder, n_distinct)

## $country
## [1] 142
##
## $continent
## [1] 5
##
## $year
## [1] 12
##
## $lifeExp
## [1] 1626
##
## $pop
## [1] 1704
##
## $gdpPerCap
## [1] 1704
```

Read it as "Feed each column of `gapminder` to the `n_distinct()` function.

(This is pretty much what `across()` is doing more nicely.)

# Iterating in a pipeline

Or, in pipeline form:

```
gapminder %>  
  map(n_distinct)
```

```
## $country  
## [1] 142  
##  
## $continent  
## [1] 5  
##  
## $year  
## [1] 12  
##  
## $lifeExp  
## [1] 1626  
##  
## $pop  
## [1] 1704  
##  
## $gdpPercap  
## [1] 1704
```

You can see we are getting a *list* back.

# Iterating in a pipeline

Or, in pipeline form:

```
result ← gapminder %>  
  map(n_distinct)  
  
class(result)  
## [1] "list"  
  
result$continent  
## [1] 5  
  
result[[2]]  
## [1] 5
```

# Iterating in a pipeline

But we know `n_distinct()` should always return an integer. So we use `map_int()` instead of the generic `map()`.

```
gapminder %>  
  map_int(n_distinct)  
  
## # A tibble: 142 x 5  
##   country continent year lifeExp pop gdpPercap  
##   <dbl>     <dbl> <dbl>    <dbl> <dbl>      <dbl>  
## 1       1        5     12     1626    1704      1704
```

The thing about the `map()` family is that it can deal with all kinds of input types and output types.

**So what's the use  
of all that stuff?**

# Grouped analysis and **list columns**

Let's say I want to fit a simple model to data for all countries in Europe in 1977.

```
eu77 <- gapminder %>
  filter(continent = "Europe", year = 1977)

fit <- lm(lifeExp ~ log(gdpPercap), data = eu77)

summary(fit)

##
## Call:
## lm(formula = lifeExp ~ log(gdpPercap), data = eu77)
##
## Residuals:
##     Min      1Q      Median      3Q      Max 
## -7.4956 -1.0306  0.0935  1.1755  3.7125 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 29.489     7.161   4.118 0.000306 ***  
## log(gdpPercap) 4.488     0.756   5.936 2.17e-06 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.114 on 28 degrees of freedom
## Multiple R-squared:  0.5572,    Adjusted R-squared:  0.5414 
## F-statistic: 35.24 on 1 and 28 DF,  p-value: 2.173e-06
```

# Grouped analysis and **list columns**

What if I want to do that for all Continent-Year combinations? I'm not going to write a loop!

```
out_le ← gapminder ▷  
  group_by(continent, year) ▷  
  nest()  
  
out_le  
  
## # A tibble: 60 × 3  
## # Groups: continent, year [60]  
##   continent year data  
##   <fct>     <int> <list>  
## 1 Asia      1952 <tibble [33 × 4]>  
## 2 Asia      1957 <tibble [33 × 4]>  
## 3 Asia      1962 <tibble [33 × 4]>  
## 4 Asia      1967 <tibble [33 × 4]>  
## 5 Asia      1972 <tibble [33 × 4]>  
## 6 Asia      1977 <tibble [33 × 4]>  
## 7 Asia      1982 <tibble [33 × 4]>  
## 8 Asia      1987 <tibble [33 × 4]>  
## 9 Asia      1992 <tibble [33 × 4]>  
## 10 Asia     1997 <tibble [33 × 4]>  
## # i 50 more rows
```

Think of nesting as a kind of "super-grouping". Look in the object inspector.

# Grouped analysis and **list columns**

Europe '77 is still in there.

```
out_le >
  filter(continent == "Europe" & year == 1977) >
  unnest(cols = c(data))

## # A tibble: 30 × 6
## # Groups: continent, year [1]
##   continent year country      lifeExp     pop gdpPercap
##   <fct>     <int> <fct>      <dbl>    <int>     <dbl>
## 1 Europe     1977 Albania     68.9  2509048     3533.
## 2 Europe     1977 Austria     72.2  7568430    19749.
## 3 Europe     1977 Belgium     72.8  9821800    19118.
## 4 Europe     1977 Bosnia and Herzegovina 69.9  4086000     3528.
## 5 Europe     1977 Bulgaria     70.8  8797022     7612.
## 6 Europe     1977 Croatia     70.6  4318673    11305.
## 7 Europe     1977 Czech Republic 70.7  10161915    14800.
## 8 Europe     1977 Denmark     74.7  5088419    20423.
## 9 Europe     1977 Finland     72.5  4738902    15605.
## 10 Europe    1977 France      73.8  53165019   18293.
## # i 20 more rows
```

# Grouped analysis and **list columns**

Here we write a tiny, very specific function and **map()** it to every row in the data column.

```
fit_ols <- function(df) {  
  lm(lifeExp ~ log(gdpPerCap), data = df)  
}  
  
out_le <- gapminder %>  
  group_by(continent, year) %>  
  nest() %>  
  mutate(model = map(data, fit_ols))
```

# Grouped analysis and **list** columns

Now we have a new column. Each row of the `model` column contains a full regression for that continent-year.

```
out_le

## # A tibble: 60 × 4
## # Groups: continent, year [60]
##   continent year data           model
##   <fct>     <int> <list>        <list>
## 1 Asia      1952 <tibble [33 × 4]> <lm>
## 2 Asia      1957 <tibble [33 × 4]> <lm>
## 3 Asia      1962 <tibble [33 × 4]> <lm>
## 4 Asia      1967 <tibble [33 × 4]> <lm>
## 5 Asia      1972 <tibble [33 × 4]> <lm>
## 6 Asia      1977 <tibble [33 × 4]> <lm>
## 7 Asia      1982 <tibble [33 × 4]> <lm>
## 8 Asia      1987 <tibble [33 × 4]> <lm>
## 9 Asia      1992 <tibble [33 × 4]> <lm>
## 10 Asia     1997 <tibble [33 × 4]> <lm>
## # i 50 more rows
```

# Grouped analysis and *list* columns

We can tidy the nested models, too.

```
fit_ols <- function(df) {  
  lm(lifeExp ~ log(gdpPerCap), data = df)  
}  
  
out_tidy <- gapminder %>  
  group_by(continent, year) %>  
  nest() %>  
  mutate(model = map(data, fit_ols),  
        tidied = map(model, tidy))  
  
out_tidy  
  
## # A tibble: 60 × 5  
## # Groups: continent, year [60]  
##   continent year data           model  tidied  
##   <fct>     <int> <list>         <list> <list>  
## 1 Asia       1952 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 2 Asia       1957 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 3 Asia       1962 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 4 Asia       1967 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 5 Asia       1972 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 6 Asia       1977 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 7 Asia       1982 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 8 Asia       1987 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 9 Asia       1992 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## 10 Asia      1997 <tibble [33 × 4]> <lm>    <tibble [2 × 5]>  
## # i 50 more rows
```

# Grouped analysis and **list** columns

We can get the tidied results out into the main table if we like.

```
out_tidy <- out_tidy %>
  unnest(cols = c(tidied)) %>
  filter(term %nin% "(Intercept)" &
        continent %nin% "Oceania")

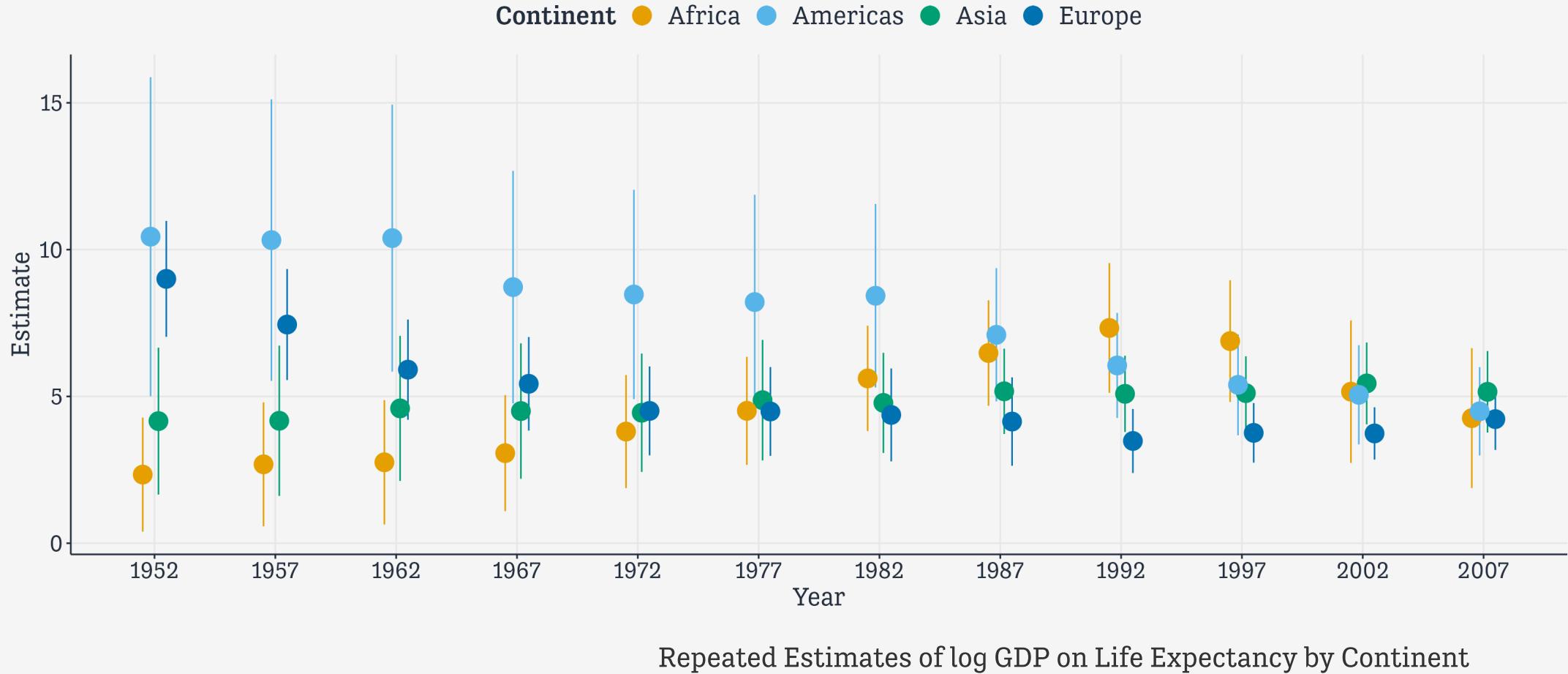
out_tidy

## # A tibble: 48 × 9
## # Groups:   continent, year [48]
##   continent year data    model term      estimate std.error statistic p.value
##   <fct>     <int> <list> <list> <chr>       <dbl>     <dbl>      <dbl>    <dbl>
## 1 Asia       1952 <tibble> <lm> log(gdp...  4.16      1.25      3.33  2.28e-3
## 2 Asia       1957 <tibble> <lm> log(gdp...  4.17      1.28      3.26  2.71e-3
## 3 Asia       1962 <tibble> <lm> log(gdp...  4.59      1.24      3.72  7.94e-4
## 4 Asia       1967 <tibble> <lm> log(gdp...  4.50      1.15      3.90  4.77e-4
## 5 Asia       1972 <tibble> <lm> log(gdp...  4.44      1.01      4.41  1.16e-4
## 6 Asia       1977 <tibble> <lm> log(gdp...  4.87      1.03      4.75  4.42e-5
## 7 Asia       1982 <tibble> <lm> log(gdp...  4.78      0.852     5.61  3.77e-6
## 8 Asia       1987 <tibble> <lm> log(gdp...  5.17      0.727     7.12  5.31e-8
## 9 Asia       1992 <tibble> <lm> log(gdp...  5.09      0.649     7.84  7.60e-9
## 10 Asia      1997 <tibble> <lm> log(gdp...  5.11      0.628     8.15  3.35e-9
## # i 38 more rows
```

# Plot what we have

```
p <- ggplot(data = out_tidy,
             mapping = aes(x = year, y = estimate,
                           ymin = estimate - 2*std.error,
                           ymax = estimate + 2*std.error,
                           group = continent,
                           color = continent))

p_out <- p +
  geom_pointrange(size = rel(1.25),
                  position = position_dodge(width = rel(1.3))) +
  scale_x_continuous(breaks = unique(gapminder$year)) +
  labs(x = "Year",
       y = "Estimate",
       color = "Continent")
```



# And there's more ...

Let's go back to this stage:

```
# New model
fit_ols2 <- function(df) {
  lm(lifeExp ~ log(gdpPercap) + log(pop), data = df)
}

out_tidy <- gapminder %>
  group_by(continent, year) %>
  nest() %>
  mutate(model = map(data, fit_ols2),
         tidied = map(model, tidy))

out_tidy

## # A tibble: 60 × 5
## # Groups:   continent, year [60]
##   continent    year   data           model   tidied
##   <fct>      <int> <list>          <list> <list>
## 1 Asia        1952 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 2 Asia        1957 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 3 Asia        1962 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 4 Asia        1967 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 5 Asia        1972 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 6 Asia        1977 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 7 Asia        1982 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 8 Asia        1987 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 9 Asia        1992 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## 10 Asia       1997 <tibble [33 × 4]> <lm>   <tibble [3 × 5]>
## # i 50 more rows
```

# A function to draw a coef plot

```
# Plot the output from our model
mod_plot <- function(data,
                      title){
  data %>
    filter(term %nin% "(Intercept)") %>
    ggplot(mapping = aes(x = estimate,
                          xmin = estimate - std.error,
                          xmax = estimate + std.error,
                          y = reorder(term, estimate))) +
    geom_pointrange() +
    labs(title = title,
         y = NULL)
}
```

# Add it using `map2()` or `pmap()`

When we have two arguments to feed a function we can use `map2()`. The general case is `pmap()`, for passing along any number of arguments in a list.

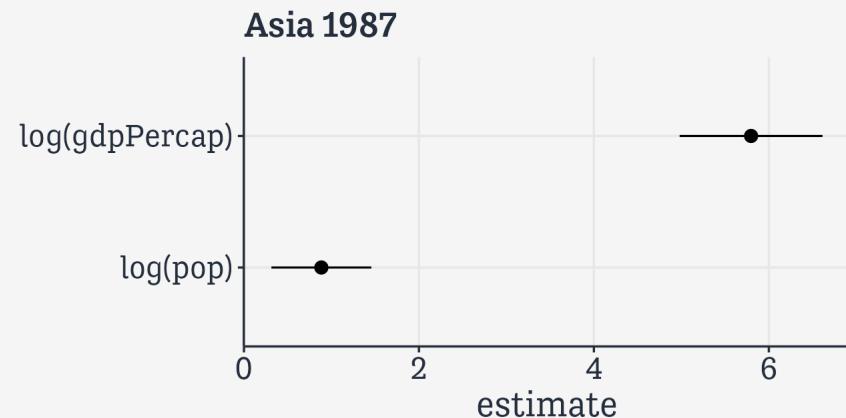
```
out_tidy ← gapminder ▷  
  group_by(continent, year) ▷  
  nest() ▷  
  mutate(title = paste(continent, year),  
        model = map(data, fit_ols2),  
        tidied = map(model, tidy),  
        ggout = pmap(list(tidied,  
                           title),  
                    mod_plot))
```

```
out_tidy
```

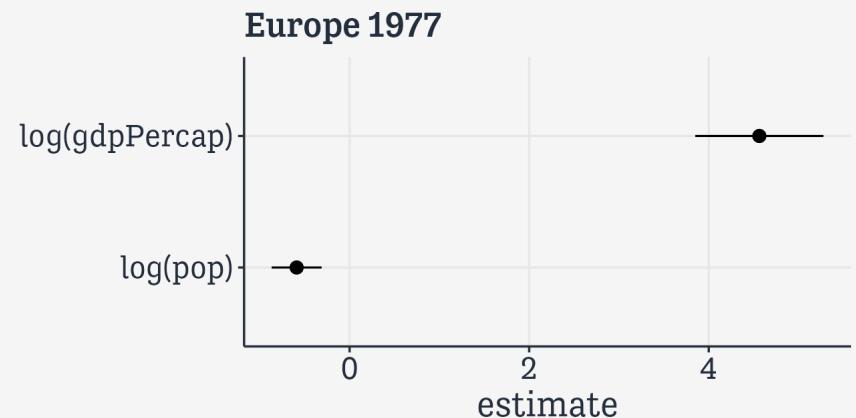
```
## # A tibble: 60 × 7  
## # Groups: continent, year [60]  
##   continent year data           title    model    tidied      ggout  
##   <fct>     <int> <list>          <chr>    <list> <list>      <list>  
## 1 Asia       1952 <tibble [33 × 4]> Asia 1952 <lm>    <tibble [3 × 5]> <gg>  
## 2 Asia       1957 <tibble [33 × 4]> Asia 1957 <lm>    <tibble [3 × 5]> <gg>  
## 3 Asia       1962 <tibble [33 × 4]> Asia 1962 <lm>    <tibble [3 × 5]> <gg>  
## 4 Asia       1967 <tibble [33 × 4]> Asia 1967 <lm>    <tibble [3 × 5]> <gg>  
## 5 Asia       1972 <tibble [33 × 4]> Asia 1972 <lm>    <tibble [3 × 5]> <gg>  
## 6 Asia       1977 <tibble [33 × 4]> Asia 1977 <lm>    <tibble [3 × 5]> <gg>  
## 7 Asia       1982 <tibble [33 × 4]> Asia 1982 <lm>    <tibble [3 × 5]> <gg>  
## 8 Asia       1987 <tibble [33 × 4]> Asia 1987 <lm>    <tibble [3 × 5]> <gg>  
## 9 Asia       1992 <tibble [33 × 4]> Asia 1992 <lm>    <tibble [3 × 5]> <gg>
```

# A plot!

out\_tidy\$ggout[[8]]



out\_tidy\$ggout[[18]]



# We don't just put them in there for fun

We can e.g. `walk` the plots out to disk

`walk()` is `map()` for when you just want a "side-effect" such as printed output. There is also `walk2()` and `pwalk()`

```
pwalk(  
  list(  
    filename = paste0(out_tidy$title, ".png"),  
    plot = out_tidy$ggout,  
    path = here("figures"),  
    height = 3, width = 4,  
    dpi = 300  
)  
,  
ggsave  
)
```

# Peek in the figures/ folder

```
fs::dir_ls(here("figures")) >  
basename()  
  
## [1] "Africa 1952.png"   "Africa 1957.png"   "Africa 1962.png"  
## [4] "Africa 1967.png"   "Africa 1972.png"   "Africa 1977.png"  
## [7] "Africa 1982.png"   "Africa 1987.png"   "Africa 1992.png"  
## [10] "Africa 1997.png"   "Africa 2002.png"   "Africa 2007.png"  
## [13] "Americas 1952.png" "Americas 1957.png" "Americas 1962.png"  
## [16] "Americas 1967.png" "Americas 1972.png" "Americas 1977.png"  
## [19] "Americas 1982.png" "Americas 1987.png" "Americas 1992.png"  
## [22] "Americas 1997.png" "Americas 2002.png" "Americas 2007.png"  
## [25] "Asia 1952.png"     "Asia 1957.png"     "Asia 1962.png"  
## [28] "Asia 1967.png"     "Asia 1972.png"     "Asia 1977.png"  
## [31] "Asia 1982.png"     "Asia 1987.png"     "Asia 1992.png"  
## [34] "Asia 1997.png"     "Asia 2002.png"     "Asia 2007.png"  
## [37] "Europe 1952.png"    "Europe 1957.png"    "Europe 1962.png"  
## [40] "Europe 1967.png"    "Europe 1972.png"    "Europe 1977.png"  
## [43] "Europe 1982.png"    "Europe 1987.png"    "Europe 1992.png"  
## [46] "Europe 1997.png"    "Europe 2002.png"    "Europe 2007.png"  
## [49] "Oceania 1952.png"   "Oceania 1957.png"   "Oceania 1962.png"  
## [52] "Oceania 1967.png"   "Oceania 1972.png"   "Oceania 1977.png"  
## [55] "Oceania 1982.png"   "Oceania 1987.png"   "Oceania 1992.png"  
## [58] "Oceania 1997.png"   "Oceania 2002.png"   "Oceania 2007.png"
```

**Get model-based  
graphics right**

# Present findings in substantive terms

**Present findings in substantive terms**

**Show degrees of confidence or uncertainty**

**Present findings in substantive terms**

**Show degrees of confidence or uncertainty**

**Show the data when you can**

These points apply just  
as well to presenting  
data in *any* format:  
tables, models, text,  
whatever. Graphs are  
not special here!

# Plot Marginal Effects

# The marginaleffects package

# An example from the GSS

gss\_sm

```
## # A tibble: 2,867 × 32
##   year   id ballot      age child� sibs degree race   sex   region income16
##   <dbl> <dbl> <labelled> <dbl> <dbl> <labe> <fct> <fct> <fct> <fct> <fct>
## 1 2016    1 1           47     3 2  Bach... White Male New E... $170000...
## 2 2016    2 2           61     0 3  High ... White Male New E... $50000 ...
## 3 2016    3 3           72     2 3  Bach... White Male New E... $75000 ...
## 4 2016    4 1           43     4 3  High ... White Fema... New E... $170000...
## 5 2016    5 3           55     2 2  Gradu... White Fema... New E... $170000...
## 6 2016    6 2           53     2 2  Junio... White Fema... New E... $60000 ...
## 7 2016    7 1           50     2 2  High ... White Male New E... $170000...
## 8 2016    8 3           23     3 6  High ... Other Fema... Middl... $30000 ...
## 9 2016    9 1           45     3 5  High ... Black Male Middl... $60000 ...
## 10 2016   10 3          71     4 1  Junio... White Male Middl... $60000 ...
## # i 2,857 more rows
## # i 21 more variables: relig <fct>, marital <fct>, padeg <fct>, madeg <fct>,
## # partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
## # zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
## # agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
## # bigregion <fct>, partners_rc <fct>, obama <dbl>
```

# Set up our model

```
gss_sm$polviews_m ← relevel(gss_sm$polviews,
                             ref = "Moderate")

out_bo ← glm(obama ~ polviews_m + sex*race,
             family = "binomial",
             data = gss_sm)

tidy(out_bo)

## # A tibble: 12 × 5
##   term                  estimate std.error statistic p.value
##   <chr>                 <dbl>     <dbl>      <dbl>    <dbl>
## 1 (Intercept)            0.296     0.134      2.21    2.70e- 2
## 2 polviews_mExtremely Liberal  2.37      0.525      4.52    6.20e- 6
## 3 polviews_mLiberal      2.60      0.357      7.29    3.10e-13
## 4 polviews_msSlightly Liberal  1.29      0.248      5.21    1.94e- 7
## 5 polviews_msSlightly Conservative -1.36     0.181     -7.48    7.68e-14
## 6 polviews_mConservative     -2.35     0.200     -11.7    1.07e-31
## 7 polviews_mEExtremely Conservative -2.73     0.387     -7.04    1.87e-12
## 8 sexFemale                0.255     0.145      1.75    7.96e- 2
## 9 raceBlack                  3.85      0.501      7.68    1.61e-14
## 10 raceOther                 -0.00214   0.436     -0.00492  9.96e- 1
## 11 sexFemale:raceBlack       -0.198     0.660     -0.299    7.65e- 1
## 12 sexFemale:raceOther        1.57      0.588      2.68    7.37e- 3
```

# Calculate the Marginal Effects

```
library(marginaleffects)

bo_mfx ← marginaleffects(out_bo)

## This gives us the marginal effects at the unit level
as_tibble(bo_mfx)

## # A tibble: 15,282 × 16
##   rowid term  contrast estimate std.error statistic p.value conf.low conf.high
##   <int> <chr> <chr>     <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1     1 polv... Conserv... -0.460    0.0329    -14.0  1.93e-44   -0.524   -0.395
## 2     1 polv... Extreme... -0.493    0.0401    -12.3  1.07e-34   -0.571   -0.414
## 3     1 polv... Extreme...  0.362    0.0433     8.35  6.77e-17    0.277   0.446
## 4     1 polv... Liberal...  0.374    0.0350    10.7   1.19e-26    0.305   0.443
## 5     1 polv... Slightl... -0.316    0.0384    -8.23  1.89e-16   -0.391   -0.241
## 6     1 polv... Slightl...  0.257    0.0417     6.17  6.83e-10    0.175   0.339
## 7     2 polv... Conserv... -0.460    0.0329    -14.0  1.93e-44   -0.524   -0.395
## 8     2 polv... Extreme... -0.493    0.0401    -12.3  1.07e-34   -0.571   -0.414
## 9     2 polv... Extreme...  0.362    0.0433     8.35  6.77e-17    0.277   0.446
## 10    2 polv... Liberal...  0.374    0.0350    10.7   1.19e-26    0.305   0.443
## # i 15,272 more rows
## # i 7 more variables: predicted <dbl>, predicted_hi <dbl>, predicted_lo <dbl>,
## #   obama <dbl>, polviews_m <fct>, sex <fct>, race <fct>
```

# Calculate the AMEs

And this gives us the Average Marginal Effects

```
summary(bo_mfx) ▷  
as_tibble()  
  
## # A tibble: 9 × 8  
##   term      contrast estimate std.error statistic p.value conf.low conf.high  
##   <chr>     <chr>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>  
## 1 polviews_m mean(Cons...) -0.351    0.0306    -11.5  1.44e-30  -0.411    -0.291  
## 2 polviews_m mean(Extr...) -0.401    0.0510    -7.88  3.41e-15  -0.501    -0.301  
## 3 polviews_m mean(Extr...)  0.194    0.0248     7.80  6.08e-15   0.145     0.242  
## 4 polviews_m mean(Libe...)  0.200    0.0215     9.30  1.46e-20   0.158     0.242  
## 5 polviews_m mean(Slig...) -0.208    0.0284    -7.32  2.45e-13  -0.264    -0.153  
## 6 polviews_m mean(Slig...)  0.140    0.0234      5.98  2.23e- 9   0.0941    0.186  
## 7 race       mean(Blac...)  0.392    0.0187     20.9  4.61e-97   0.355     0.429  
## 8 race       mean(Othe...)  0.0973   0.0357     2.73  6.40e- 3   0.0273    0.167  
## 9 sex        mean(Fema...)  0.0869   0.0262     3.32  8.95e- 4   0.0356    0.138
```

# Alternatively, do it with broom

```
tidy(bo_mfx)

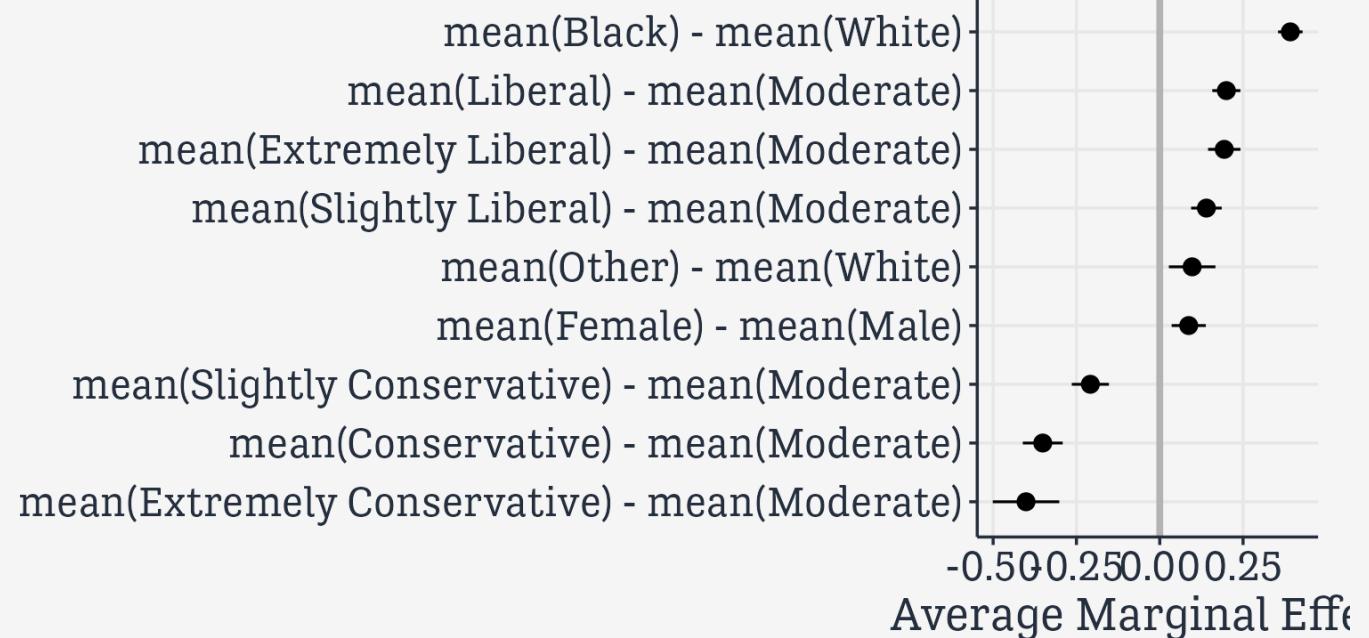
## # A tibble: 9 × 8
##   term      contrast estimate std.error statistic p.value conf.low conf.high
##   <chr>     <chr>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
## 1 polviews_m mean(Cons... -0.351    0.0306    -11.5  1.44e-30  -0.411   -0.291
## 2 polviews_m mean(Extr... -0.401    0.0510    -7.88  3.41e-15  -0.501   -0.301
## 3 polviews_m mean(Extr...  0.194    0.0248     7.80  6.08e-15   0.145    0.242
## 4 polviews_m mean(Libe...  0.200    0.0215     9.30  1.46e-20   0.158    0.242
## 5 polviews_m mean(Slig... -0.208    0.0284    -7.32  2.45e-13  -0.264   -0.153
## 6 polviews_m mean(Slig...  0.140    0.0234     5.98  2.23e- 9   0.0941   0.186
## 7 race       mean(Blac...  0.392    0.0187    20.9   4.61e-97   0.355    0.429
## 8 race       mean(Othe...  0.0973   0.0357     2.73  6.40e- 3   0.0273   0.167
## 9 sex        mean(Fema...  0.0869   0.0262     3.32  8.95e- 4   0.0356   0.138
```

# Which gets us back to familiar territory

```
tidy(bo_mfx) %>%  
  ggplot(mapping = aes(x = estimate,  
                        xmin = conf.low,  
                        xmax = conf.high,  
                        y = reorder(contrast,  
                                    estimate))) +  
  geom_vline(xintercept = 0, color = "gray70",  
             size = rel(1.2)) +  
  geom_pointrange() +  
  labs(x = "Average Marginal Effect",  
       y = NULL)
```

# Which gets us back to familiar territory

```
tidy(bo_mfx) %>%  
  ggplot(mapping = aes(x = estimate,  
                        xmin = conf.low,  
                        xmax = conf.high,  
                        y = reorder(contrast,  
                                    estimate))) +  
  geom_vline(xintercept = 0, color = "gray70",  
             size = rel(1.2)) +  
  geom_pointrange() +  
  labs(x = "Average Marginal Effect",  
       y = NULL)
```



# marginaleffects can do a lot more

marginaleffects 0.4.1.9000    [Adjusted predictions](#)    [Marginal effects](#)    [Contrasts](#)    [Marginal means](#)    [Functions](#)    [Changelog](#)

## The marginaleffects package for R



marginaleffects is an R package to compute and plot adjusted predictions, marginal effects, contrasts, and marginal means for a *wide* variety of models.

### Table of contents

- [What?](#)
- [Why?](#)
- [Getting started](#)
- Vignettes:
  - [Adjusted predictions](#)
  - [Marginal effects](#)
  - [Contrasts](#)
  - [Marginal means](#)
- [61 supported models](#)
- [Performance tips](#)
- Case studies:
  - [Bayesian analyses with brms](#)
  - [Mixed effects models](#)
  - [Generalized Additive Models](#)
  - [Multinomial Logit and Discrete Choice Models](#)
  - [Tables and plots](#)
  - [Robust standard errors and more](#)
  - [Transformations and Custom Contrasts: Adjusted Risk Ratio Example](#)
- [Alternative software](#)
- [Technical notes](#)

### Links

[View on CRAN](#)  
[Browse source code](#)  
[Report a bug](#)

### License

[Full license](#)  
GPL (>= 3)

### Citation

[Citing marginaleffects](#)

### Developers

Vincent Arel-Bundock  
Author, maintainer, copyright holder 

### Dev status

 codecov 90%  
 R-CMD-check passing  
CRAN 0.4.1  
dependencies 3/4

# `marginalEffects` can do a lot more

It includes a range of plotting methods, to produce graphics directly.

These are built on `ggplot`. Similarly integration with `broom` means that you can use the package-specific plotting functions take the tidy output and adapt it to your own needs.

Also check out `modelsummary`, by the same author, for quick and flexible summaries of models and datasets. Again, this sort of package is very convenient to use directly. But with just a little facility with R and tidyverse-style idioms and patterns, you'll get even more out of it. You'll better understand how to adapt it and why its functions work as they do.

# Complex Surveys

# Using the `survey` and `srvyr` packages

# Working with complex surveys

As always, our question is "What's the smoothest way for me to get the **tidy table of results** I need to hand off to **ggplot**?"

For complex surveys, we use **survey**, the standard package for survey analysis in R, and **srvyr**, a helper package designed to integrate what **survey** can do with the Tidyverse framework.

```
## Load the packages
library(survey)
library(srvyr)
```

# Example: The GSS again

This time, a small piece of the full GSS from the early 1970s to 2018.

gss\_lon

```
## # A tibble: 62,466 × 25
##   year   id ballot age degree race sex   siblings kids bigregion income16
##   <dbl> <dbl> <labe> <lab> <fct> <fct> <fct> <fct> <fct> <fct> <fct>
## 1 1972     1 NA    23 Bache... White Fema... 3      0   Midwest <NA>
## 2 1972     2 NA    70 Lt Hi... White Male  4     4+   Midwest <NA>
## 3 1972     3 NA    48 High ... White Fema... 5     4+   Midwest <NA>
## 4 1972     4 NA    27 Bache... White Fema... 5      0   Midwest <NA>
## 5 1972     5 NA    61 High ... White Fema... 2      2   Midwest <NA>
## 6 1972     6 NA    26 High ... White Male  1      0   Midwest <NA>
## 7 1972     7 NA    28 High ... White Male  6+     2   Midwest <NA>
## 8 1972     8 NA    27 Bache... White Male  1      0   Midwest <NA>
## 9 1972     9 NA    21 High ... Black Fema... 2      2   South  <NA>
## 10 1972    10 NA   30 High ... Black Fema... 6+    4+   South  <NA>
## # i 62,456 more rows
## # i 14 more variables: religion <fct>, marital <fct>, padeg <fct>, madeg <fct>,
## # partyid <fct>, polviews <fct>, happy <fct>, partners_rc <fct>, grass <fct>,
## # zodiac <fct>, pres12 <labelled>, wtssall <dbl>, vpsu <dbl>, vstrat <dbl>
```

# Add the weighting information

```
# These details are dependent on the kind of survey you're working with
options(survey.lonely.psu = "adjust")
options(na.action="na.pass")

gss_svy ← gss_lon ▷
  filter(year > 1974) ▷
  mutate(stratvar = interaction(year, vstrat)) ▷
  as_survey_design(ids = vpsu,
                  strata = stratvar,
                  weights = wtssall,
                  nest = TRUE)

gss_svy # Now it's no longer simply a tibble

## Stratified 1 - level Cluster Sampling design (with replacement)
## With (4399) clusters.
## Called via srvyr
## Sampling variables:
##   - ids: vpsu
##   - strata: stratvar
##   - weights: wtssall
## Data variables: year (dbl), id (dbl), ballot (labelled), age (labelled), degree
##   (fct), race (fct), sex (fct), siblings (fct), kids (fct), bigregion (fct),
##   income16 (fct), religion (fct), marital (fct), padeg (fct), madeg (fct),
##   partyid (fct), polviews (fct), happy (fct), partners_rc (fct), grass (fct),
##   zodiac (fct), pres12 (labelled), wtssall (dbl), vpsu (dbl), vstrat (dbl),
##   stratvar (fct)
```

# Trends in the happy measure

```
out_hap ← gss_svy ▷  
  group_by(year, sex, happy) ▷  
  summarize(prop = survey_mean(na.rm = TRUE, vartype = "ci"))  
  
out_hap  
  
## # A tibble: 221 × 6  
## # Groups:   year, sex [56]  
##   year   sex   happy       prop   prop_low  prop_upp  
##   <dbl> <fct> <fct>     <dbl>     <dbl>      <dbl>  
## 1 1975 Male  Very Happy  0.319    0.279     0.358  
## 2 1975 Male  Pretty Happy 0.555    0.514     0.597  
## 3 1975 Male  Not Too Happy 0.119    0.0934    0.145  
## 4 1975 Male  <NA>        0.00670 -0.0000539 0.0134  
## 5 1975 Female Very Happy  0.345    0.310     0.380  
## 6 1975 Female Pretty Happy 0.516    0.472     0.560  
## 7 1975 Female Not Too Happy 0.138    0.108     0.167  
## 8 1975 Female <NA>        0.00117 -0.00113  0.00347  
## 9 1976 Male  Very Happy  0.328    0.289     0.367  
## 10 1976 Male  Pretty Happy 0.543    0.505     0.580  
## # i 211 more rows
```

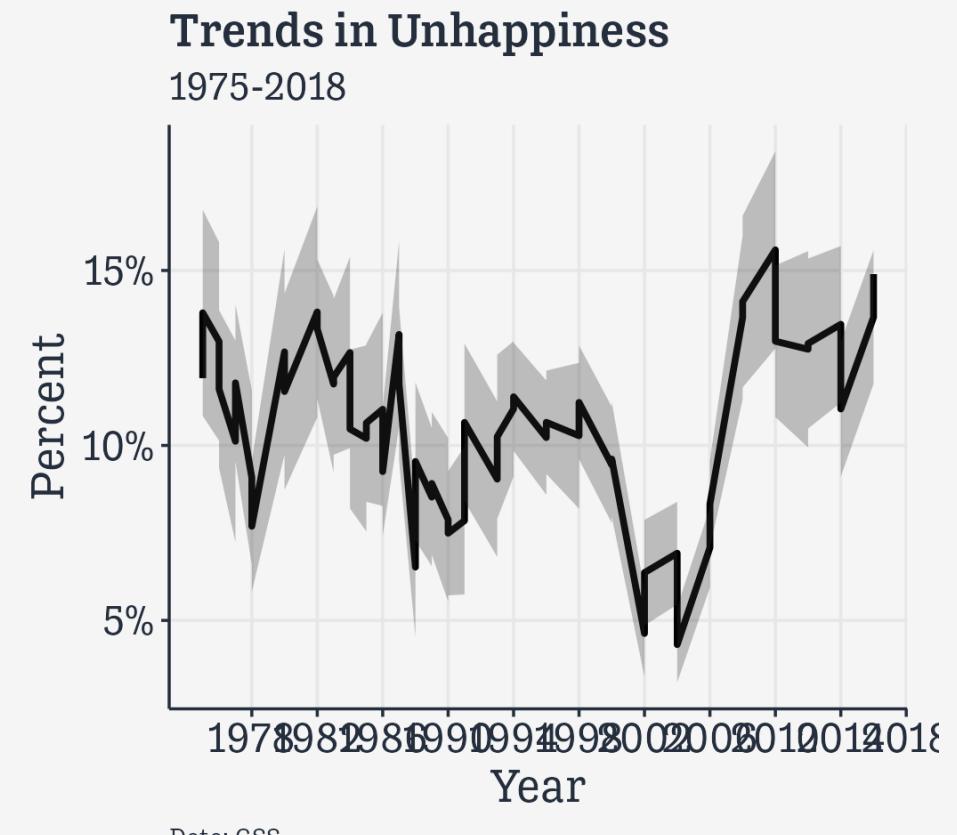
Once again, it's now a tidy tibble, and we know what to do with those.

```
out_hap %>
  filter(happy == "Not Too Happy") %>
  ggplot(mapping = aes(x = year,
                       y = prop,
                       ymin = prop_low,
                       ymax = prop_upp)) +
  geom_line(linewidth = 1.2) +
  geom_ribbon(alpha = 0.3) +
  scale_x_continuous(breaks =
    seq(1978, 2018, 4)) +
  scale_y_continuous(labels =
    label_percent(accuracy = 1)) +
  labs(x = "Year",
       y = "Percent",
       title = "Trends in Unhappiness",
       subtitle = "1975-2018",
       caption = "Data: GSS.")
```

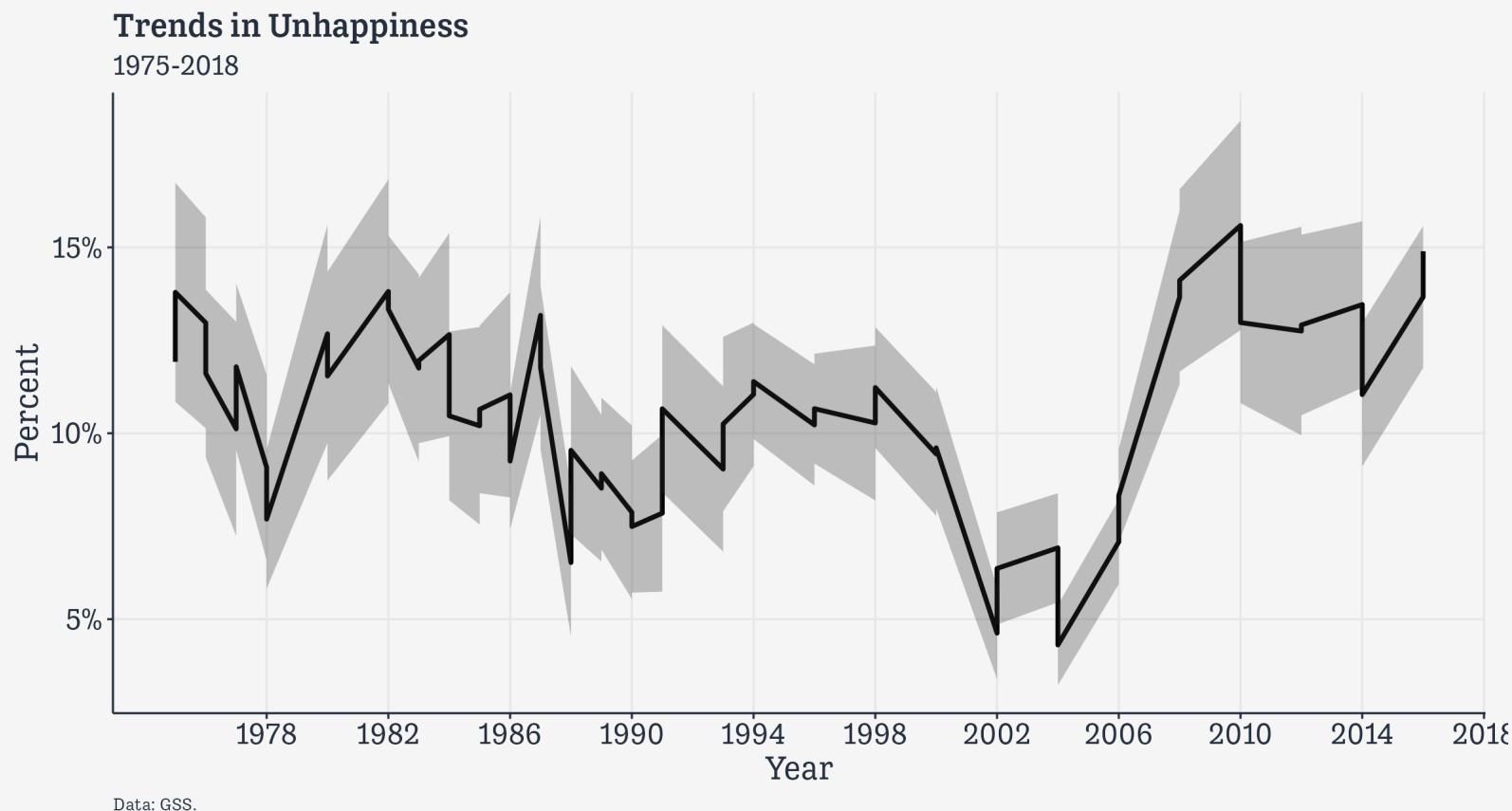
```

out_hap %>
  filter(happy == "Not Too Happy") %>
  ggplot(mapping = aes(x = year,
                        y = prop,
                        ymin = prop_low,
                        ymax = prop_upp)) +
  geom_line(linewidth = 1.2) +
  geom_ribbon(alpha = 0.3) +
  scale_x_continuous(breaks =
    seq(1978, 2018, 4)) +
  scale_y_continuous(labels =
    label_percent(accuracy = 1)) +
  labs(x = "Year",
       y = "Percent",
       title = "Trends in Unhappiness",
       subtitle = "1975-2018",
       caption = "Data: GSS.")

```



# With a proper aspect ratio



# A more complex example

```
gss_svy ▷  
  filter(year %in% seq(1976, 2016, by = 4)) ▷  
  group_by(year, race, degree) ▷  
  summarize(prop = survey_mean(na.rm = TRUE))  
  
## # A tibble: 162 × 5  
## # Groups:   year, race [30]  
##   year race  degree      prop  prop_se  
##   <dbl> <fct> <fct>      <dbl>  <dbl>  
## 1 1976 White Lt High School 0.327  0.0160  
## 2 1976 White High School   0.517  0.0161  
## 3 1976 White Junior College 0.0128  0.00298  
## 4 1976 White Bachelor     0.101   0.00955  
## 5 1976 White Graduate     0.0392  0.00642  
## 6 1976 White <NA>         0.00285 0.00151  
## 7 1976 Black Lt High School 0.558  0.0603  
## 8 1976 Black High School   0.335  0.0476  
## 9 1976 Black Junior College 0.0423  0.0192  
## 10 1976 Black Bachelor    0.0577  0.0238  
## # i 152 more rows
```

# Let's put that in an object

```
out_yrd ← gss_svy ▷  
  filter(year %in% seq(1976, 2016, by = 4)) ▷  
  group_by(year, race, degree) ▷  
  summarize(prop = survey_mean(na.rm = TRUE))
```

# Check the sums

```
out_yrd %>  
  group_by(year, race) %>  
  summarize(tot = sum(prop))  
  
## # A tibble: 30 × 3  
## # Groups:   year [10]  
##       year race   tot  
##   <dbl> <fct> <dbl>  
## 1 1976 White  1.00  
## 2 1976 Black  1.00  
## 3 1976 Other  1  
## 4 1980 White  1.00  
## 5 1980 Black  1  
## 6 1980 Other  1  
## 7 1984 White  1.00  
## 8 1984 Black  1.00  
## 9 1984 Other  1  
## 10 1988 White  1.00  
## # ... with 20 more rows
```

# Set up the plot

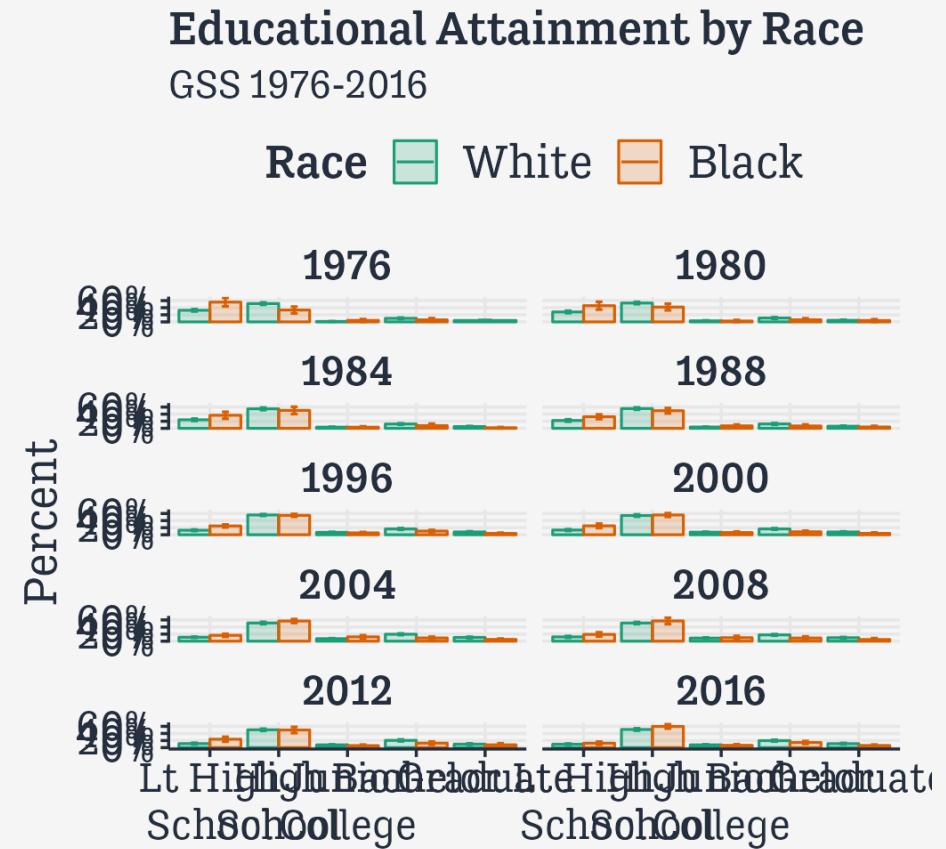
```
p <- out_yrd %>  
  drop_na() %>  
  filter(race %nin% "Other") %>  
  ggplot(mapping = aes(x = degree,  
                        y = prop,  
                        ymin = prop - 2*prop_se,  
                        ymax = prop + 2*prop_se,  
                        fill = race,  
                        color = race,  
                        group = race))  
  
dodge_w <- position_dodge(width = 0.9)
```

# Draw the plot

```
p + geom_col(position = dodge_w, alpha = 0.2) +
  geom_errorbar(position = dodge_w, width = 0.2) +
  scale_x_discrete(labels = wrap_format(10)) +
  scale_y_continuous(labels = label_percent()) +
  scale_color_brewer(type = "qual",
                      palette = "Dark2") +
  scale_fill_brewer(type = "qual",
                     palette = "Dark2") +
  labs(title = "Educational Attainment by Race",
       subtitle = "GSS 1976-2016",
       fill = "Race",
       color = "Race",
       x = NULL, y = "Percent") +
  facet_wrap(~ year, ncol = 2)
```

# Draw the plot

```
p + geom_col(position = dodge_w, alpha = 0.2) +  
  geom_errorbar(position = dodge_w, width = 0.2) +  
  scale_x_discrete(labels = wrap_format(10)) +  
  scale_y_continuous(labels = label_percent()) +  
  scale_color_brewer(type = "qual",  
                     palette = "Dark2") +  
  scale_fill_brewer(type = "qual",  
                     palette = "Dark2") +  
  labs(title = "Educational Attainment by Race",  
       subtitle = "GSS 1976-2016",  
       fill = "Race",  
       color = "Race",  
       x = NULL, y = "Percent") +  
  facet_wrap(~ year, ncol = 2)
```



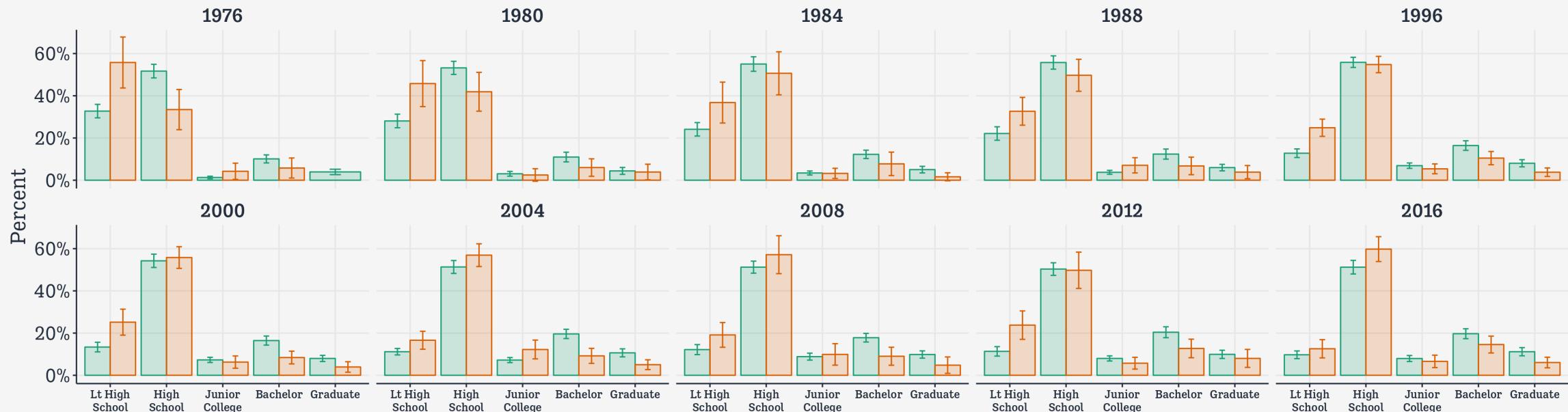
# In full (but switch to rows)

```
p_out ← p +
  geom_col(position = dodge_w, alpha = 0.2) +
  geom_errorbar(position = dodge_w, width = 0.2) +
  scale_x_discrete(labels = wrap_format(10)) +
  scale_y_continuous(labels = label_percent()) +
  scale_color_brewer(type = "qual",
                      palette = "Dark2") +
  scale_fill_brewer(type = "qual",
                     palette = "Dark2") +
  labs(title = "Educational Attainment by Race",
       subtitle = "GSS 1976-2016",
       fill = "Race",
       color = "Race",
       x = NULL, y = "Percent") +
  facet_wrap(~ year, nrow = 2) +
  theme(axis.text.x =
        element_text(size = rel(0.6),
                     face = "bold"))
```

## Educational Attainment by Race

GSS 1976-2016

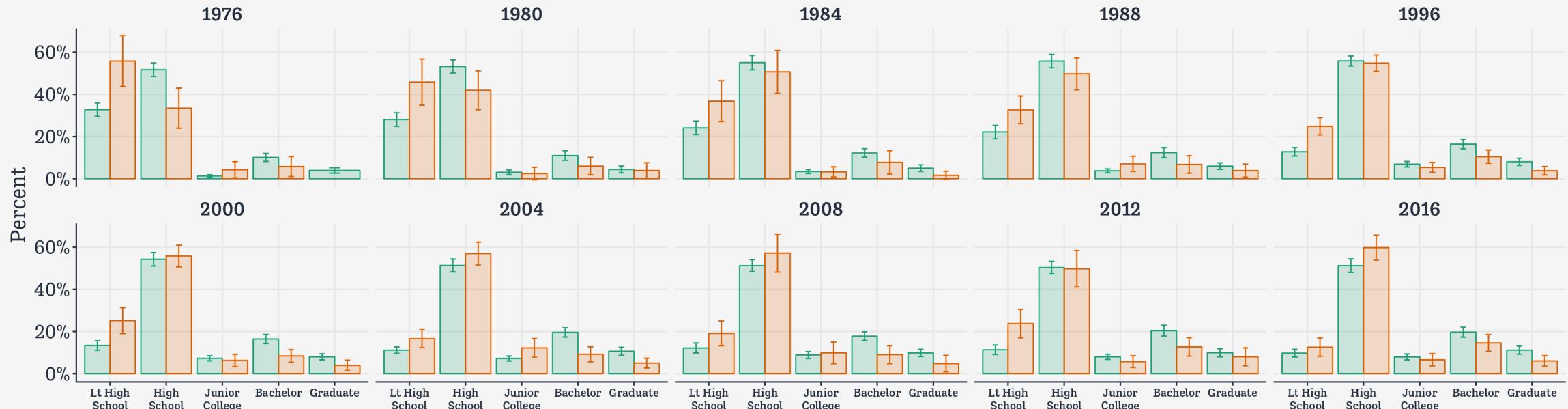
Race █ White █ Black



## Educational Attainment by Race

GSS 1976-2016

Race ■ White ■ Black



Is this figure effective? Not really!

# Let's try a different view

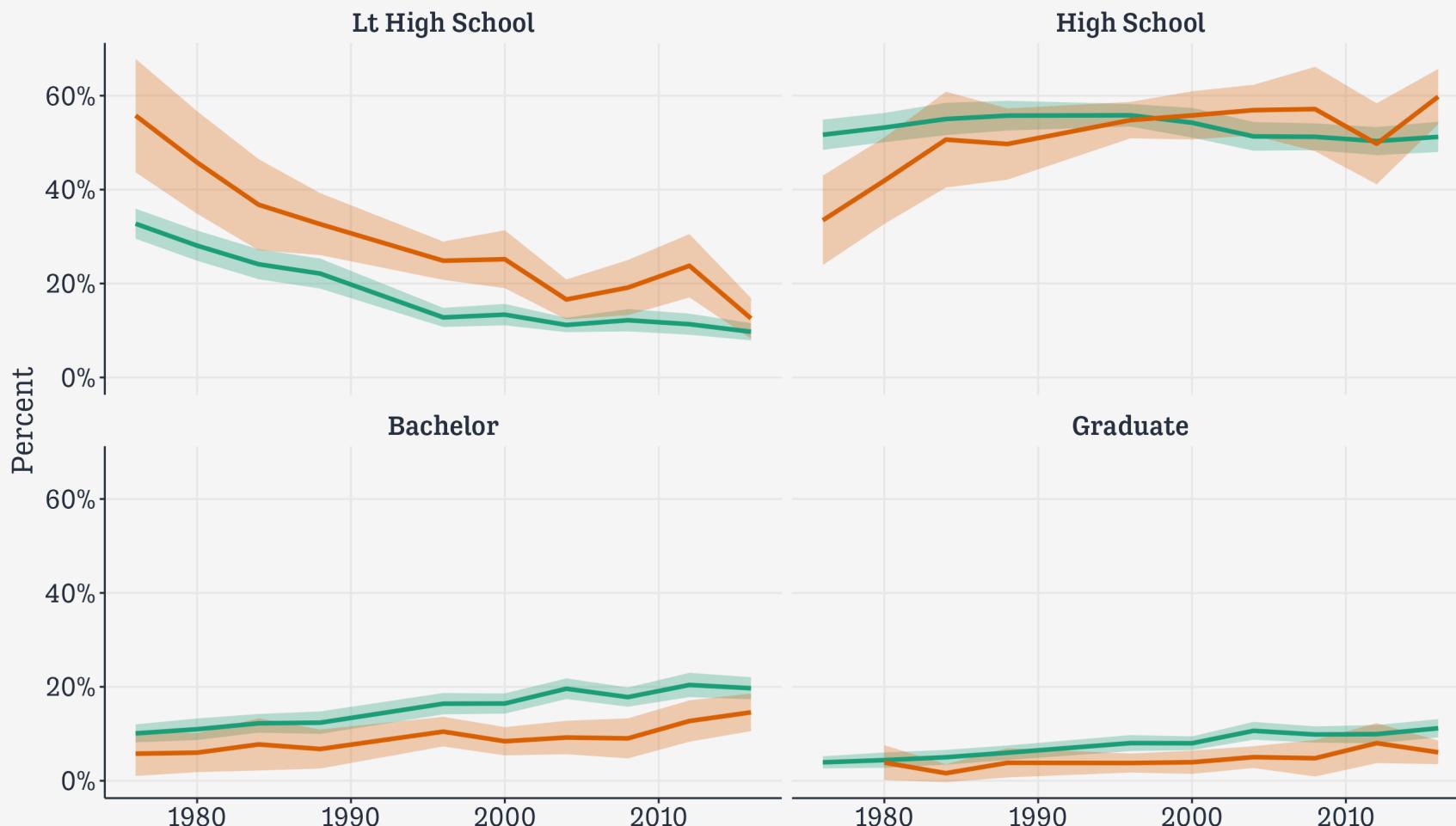
```
p <- out_yrd %>
  drop_na() %>
  filter(race %nin% "Other",
         degree %nin% "Junior College") %>
ggplot(mapping = aes(x = year, y = prop,
                      ymin = prop - 2*prop_se,
                      ymax = prop + 2*prop_se,
                      fill = race, color = race,
                      group = race))

p_out <- p +
  geom_ribbon(mapping = aes(color = NULL),
              alpha = 0.3) +
  geom_line(linewidth = rel(1.25)) +
  scale_y_continuous(labels = label_percent()) +
  scale_color_brewer(type = "qual", palette = "Dark2") +
  scale_fill_brewer(type = "qual", palette = "Dark2") +
  facet_wrap(~ degree, ncol = 2) +
  labs(title = "Educational Attainment by Race",
       subtitle = "GSS 1976-2016", fill = "Race",
       color = "Race", x = NULL, y = "Percent")
```

## Educational Attainment by Race

GSS 1976-2016

Race ■ White ■ Black



# Two other good packages: ggeffects

ggeffects 1.1.2.1  News Reference Introductions ▾ Plotting ▾ Practical Examples ▾ Technical Details ▾ 

## ggeffects - Estimated Marginal Means and Adjusted Predictions from Regression Models

Lüdecke D (2018). *ggeffects: Tidy Data Frames of Marginal Effects from Regression Models*. Journal of Open Source Software, 3(26), 772. doi: [10.21105/joss.00772](https://doi.org/10.21105/joss.00772)



### Why do we need (marginal/conditional) effects or (adjusted) predicted values?

Results of regression models are typically presented as tables that are easy to understand. For more complex models that include interaction or quadratic / spline terms, tables with numbers are less helpful and difficult to interpret. In such cases, *marginal effects* or *adjusted predictions* are far easier to understand. In particular, the visualization of such effects or predictions allows to intuitively get the idea of how predictors and outcome are associated, even for complex models.

### Aim of this package

**ggeffects** is a light-weight package that aims at easily calculating marginal effects and adjusted predictions (or: *estimated marginal means*) at the mean or at representative values of covariates ([see definitions here](#)) from statistical models, i.e. **predictions generated by a model when one holds the non-focal variables constant and varies the focal variable(s)**. This is achieved by three core ideas that describe the philosophy of the function design:

1. Functions are type-safe and always return a data frame with the same, consistent structure;

### Links

[View on CRAN](#)

[Browse source code](#)

[Report a bug](#)

### License

[GPL-3](#)

### Citation

[Citing ggeffects](#)

### Developers

[Daniel Lüdecke](#)

Author, maintainer 

[More about authors...](#)

### Dev status

CRAN 1.1.2

JOSS 10.21105/joss.00772

documentation [ggeffects](#)

downloads 30K/month

downloads 819K

# Two other good packages: interactions

ggeffects 1.1.2.1  News Reference Introductions ▾ Plotting ▾ Practical Examples ▾ Technical Details ▾ 

## ggeffects - Estimated Marginal Means and Adjusted Predictions from Regression Models

Lüdecke D (2018). *ggeffects: Tidy Data Frames of Marginal Effects from Regression Models*. Journal of Open Source Software, 3(26), 772. doi: [10.21105/joss.00772](https://doi.org/10.21105/joss.00772)



### Why do we need (marginal/conditional) effects or (adjusted) predicted values?

Results of regression models are typically presented as tables that are easy to understand. For more complex models that include interaction or quadratic / spline terms, tables with numbers are less helpful and difficult to interpret. In such cases, *marginal effects* or *adjusted predictions* are far easier to understand. In particular, the visualization of such effects or predictions allows to intuitively get the idea of how predictors and outcome are associated, even for complex models.

### Aim of this package

**ggeffects** is a light-weight package that aims at easily calculating marginal effects and adjusted predictions (or: *estimated marginal means*) at the mean or at representative values of covariates ([see definitions here](#)) from statistical models, i.e. **predictions generated by a model when one holds the non-focal variables constant and varies the focal variable(s)**. This is achieved by three core ideas that describe the philosophy of the function design:

1. Functions are type-safe and always return a data frame with the same, consistent structure;

### Links

[View on CRAN](#)

[Browse source code](#)

[Report a bug](#)

### License

[GPL-3](#)

### Citation

[Citing ggeffects](#)

### Developers

[Daniel Lüdecke](#)

Author, maintainer 

[More about authors...](#)

### Dev status

CRAN 1.1.2

JOSS 10.21105/joss.00772

documentation [ggeffects](#)

downloads 30K/month

downloads 819K

**Look for packages  
that work tidily**

