An Introduction to R for the Social Sciences

Tables, Regression Tables and Coefficient Plots

Benjamin Elbers
be2239@columbia.edu

Introduction

- There are several packages for regression **tables**:
  - 'modelsummary' is relatively new, but extremely flexible—we'll use this
  - There are also 'texreg', 'gtsummary', and a few others
  - You'll often find references to the 'stargazer' package, but this package hasn't been updated in a long time and misses some useful options.
- For other tables (e.g., "Table 1") there are also several options, but one flexible option is provided by `datasummary()` from 'modelsummary'
- For coefficient **plots**, (you guessed it) 'modelsummary' provides a `modelplot()` function.
  - For more customization, it's also not hard to plot the coefficients ourselves with 'ggplot2'
  - See also the package 'dotwhisker'

Note

- These slides require the most recent version of ‘modelsummary’ (0.8.0), so it’s a good idea to update:

```
install.packages("modelsummary")
```

Overview

datasummary

- 'modelsummary' provides the powerful function `datasummary()` (docs)
- The "capitalized" functions `Mean`, `SD`, `Median`, etc. are provided by the package and automatically drop NA.
- Start by specifying a formula and the dataset:

```
library("modelsummary")
g <- filter(gapminder, year %in% c(1952, 2007))

datasummary(lifeExp ~ Mean, data = g)
```

	Mean
lifeExp	58.03

datasummary

- Add more variables/statistics with “+”

```
datasummary(lifeExp + gdpPercap ~ Mean + SD, data = g)
```

	Mean	SD
lifeExp	58.03	15.10
gdpPercap	7702.67	11897.90

datasummary

- Add a "*" for nesting:

```
datasummary((lifeExp + gdpPercap) * continent ~ Mean + SD, data = g)
```

	continent	Mean	SD
lifeExp	Africa	46.97	11.00
	Americas	63.44	12.56
	Asia	58.52	15.00
	Europe	71.03	8.30
	Oceania	74.99	6.63
gdpPercap	Africa	2170.80	2794.95
	Americas	7541.05	7928.03
	Asia	8834.26	16823.87
	Europe	15357.77	12993.47
	Oceania	20054.14	11883.33

datasummary

- The order is meaningful. This is the other way around:

```
datasummary(continent * (lifeExp + gdpPercap) ~ Mean + SD, data = g)
```

continent		Mean	SD
Africa	lifeExp	46.97	11.00
	gdpPercap	2170.80	2794.95
Americas	lifeExp	63.44	12.56
	gdpPercap	7541.05	7928.03
Asia	lifeExp	58.52	15.00
	gdpPercap	8834.26	16823.87
Europe	lifeExp	71.03	8.30
	gdpPercap	15357.77	12993.47
Oceania	lifeExp	74.99	6.63
	gdpPercap	20054.14	11883.33

datasummary

- Nesting works on both sides:

```
# Factor() to turn numeric variables into categories
datasummary((lifeExp + gdpPercap) * continent
  ~ Factor(year) * (Mean + SD), data = g)
```

		1952		2007	
	continent	Mean	SD	Mean	SD
lifeExp	Africa	39.14	5.15	54.81	9.63
	Americas	53.28	9.33	73.61	4.44
	Asia	46.31	9.29	70.73	7.96
	Europe	64.41	6.36	77.65	2.98
	Oceania	69.25	0.19	80.72	0.73
gdpPercap	Africa	1252.57	982.95	3089.03	3618.16
	Americas	4079.06	3001.73	11003.03	9713.21
	Asia	5195.48	18634.89	12473.03	14154.94
	Europe	5661.06	3114.06	25054.48	11800.34
	Oceania	10298.09	365.56	29810.19	6540.99

datasummary

- Nesting applies only to the “multiplied” functions:

```
datasummary(lifeExp * continent
  ~ Factor(year) * (Mean + SD) + N + Percent(), data = g)
```

		1952		2007			
	continent	Mean	SD	Mean	SD	N	Percent
lifeExp	Africa	39.14	5.15	54.81	9.63	104	36.62
	Americas	53.28	9.33	73.61	4.44	50	17.61
	Asia	46.31	9.29	70.73	7.96	66	23.24
	Europe	64.41	6.36	77.65	2.98	60	21.13
	Oceania	69.25	0.19	80.72	0.73	4	1.41

datasummary

- Add a "1" to get an overall summary—again, the location is meaningful:

```
datasummary(lifeExp * (continent + 1)
  ~ Factor(year) * (Mean + SD) + N + Percent(), data = g)
```

		1952		2007			
	continent	Mean	SD	Mean	SD	N	Percent
lifeExp	Africa	39.14	5.15	54.81	9.63	104	36.62
	Americas	53.28	9.33	73.61	4.44	50	17.61
	Asia	46.31	9.29	70.73	7.96	66	23.24
	Europe	64.41	6.36	77.65	2.98	60	21.13
	Oceania	69.25	0.19	80.72	0.73	4	1.41
	All	49.06	12.23	67.01	12.07	284	100.00

datasummary

- Custom summary functions:

```
minmax <- function(x)
  paste0("[", round(min(x), 2), ", ", round(max(x), 2), "]")
datasummary(lifeExp * continent
  ~ N + Factor(year) * (Mean + minmax), data = g)
```

			1952		2007	
	continent	N	Mean	minmax	Mean	minmax
lifeExp	Africa	104	39.14	[30, 52.72]	54.81	[39.61, 76.44]
	Americas	50	53.28	[37.58, 68.75]	73.61	[60.92, 80.65]
	Asia	66	46.31	[28.8, 65.39]	70.73	[43.83, 82.6]
	Europe	60	64.41	[43.58, 72.67]	77.65	[71.78, 81.76]
	Oceania	4	69.25	[69.12, 69.39]	80.72	[80.2, 81.24]

datasummary

- Renaming variables, and this is pretty much publication-ready.
- Either rename in the data frame:

```
g2 <- rename(g, Continent = continent, `Life Exp.` = lifeExp)
datasummary((`Life Exp.`) * (Continent + 1)
  ~ Factor(year) * (Mean + SD) + N + Percent(), data = g2)
```

		1952		2007			
	Continent	Mean	SD	Mean	SD	N	Percent
Life Exp.	Africa	39.14	5.15	54.81	9.63	104	36.62
	Americas	53.28	9.33	73.61	4.44	50	17.61
	Asia	46.31	9.29	70.73	7.96	66	23.24
	Europe	64.41	6.36	77.65	2.98	60	21.13
	Oceania	69.25	0.19	80.72	0.73	4	1.41
	All	49.06	12.23	67.01	12.07	284	100.00

datasummary

- Or:

```
datasummary(Heading("Life Exp.")*lifeExp *
  (Heading("Continent")*continent + 1)
  ~ Factor(year) * (Mean + SD) +
  N + Heading("%")*Percent(), data = g)
```

		1952		2007			
	Continent	Mean	SD	Mean	SD	N	%
Life Exp.	Africa	39.14	5.15	54.81	9.63	104	36.62
	Americas	53.28	9.33	73.61	4.44	50	17.61
	Asia	46.31	9.29	70.73	7.96	66	23.24
	Europe	64.41	6.36	77.65	2.98	60	21.13
	Oceania	69.25	0.19	80.72	0.73	4	1.41
	All	49.06	12.23	67.01	12.07	284	100.00

datasummary

- `datasummary()` is very useful interactively for quick exploratory tables, but with a bit of polishing the tables are publication-ready.
- Export the table:

```
f <- <the summary formula>
datasummary(f, data = <dataframe>, output = "table.html")
datasummary(f, data = <dataframe>, output = "table.tex")
datasummary(f, data = <dataframe>, output = "table.docx")
```

- Many other customization options are available.

datasummary

- There are also some helper functions, such as `datasummary_crosstab()`:

```
datasummary_crosstab(cyl ~ vs * am,
  statistic = 1 ~ 1 + N + Percent(),
  data = mtcars)
```

		vs				
		0		1		
cyl		0	1	0	1	All
4	N	0	1	3	7	11
	%	0.0	3.1	9.4	21.9	34.4
6	N	0	3	4	0	7
	%	0.0	9.4	12.5	0.0	21.9
8	N	12	2	0	0	14
	%	37.5	6.2	0.0	0.0	43.8
All	N	12	6	7	7	32
	%	37.5	18.8	21.9	21.9	100.0

Exercises A

Use the `flights` dataset from the `nycflights13` package to recreate the following table. Output the table in tex, html, and docx format.

origin		Dep. Delay (mins.)	Proportion Dep. Delay >30min
EWR	Mean	15.11	0.17
	SD	41.32	0.38
	N	117596	117596
	N cancelled	3239	3239
JFK	Mean	12.11	0.14
	SD	39.04	0.35
	N	109416	109416
	N cancelled	1863	1863
LGA	Mean	10.35	0.13
	SD	39.99	0.34
	N	101509	101509
	N cancelled	3153	3153

(Getting “N cancelled” into the table is a bit tricky.)

Working with models

- The ‘broom’ and ‘parameters’ packages offer standardized mechanisms to extract information from statistical models and return data frames—exactly what we need for ggplot
- These packages are an attempt at standardization of the idiosyncracies of the different modeling packages—works well for some models, may not work so well with other packages
- ‘modelsummary’ is based on these packages—hence, if the model is supported by either ‘broom’ or ‘parameters’ it will automatically work in ‘modelsummary’ as well
- These functions are useful to work with models:
  - `broom::tidy()` and `parameters::model_parameters()` summarize information about model components
  - `broom::glance()` reports information about the entire model
  - `broom::augment()` adds informations about observations to a dataset

Working with models

- `broom::tidy()` for model summary:

```
library("broom")
mod <- lm(lifeExp ~ log(gdpPercap), data = gapminder)

broom::tidy(mod)
```

```
#> # A tibble: 2 x 5
#>   term           estimate std.error statistic  p.value
#>   <chr>             <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)       -9.10     1.23      -7.41 1.93e-13
#> 2 log(gdpPercap)     8.41     0.149     56.5  0
```

- Often useful: `broom::tidy(mod, conf.int = TRUE)`

Working with models

- `parameters::model_parameters()` for model summary:

```
library("parameters")
mod <- lm(lifeExp ~ log(gdpPercap), data = gapminder)

parameters::model_parameters(mod)
```

```
#> Parameter      | Coefficient |   SE |         95% CI | t(1702) |     p
#> ------------------------------------------------------------------------
#> (Intercept)    |       -9.10 | 1.23 | [-11.51, -6.69] |   -7.41 | < .001
#> gdpPercap [log] |        8.41 | 0.15 | [  8.11,  8.70] |   56.50 | < .001
```

Working with models

- `broom::glance()` for model statistics (model "metadata"):

```
broom::glance(mod) %>% glimpse()
```

```
#> Rows: 1
#> Columns: 12
#> $ r.squared     <dbl> 0.652
#> $ adj.r.squared <dbl> 0.652
#> $ sigma         <dbl> 7.62
#> $ statistic     <dbl> 3192
#> $ p.value       <dbl> 0
#> $ df            <dbl> 1
#> $ logLik        <dbl> -5877
#> $ AIC           <dbl> 11760
#> $ BIC           <dbl> 11777
#> $ deviance      <dbl> 98814
#> $ df.residual   <int> 1702
#> $ nobs          <int> 1704
```

Working with models

- `broom::augment()` to return a data frame with observation-specific model data, such as the fitted values and the standardized residuals:

```
broom::augment(mod)
```

```
#> # A tibble: 1,704 x 7
#>    lifeExp `log(gdpPercap)` .fitted    .hat .sigma  .cooksd
#>      <dbl>            <dbl>   <dbl>   <dbl>  <dbl>    <dbl>
#>  1    28.8             6.66    46.9 0.00144   7.61 0.00407
#>  2    30.3             6.71    47.3 0.00139   7.61 0.00345
#>  3    32.0             6.75    47.6 0.00134   7.61 0.00284
#>  4    34.0             6.73    47.5 0.00137   7.61 0.00213
#>  5    36.1             6.61    46.4 0.00151   7.62 0.00139
#>  6    38.4             6.67    46.9 0.00144   7.62 0.000895
#>  7    39.9             6.89    48.8 0.00120   7.62 0.000827
#>  8    40.8             6.75    47.6 0.00135   7.62 0.000536
#>  9    41.7             6.48    45.3 0.00167   7.62 0.000192
#> 10    41.8             6.45    45.1 0.00169   7.62 0.000168
#> # ... with 1,694 more rows, and 1 more variable:
#> #   .std.resid <dbl>
```

Regression tables

- We'll be using 'modelsummary', but depending on what kinds of models you're estimating, other packages might be more convenient
- Because `summary()` prints a lot of useless information, I also use `modelsummary()` interactively a lot—also makes it easy to compare models
- `modelsummary()` supports over 100 different kinds of models (all that are supported by broom/parameters, which is a lot), which also means that not all models are necessarily *perfectly* supported—always good to check the output table, especially if you rely on correct standard errors, AIC, BIC, ...

Regression tables

- Basic usage of `modelsummary()` is very similar to `datasummary()`:

```
mod1 <- lm(lifeExp ~ log(pop) + log(gdpPercap), data=gapminder)
mod2 <- lm(lifeExp ~ log(pop) + log(gdpPercap) + year, data=gapminder)

library("modelsummary")
modelsummary(list(mod1, mod2)) # show in RStudio's Viewer panel

# same output options:
modelsummary(list(mod1, mod2), output = "table.html")
modelsummary(list(mod1, mod2), output = "table.tex")
modelsummary(list(mod1, mod2), output = "table.docx") # etc.
```

Regression tables

- This is what the output looks like:

```
modelsummary(list(mod1, mod2), stars = TRUE)
```

	Model 1	Model 2
(Intercept)	-28.771***	-369.699***
	(2.076)	(19.189)
log(pop)	1.279***	0.885***
	(0.111)	(0.104)
log(gdpPercap)	8.344***	7.786***
	(0.143)	(0.135)
year		0.178***
		(0.010)
Num.Obs.	1704	1704
R2	0.677	0.728
R2 Adj.	0.677	0.728
AIC	11634.1	11343.3
BIC	11655.9	11370.5
Log.Lik.	-5813.073	-5666.639
F	1786.413	1519.705

+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

Customization

- Use a named list to rename the models:

```
modelsummary(list("Base Model" = mod1, "Extended Model" = mod2))
```

	Base Model	Extended Model
(Intercept)	-28.771	-369.699
	(2.076)	(19.189)
log(pop)	1.279	0.885
	(0.111)	(0.104)
log(gdpPercap)	8.344	7.786
	(0.143)	(0.135)
year		0.178
		(0.010)
Num.Obs.	1704	1704
R2	0.677	0.728
R2 Adj.	0.677	0.728
AIC	11634.1	11343.3
BIC	11655.9	11370.5
Log.Lik.	-5813.073	-5666.639
F	1786.413	1519.705

Customization

- The `estimate` and `statistic` can be modified separately:

```
modelsummary(list(mod1, mod2), fmt = 1,
  estimate = "{estimate} {stars}",
  statistic = "{std.error} [{conf.low}, {conf.high}]")
```

	Model 1	Model 2
(Intercept)	-28.8 ***	-369.7 ***
	2.1 [-32.8, -24.7]	19.2 [-407.3, -332.1]
log(pop)	1.3 ***	0.9 ***
	0.1 [1.1, 1.5]	0.1 [0.7, 1.1]
log(gdpPercap)	8.3 ***	7.8 ***
	0.1 [8.1, 8.6]	0.1 [7.5, 8.1]
year		0.2 ***
		0.0 [0.2, 0.2]
Num.Obs.	1704	1704
R2	0.677	0.728
R2 Adj.	0.677	0.728
AIC	11634.1	11343.3
BIC	11655.9	11370.5
Log.Lik.	-5813.073	-5666.639
F	1786.413	1519.705

Customization

- e.g., a more compact display:

```
modelsummary(list(mod1, mod2),
  estimate = "{estimate} {stars} ({std.error})",
  statistic = NULL)
```

	Model 1	Model 2
(Intercept)	-28.771 *** (2.076)	-369.699 *** (19.189)
log(pop)	1.279 *** (0.111)	0.885 *** (0.104)
log(gdpPercap)	8.344 *** (0.143)	7.786 *** (0.135)
year		0.178 *** (0.010)
Num.Obs.	1704	1704
R2	0.677	0.728
R2 Adj.	0.677	0.728
AIC	11634.1	11343.3
BIC	11655.9	11370.5
Log.Lik.	-5813.073	-5666.639
F	1786.413	1519.705

Customization

- Rename coefficients using a named vector (see also `coef_map`):

```
modelsummary(list(mod1, mod2),
  coef_rename = c("log(gdpPercap)" = "log(GDP per capita)",
                  "log(Pop)" = "log(Population)"))
```

	Model 1	Model 2
(Intercept)	-28.771	-369.699
	(2.076)	(19.189)
log(pop)	1.279	0.885
	(0.111)	(0.104)
log(GDP per capita)	8.344	7.786
	(0.143)	(0.135)
year		0.178
		(0.010)
Num.Obs.	1704	1704
R2	0.677	0.728
R2 Adj.	0.677	0.728
AIC	11634.1	11343.3
BIC	11655.9	11370.5
Log.Lik.	-5813.073	-5666.639
F	1786.413	1519.705

Customization

- Remove goodness-of-fit statistics (see also `gof_map`):

```
modelsummary(list(mod1, mod2), gof_omit = "AIC|BIC|F|Lik")
```

	Model 1	Model 2
(Intercept)	-28.771	-369.699
	(2.076)	(19.189)
log(pop)	1.279	0.885
	(0.111)	(0.104)
log(gdpPercap)	8.344	7.786
	(0.143)	(0.135)
year		0.178
		(0.010)
Num.Obs.	1704	1704
R2	0.677	0.728
R2 Adj.	0.677	0.728

Customization

- Omitting coefficients:

```
mod1f <- lm(lifeExp ~ log(pop) + log(gdpPercap), data=gapminder)
mod2f <- lm(lifeExp ~ log(pop) + log(gdpPercap) + as.factor(year),
  data=gapminder)
modelsummary(list(mod1f, mod2f))
```

	Model 1	Model 2
(Intercept)	-28.771	-23.323
	(2.076)	(1.979)
log(pop)	1.279	0.881
	(0.111)	(0.104)
log(gdpPercap)	8.344	7.784
	(0.143)	(0.135)
as.factor(year)1957		1.265
		(0.798)
as.factor(year)1962		2.307
		(0.799)
as.factor(year)1967		3.122
		(0.801)
as.factor(year)1972		3.744
		(0.803)
as.factor(year)1977		4.777

Customization

- Omitting coefficients:

```
modelsummary(list(mod1f, mod2f), coef_omit = "year",
        add_rows = data.frame("Year fixed-effects", "No", "Yes"))
```

	Model 1	Model 2
(Intercept)	-28.771	-23.323
	(2.076)	(1.979)
log(pop)	1.279	0.881
	(0.111)	(0.104)
log(gdpPercap)	8.344	7.784
	(0.143)	(0.135)
Num.Obs.	1704	1704
R2	0.677	0.731
R2 Adj.	0.677	0.729
AIC	11634.1	11347.1
BIC	11655.9	11428.7
Log.Lik.	-5813.073	-5658.566
F	1786.413	353.195
Year fixed-effects	No	Yes

Standard Errors

- Different standard errors—this works well for `lm` models, but not necessarily for other kinds of models you're estimating:

```
modelsummary(mod2, vcov = list("iid", ~country, "robust"))
```

	Model 1	Model 2	Model 3
(Intercept)	-369.699	-369.699	-369.699
	(19.189)	(34.177)	(21.362)
log(pop)	0.885	0.885	0.885
	(0.104)	(0.310)	(0.109)
log(gdpPercap)	7.786	7.786	7.786
	(0.135)	(0.351)	(0.135)
year	0.178	0.178	0.178
	(0.010)	(0.018)	(0.011)
Num.Obs.	1704	1704	1704
R2	0.728	0.728	0.728
R2 Adj.	0.728	0.728	0.728
AIC	11343.3	11343.3	11343.3
BIC	11370.5	11370.5	11370.5
Log.Lik.	-5666.639	-5666.639	-5666.639
F	1519.705	1519.705	1519.705
Std. Errors	IID	C: country	Robust

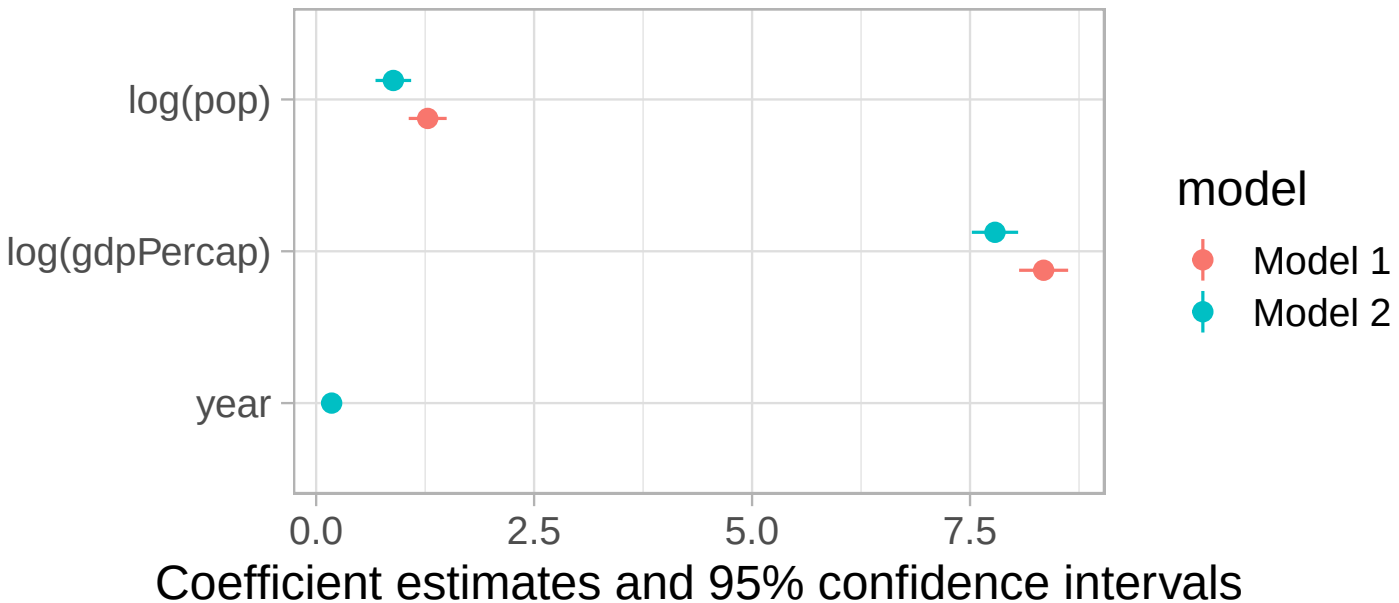
Exercises B

1. Use `webuse::webuse("margex")` to load the `margex` dataset. Estimate two logistic regression models, using `outcome` and a number of predictors, including interactions and/or polynomials. Create a polished regression table.
2. Using the same two models, explore what happens when you run `margins()` on the models and pass these objects to modelsummary.

Plotting regression coefficients

- Again, a simple starting point is provided by the ‘modelsummary‘ package:

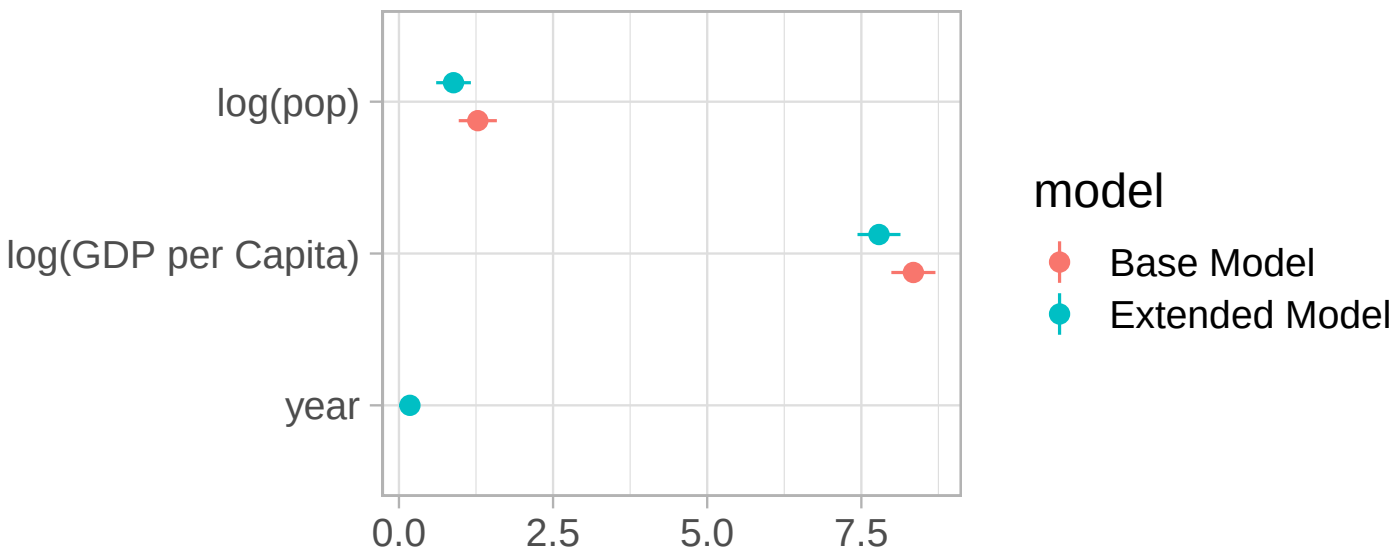
```
modelplot(list(mod1, mod2), coef_omit = "Intercept") +
  theme_light(base_size = 16)
```



Plotting regression coefficients

- Works the same way as `modelsummary()`:

```
(p <- modelplot(list("Base Model" = mod1, "Extended Model" = mod2),
    coef_omit = "Intercept",
    conf_level = .99, vcov = "robust",
    coef_rename = c("log(gdpPercap)" = "log(GDP per Capita)")) +
  theme_light(base_size = 16))
```

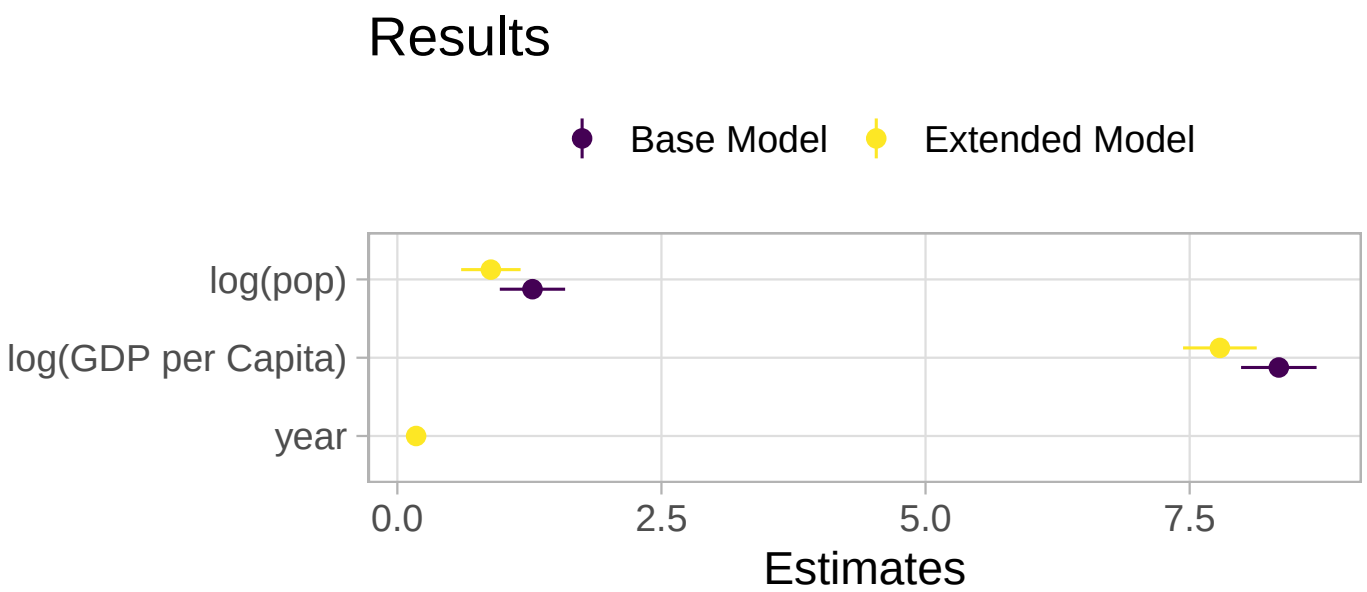


Coefficient estimates and 99% confidence intervals

Plotting regression coefficients

- `modelplot()` returns a ggplot2 plot—so we know already how to customize:

```
p + labs(color = NULL, x = "Estimates", title = "Results") +
    scale_color_viridis_d() +
    theme(panel.grid.minor = element_blank(),
          legend.position = "top")
```



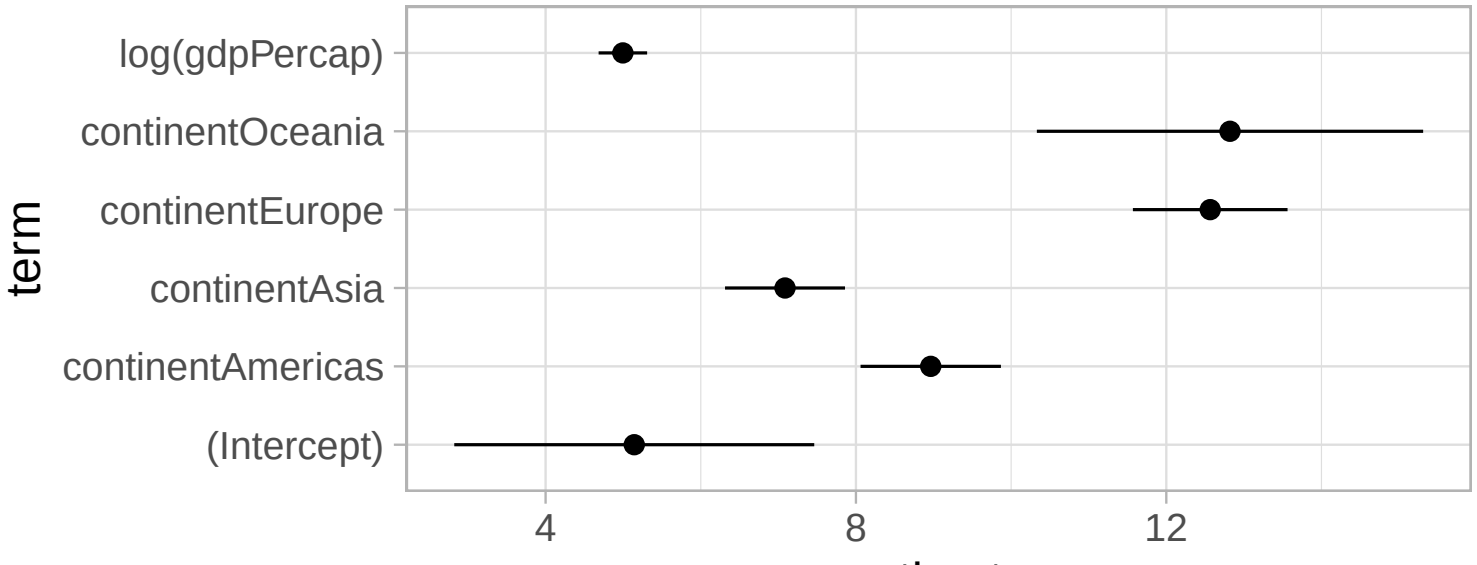
The manual way

- It's also not hard to do this ourselves (and to plot things other than coefficients):

1. Estimate one or several models.
2. Turn model output into a data frame using
   - `broom::tidy()` or `parameters::model_parameters()` to plot regression coefficients, standard errors, confidence intervals, ...
   - `broom::glance()` to plot model fit statistics such as AIC, R^2, ...
   - `broom::augment()` to plot residuals, fitted values, outliers statistics....
3. Manipulate the data frame, using `mutate`, `pivot_*`, `filter`, ...
4. Plot the data frame using 'ggplot2'.

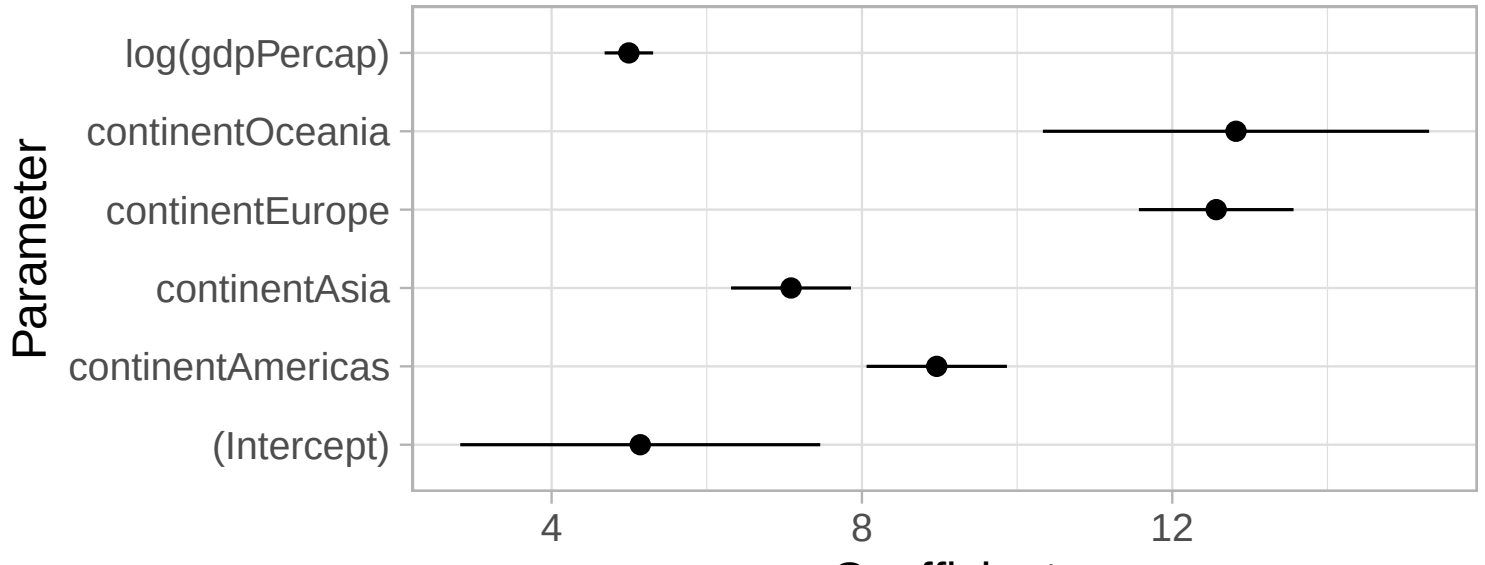
The manual way: broom

```
# 1. estimate model(s)
mod <- lm(lifeExp ~ log(gdpPercap) + as.factor(year) + continent,
  data = gapminder)
# 2. extract as data frame
moddf <- broom::tidy(mod, conf.int = TRUE)
# 3. manipulate (here: remove year FE)
moddf <- filter(moddf, str_detect(term, "year", negate = TRUE))
# 4. plot
ggplot(moddf, aes(y = term, x = estimate,
                  xmin = conf.low, xmax = conf.high)) +
  geom_pointrange()
```



The manual way: parameters

```
# 1. estimate model(s)
mod <- lm(lifeExp ~ log(gdpPercap) + as.factor(year) + continent,
  data = gapminder)
# 2. extract as data frame
moddf <- parameters::model_parameters(mod)
# 3. manipulate (remove year FE)
moddf <- filter(moddf, str_detect(Parameter, "year", negate = TRUE))
# 4. plot
ggplot(moddf, aes(y = Parameter, x = Coefficient,
                  xmin = CI_low, xmax = CI_high)) +
  geom_pointrange()
```



The manual way: many models

- This can get a bit annoying with many models:

```
# fit two models
mod1 <- lm(lifeExp ~ log(gdpPercap) + as.factor(year) + continent,
  data = gapminder)
mod2 <- lm(lifeExp ~ log(gdpPercap) + log(pop) +
  as.factor(year) + continent, data = gapminder)
mod3 <- lm(lifeExp ~ log(gdpPercap) + pop +
  as.factor(year) + continent, data = gapminder)

# extract data frames
mod1df <- broom::tidy(mod1, conf.int = TRUE) %>%
  mutate(model = "excluding pop")
mod2df <- broom::tidy(mod2, conf.int = TRUE) %>%
  mutate(model = "including log(pop)")
mod3df <- broom::tidy(mod3, conf.int = TRUE) %>%
  mutate(model = "including pop")

# merge
moddf <- bind_rows(mod1df, mod2df, mod3df) %>%
  filter(str_detect(term, "year", negate = TRUE))
```

The manual way: three models

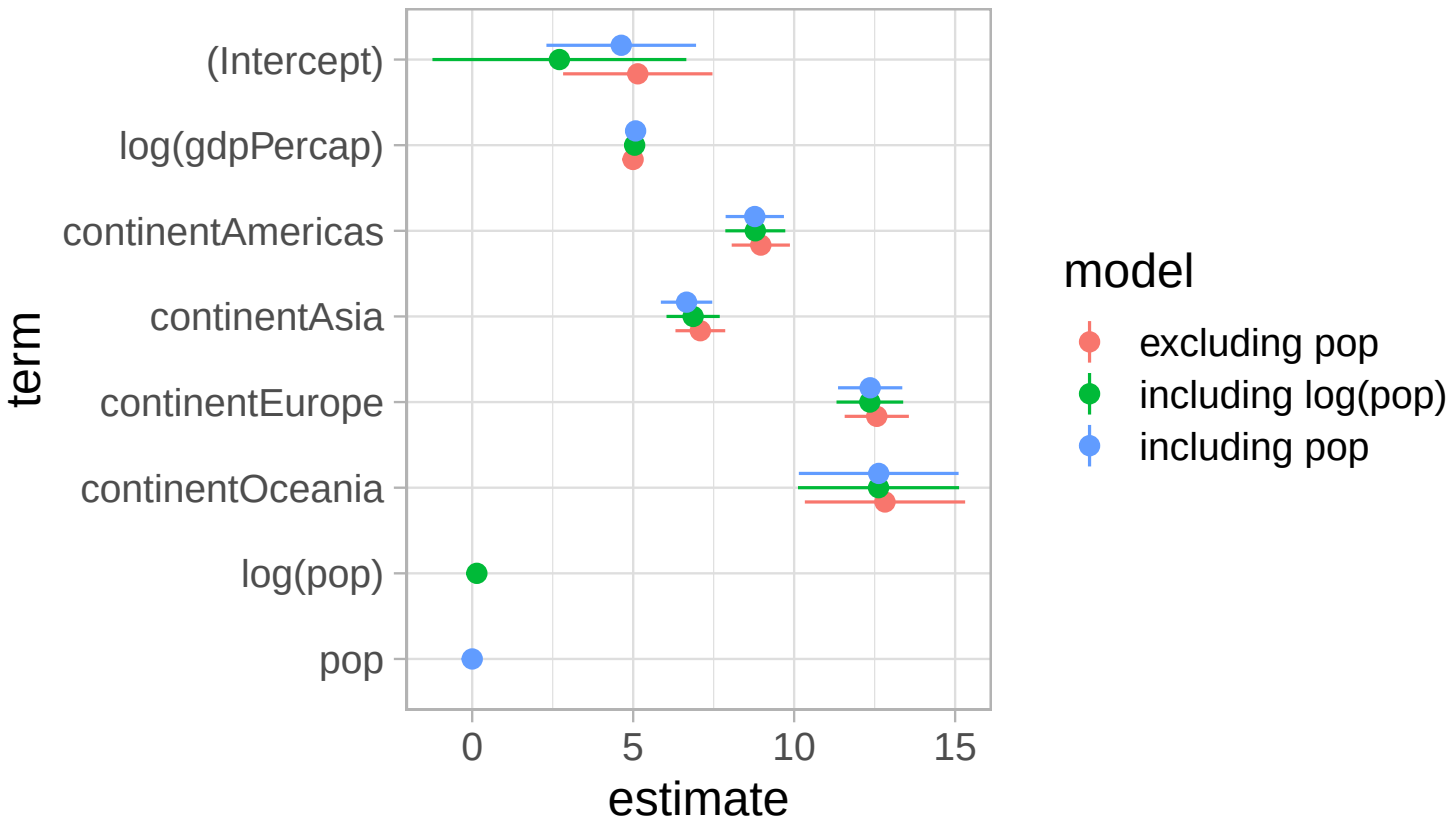
- 'modelsummary' comes to the rescue (again):

```
models <- list("excluding pop" = mod1,
  "including log(pop)" = mod2,
  "including pop" = mod3)
moddf <- modelplot(models, draw = FALSE, coef_omit = "year")
head(moddf, 5)
```

```
#>                 term                model      estimate
#> 20               pop      including pop  0.0000000054
#> 13          log(pop) including log(pop)  0.1441474914
#> 6  continentOceania      excluding pop 12.8210983160
#> 12 continentOceania including log(pop) 12.6211042509
#> 19 continentOceania      including pop 12.6263743957
#>         std.error      conf.low      conf.high
#> 20 0.00000000137  0.0000000027  0.00000000809
#> 13 0.09626383959 -0.0446617100  0.33295669284
#> 6  1.26928452484 10.3315602249 15.31063640709
#> 12 1.27582746548 10.1187319589 15.12347654283
#> 19 1.26486164406 10.1455101590 15.10723863230
```

The manual way: three models

```
ggplot(moddf, aes(y = term, x = estimate, color = model,
  xmin = conf.low, xmax = conf.high)) +
  geom_pointrange(position = position_dodge(width = 0.5))
```



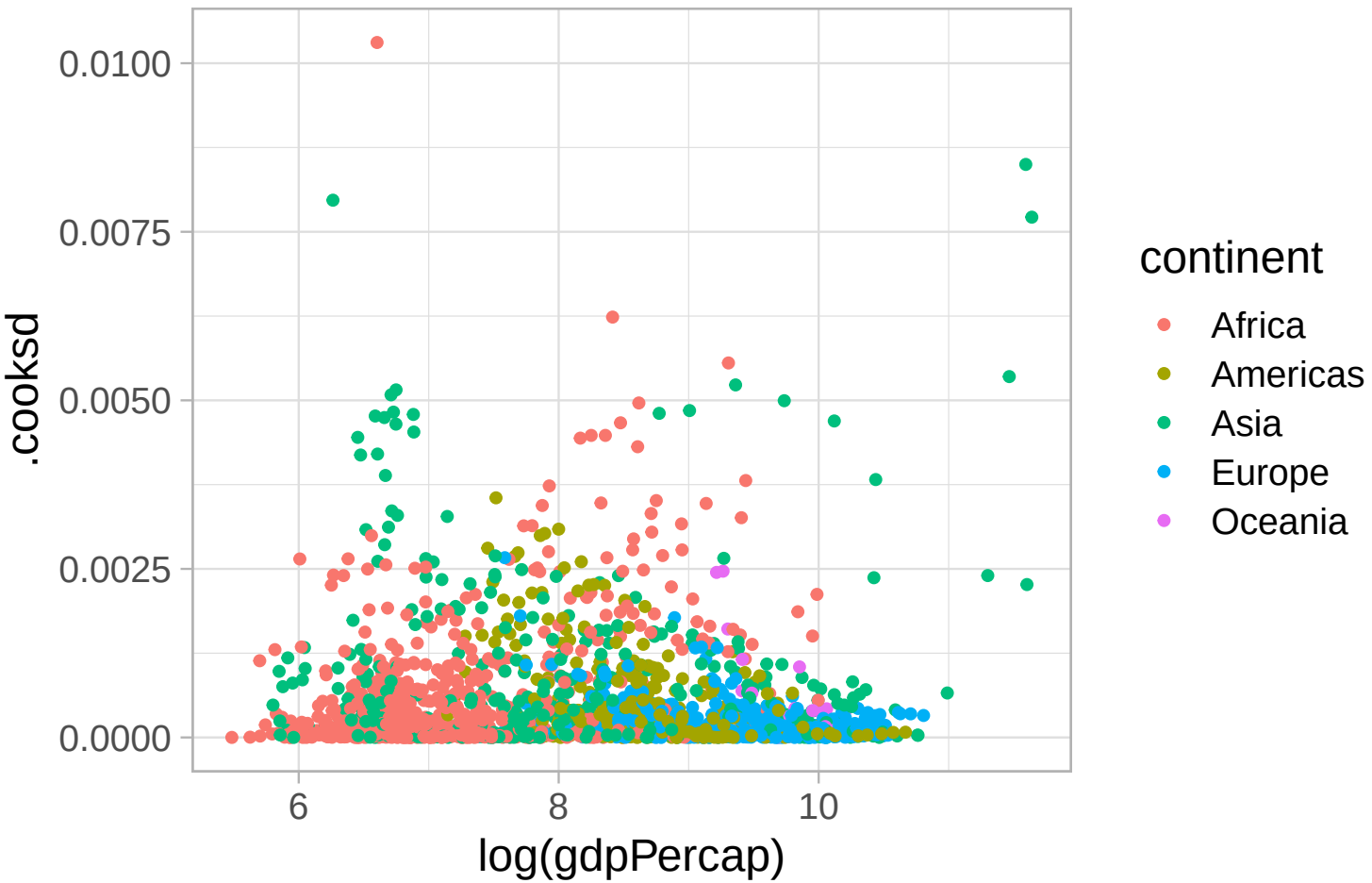
The manual way

- The previous slides showed how to plot regression coefficients.
- `broom::glance` and `broom::augment` can be used to extract tidy dataframes of model summary statistics (e.g., R^2) and case-specific statistics (e.g., residuals or fitted values), respectively.
- For instance,

```
mod <- lm(lifeExp ~ log(gdpPercap) + as.factor(year) + continent,
  data = gapminder)
p <- ggplot(broom::augment(mod),
  aes(x = `log(gdpPercap)`, y = .cooksd, color = continent)) +
    geom_point()
```

The manual way: `broom::augment`

p



An advanced pattern

- If we want to fit *many* models (for instance, for a specification curve), there are a few advanced patterns that can come in very useful.
- To understand this example, it is useful to read up on the `lapply()` function, and to know that we can store a list or a model *inside* a 'tibble' cell.
- First, we create a dataframe that holds the different model formulae that we want to estimate:

```
models <- tribble(
~model, ~pred,
"Model 1", "log(gdpPercap) + as.factor(year) + continent",
"Model 2", "log(gdpPercap) + year + continent",
"Model 3", "log(gdpPercap) + year + log(pop) + continent",
"Model 4", "log(gdpPercap) + as.factor(year) + log(pop) + continent")

# fit a model based on predictors in pred
fit_model <- function(pred) {
  lm(paste("lifeExp ~ ", pred), data = gapminder)
}
```

An advanced pattern

- Now we fit the models and store them inside the tibble; then we tidy each model and store the result in the 'tidied' column

```
tidy_confint <- function(x) broom::tidy(x, conf.int = TRUE)
moddf <- mutate(models,
    lm = lapply(pred, fit_model),
    tidied = lapply(lm, tidy_confint))

moddf
```

```
#> # A tibble: 4 x 4
#>   model  pred                                lm     tidied
#>   <chr>  <chr>                               <list> <list>
#> 1 Model~ log(gdpPercap) + as.factor(ye~ <lm>   <tibble[,7] ~
#> 2 Model~ log(gdpPercap) + year + conti~ <lm>   <tibble[,7] ~
#> 3 Model~ log(gdpPercap) + year + log(p~ <lm>   <tibble[,7] ~
#> 4 Model~ log(gdpPercap) + as.factor(ye~ <lm>   <tibble[,7] ~
```

An advanced pattern

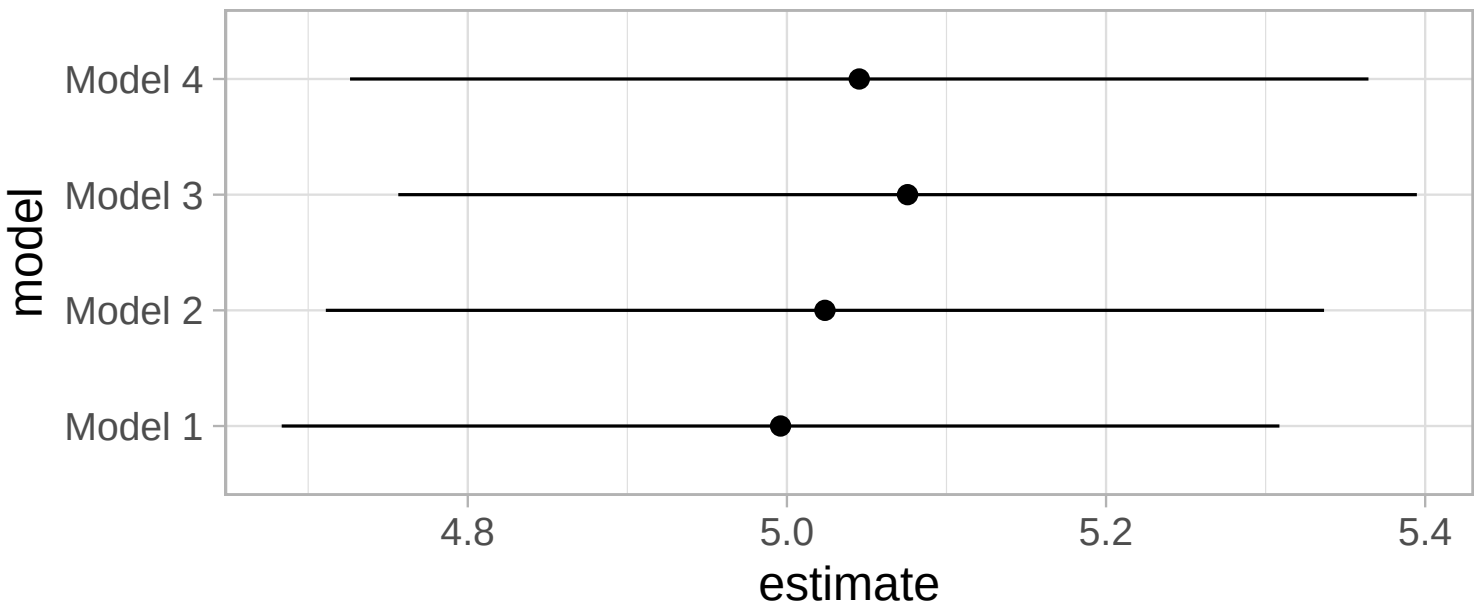
- The last step is to expand the 'tidied' column so that we arrive at a dataframe in long format—now we're ready to plot:

```
unnest(moddf, tidied) %>% select(model, term, estimate, std.error)
```

```
#> # A tibble: 50 x 4
#>    model   term                  estimate std.error
#>    <chr>   <chr>                    <dbl>     <dbl>
#>  1 Model 1 (Intercept)               5.14     1.18
#>  2 Model 1 log(gdpPercap)            5.00     0.159
#>  3 Model 1 as.factor(year)1957       1.75     0.687
#>  4 Model 1 as.factor(year)1962       3.23     0.688
#>  5 Model 1 as.factor(year)1967       4.56     0.690
#>  6 Model 1 as.factor(year)1972       5.72     0.693
#>  7 Model 1 as.factor(year)1977       7.13     0.695
#>  8 Model 1 as.factor(year)1982       8.87     0.696
#>  9 Model 1 as.factor(year)1987      10.5      0.697
#> 10 Model 1 as.factor(year)1992      11.5      0.696
#> # ... with 40 more rows
```

An advanced pattern

```
unnest(moddf, tidied) %>%
  filter(term == "log(gdpPercap)") %>%
 ggplot(aes(y = model, x = estimate, xmin = conf.low, xmax = conf.high)) +
    geom_pointrange()
```



- If you want to learn more, read chapter 21 in R for Data Science (especially 21.5), nested data, and broom and dplyr

From Stata

- It's easy to plot any data that you can obtain in the form of a dataframe
- The Stata program `parmest` can create a dataset from your models
- Simple example:

```
parmby "regr mpg weight, robust", by(foreign) label command
  saving("regresults.dta")
```

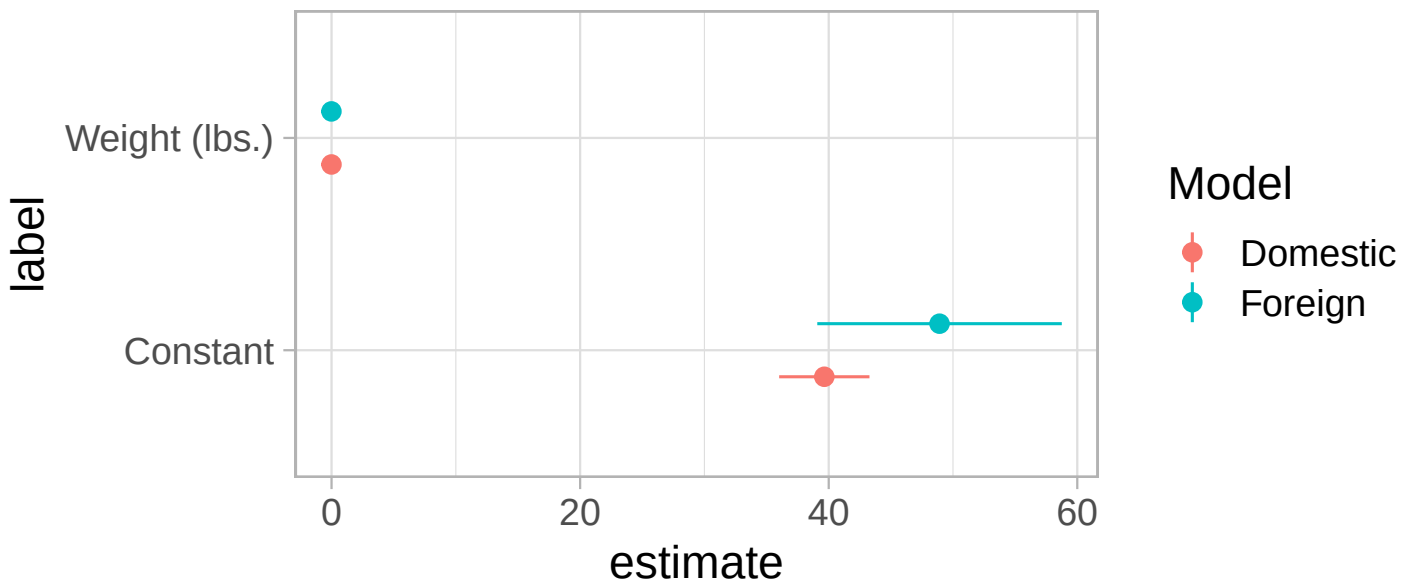
creates the following dataset:

	foreign	parmseq	command	parm	label	estimate	stderr
1	Domestic	1	regr mpg weight, robust	weight	Weight (lbs.)	-.00597508	.00054024
2	Domestic	2	regr mpg weight, robust	_cons	Constant	39.646965	1.8085882
3	Foreign	1	regr mpg weight, robust	weight	Weight (lbs.)	-.01042596	.00168877
4	Foreign	2	regr mpg weight, robust	_cons	Constant	48.918297	4.7155533

From Stata

- The rest is fairly straightforward:

```
haven::read_dta("regresults.dta") %>%
  haven::as_factor() %>%
  ggplot(aes(x = estimate, y = label, color = foreign,
         xmin = min95, xmax = max95)) +
    geom_pointrange(position = position_dodge(width = .5)) +
    labs(color = "Model")
```



From Stata

- There are also ways to call R directly from Stata, e.g. rcall.
- I'm not a fan of this workflow—adds complications for setup and reproducibility, the Stata editor is not helpful when writing R code, etc.

Exercises C

The following plot was produced using Stata's `coefplot`. Recreate this plot in R. The dataset uses is `auto`, and the two regressions that are shown are estimated in Stata by `reg price mpg trunk length turn if foreign==0` (or 1). You can go through `parmby` or estimate the regressions in R.