Big Data and Economics

Regression analysis in R

Kyle Coombs

Bates College | DCS/ECON 368

Contents

This lecture covers the bread-and-butter tool of applied econometrics and data science: regression analysis. This adapted from work by Grant McDermott. Today is focused on practical skills of using regression tools with R to do some analysis. I am not working through major econometric theory of regression analysis and inference. We're instead focusing on the practical skills of producing regression tables, interpreting coefficients, standardizing variables, and understanding the different types of standard errors.

There are also many R packages I'm leaving out! There are tons of ways to do regressions in R. I'll present just a few.

Software requirements

R packages

It's important to note that "base" R already provides all of the tools we need for basic regression analysis. However, we'll be using several additional packages today, because they will make our lives easier and offer increased power for some more sophisticated analyses.

- New: **fixest**, **estimatr**, **ivreg**, **sandwich**, **lmtest**, **mfx**, **margins**, **broom**, **modelsummary**, **vtable**
- Already used: **tidyverse**, **hrbrthemes**, **listviewer**, **tidycensus**, **tigris**

A convenient way to install (if necessary) and load everything is by running the below code chunk.

```
## Load and install the packages that we'll be using today
if (!require("pacman")) install.packages("pacman")
pacman::p_load(mfx, tidyverse, hrbrthemes, estimatr, ivreg, fixest, sandwich,
               lmtest, margins, vtable, broom, modelsummary,tidycensus,tigris)
## Make sure we have at least version 0.6.0 of ivreg
if (numeric_version(packageVersion("ivreg")) < numeric_version("0.6.0")) install.packages("ivreg")

## My preferred ggplot2 plotting theme (optional)
theme_set(theme_minimal())
```

While we've already loaded all of the required packages for today, I'll try to be as explicit about where a particular function is coming from, whenever I use it below.

Something else to mention up front is that we are using the Opportunity Atlas today for our regressions. Some of these

regressions will be a bit simple, but the point is to show you how to run them in R with data that are meaningful. We'll also use the `fips_codes` data from the **tidycensus** package to merge in county names and state abbreviations.

The Opportunity Atlas file lives on GitHub, so we'll download it from there (*Note*: I am using the githack URL. You could also sync your fork, pull to your cloned repo, and navigate to `lectures/10-regression/` and the file is there too.). I've also amended the file to make it smaller so it can be easily pushed to GitHub. *Note: This is bad practice generally, do not store data files on GitHub unless you have a really good reason to do so.*

```
# Create opp atlas object by reading in a CSV off the internet.
opp_atlas <- read_csv("https://raw.githubusercontent.com/big-data-and-economics/big-data-class-materials
```

```
## Rows: 3219 Columns: 8
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): czname
## dbl (7): state, county, cz, kfr_pooled_pooled_p25, poor_share1990, ann_avg_j...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#Quickly renaming fips data to make it easier to merge
fips <- fips_codes %>%
  rename(state_abb=state,
    state=state_code,
    county_name=county,
    county=county_code) %>%
  mutate(across(c(state,county),as.numeric))
```

```
# Look at file to refresh your memory
View(opp_atlas)
```

```
opp_atlas <- opp_atlas %>%
  left_join(fips) %>% # Join together the data
  rename(kfr_p25=kfr_pooled_pooled_p25)  # Rename so you have to type less later.
```

```
## Joining with `by = join_by(state, county)`
```

Regression basics

The `lm()` function

R's base workhorse command for running regression models is the built-in `lm()` function. The "**lm**" stands for "**l**inear **m**odels" and the syntax is very intuitive. The syntax is straight-forward to run a simple regression, but it can be a bit cumbersome to run more complex models.

```
lm(y ~ x1 + x2 + x3 + ..., data = df)
```

Where `y` is the dependent variable, `x1`, `x2`, `x3`, etc. are the independent variables, and `df` is the data frame that contains these variables. You'll note that the `lm()` call includes a reference to the data source. We covered this in our earlier lecture on R language basics and object-orientated programming, but the reason is that many objects (e.g. data frames) can exist in your R environment at the same time. So we need to be specific about where our regression variables are coming from — even if `opp_atlas` is the only data frame in our global environment at the time.

Here's an example using the Opportunity Atlas data. We'll run a simple bivariate regression of the 1990 poverty rate on income mobility for children in the 25th percentile.

```
ols_lm = lm(kfr_p25 ~ poor_share1990, data = opp_atlas)
ols_lm
```

```
##
## Call:
## lm(formula = kfr_p25 ~ poor_share1990, data = opp_atlas)
##
## Coefficients:
##    (Intercept)  poor_share1990
##       5.601828       -0.004268
```

You might immediately notice that the coefficient seems wrong. A higher share that are poor in 1990 is associated with greater income mobility? We'll get there in a second.

First, let's note some limitations of `lm()`. If you want to throw in fixed effects, non-standard errors, or any other fancy stuff, you'll need to bring in other packages. So why not just teach you the same syntax, but with a more flexible package? That sounds better. Let's do that.

The fixest package and `feols()` function

The **fixest** package (link) is a powerful and flexible package for running linear models in R created by Laurent Bergé. It's particularly well-suited for high-dimensional fixed effects models, but it also has a bunch of other neat features. It is **extremely** fast too because it leverages some more complicated econometric theorems to speed up computation, reduce memory usage, and parallelize the computation. (We'll do parallelization later.)

fixest has many different linear models built in, but `feols()` is what is used for ordinary least squares (OLS) regression. The syntax is very similar to `lm()`, but with a few extra bells and whistles.

```
#library(fixest) # already loaded
ols_fixest = feols(kfr_p25 ~ poor_share1990, data = opp_atlas)
ols_fixest
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 3,219
## Standard-errors: IID
##                 Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)     5.601828   0.401529 13.95124 < 2.2e-16 ***
## poor_share1990 -0.004268   0.015307 -0.27882    0.7804
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 21.9   Adj. R2: -2.867e-4
```

The output of `ols_lm` and `ols_fixest` differs a little, but check – are the coefficients and standard errors the same? Always check if a new tool gives the same results as the simpler tool before diving into further.

What's in these objects? First, let's talk about the object we have. The resulting **fixest** (and **lm**) object is pretty terse, but that's only because it buries most of its valuable information — of which there is a lot — within an internal list structure. If you're in RStudio, you can inspect this structure by typing `View(ols_fixest)` or simply clicking on the "ols_fixest" object in your environment pane. Doing so will prompt an interactive panel to pop up for you to play around with. That approach won't work for this knitted R Markdown document, however, so I'll use the `listviewer::jsonedit()` function (used in the APIs lecture) instead.

```
# View(ols_fixest) ## Run this instead if you're in a live session
listviewer::jsonedit(ols_fixest, mode="view") ## Better for R Markdown
```

As we can see, this `ols_fixest` object has a bunch of important slots… containing everything from the regression coefficients, to vectors of the residuals and fitted (i.e. predicted) values, to the rank of the design matrix, to the input data, etc. etc. To summarise the key pieces of information, we can use the — *wait for it* — generic `summary()` function. This will look pretty similar to the default regression output from Stata that many of you will be used to.

```
summary(ols_fixest)
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 3,219
## Standard-errors: IID
##                 Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)     5.601828   0.401529 13.95124 < 2.2e-16 ***
## poor_share1990 -0.004268   0.015307 -0.27882    0.7804
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 21.9   Adj. R2: -2.867e-4
```

We can then dig down further by extracting a summary of the regression coefficients:

```
summary(ols_fixest)$coefficients
```

```
##    (Intercept) poor_share1990
##    5.601827509   -0.004267784
```

Get "tidy" regression coefficients with the `broom` package

While it's easy to extract regression coefficients via the `summary()` function, in practice I often use the **broom** package (link) to do so. **broom** has a bunch of neat features to convert statistical objects like regressions into "tidy" data frames. This is especially useful because regression output is so often used as an input to something else, e.g. a plot of coefficients or marginal effects. Here, I'll use `broom::tidy(..., conf.int = TRUE)` to coerce the `ols_fixest` regression object into a tidy data frame of coefficient values and key statistics.

```
# library(broom) ## Already loaded
tidy(ols_fixest, conf.int = TRUE)
```

```
## # A tibble: 2 x 7
##   term           estimate std.error statistic  p.value conf.low conf.high
##   <chr>             <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
## 1 (Intercept)     5.60       0.402     14.0   5.40e-43   4.81      6.39
## 2 poor_share1990 -0.00427    0.0153    -0.279 7.80e- 1  -0.0343    0.0257
```

Again, I could now pipe this tidied coefficients data frame to a **ggplot2** call, using saying `geom_pointrange()` to plot the error bars. Feel free to practice doing this yourself now, but we'll get to some explicit examples further below.

broom has a couple of other useful functions too. For example, `broom::glance()` summarises the model "meta" data (R2, AIC, etc.) in a data frame.

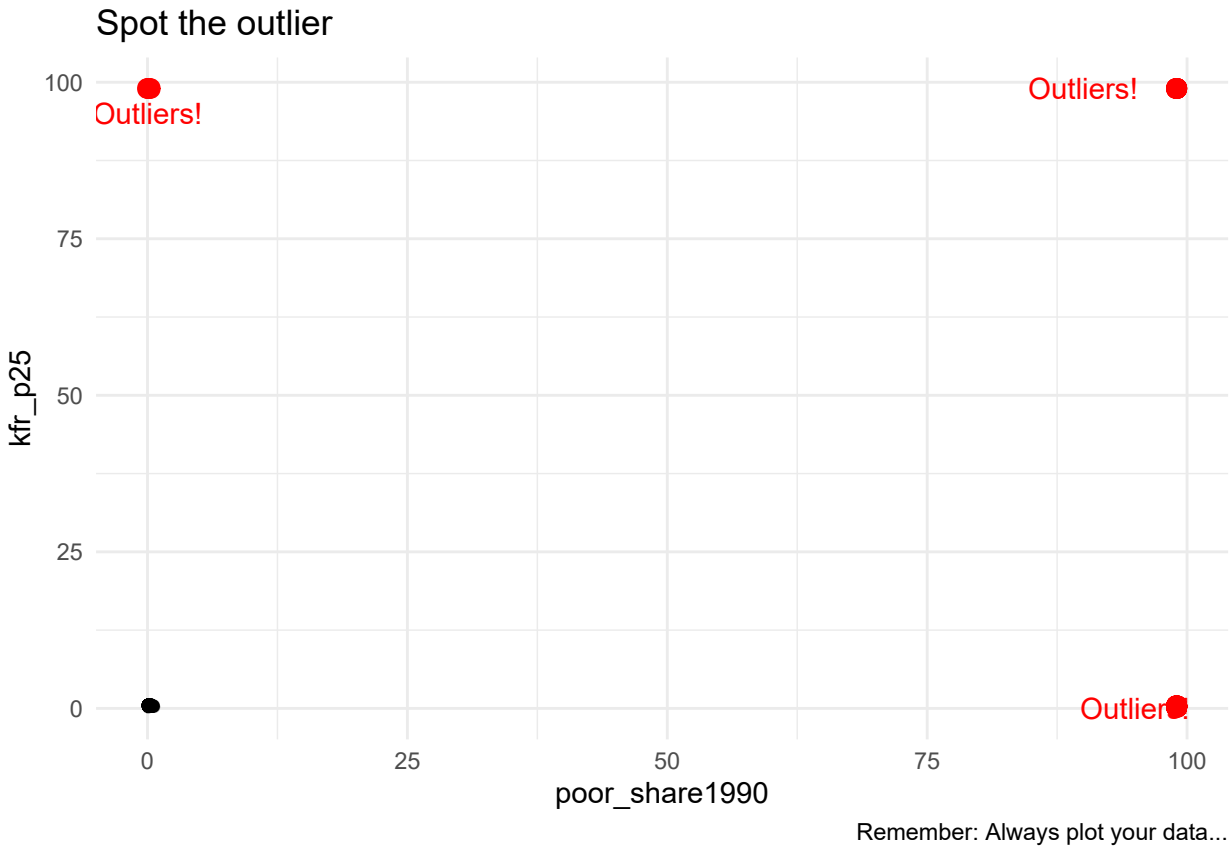
```
glance(ols_fixest)
```

```
## # A tibble: 1 x 9
##   r.squared adj.r.squared within.r.squared pseudo.r.squared sigma  nobs    AIC
##       <dbl>         <dbl>            <dbl>            <dbl> <dbl> <int>  <dbl>
## 1 0.0000242     -0.000287               NA               NA  21.9  3219 29017.
## # i 2 more variables: BIC <dbl>, logLik <dbl>
```

By the way, if you're wondering how to export regression results to other formats (e.g. LaTeX tables), don't worry: We'll get to that at the end of the lecture.

Regressing on subsetted data

Our simple model isn't particularly good; the R2 is only 0. Let's check for outliers by plotting the data.



It looks like NAs were replaced with 99 by someone… ahem… someone who wants to remind you to plot your data and check for quirks like this. Maybe we should exclude outliers from our regression? You can do this in two ways:

1. Subset (`filter()`) the original data frame directly in the `feols()` call.
2. Create a new data frame and then regress

1) Subset directly in the `lm()` call Running a regression directly on a subsetted data frame is equally easy.

```
ols_fixest_sub = feols(kfr_p25 ~ poor_share1990, data = opp_atlas %>% filter(kfr_p25!=99 & poor_share19
summary(ols_fixest_sub)
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,836
## Standard-errors: IID
##                 Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)     0.462206   0.002605 177.4567 < 2.2e-16 ***
## poor_share1990 -0.196645   0.014276 -13.7744 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.060279   Adj. R2: 0.062417
```

2) Create a new data frame Recall that we can keep multiple objects in memory in R. So we can easily create a new data frame that excludes Jabba using, say, **dplyr** (lecture) or **data.table** (lecture. For these lecture notes, I'll stick with **dplyr** commands. But it would take just a little elbow grease (with help from ChatGPT or CoPilot) to switch to **data.table** if you prefer.

```
opp_atlas =
  opp_atlas %>%
```

```
  filter(kfr_p25!=99 & poor_share1990!=99)

ols_fixest_filter = feols(kfr_p25 ~ poor_share1990, data = opp_atlas)
summary(ols_fixest_filter)
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,836
## Standard-errors: IID
##                 Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)     0.462206   0.002605 177.4567 < 2.2e-16 ***
## poor_share1990 -0.196645   0.014276 -13.7744 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.060279   Adj. R2: 0.062417
```

The overall model fit is much improved by the exclusion of this outlier, with R2 increasing to 0.063. Still, we should be cautious about throwing out data. Another approach is to handle or account for outliers with statistical methods. Which provides a nice segue to nonstandard errors.

Nonstandard errors

Dealing with statistical irregularities (heteroskedasticity, clustering, etc.) is a fact of life for empirical researchers. However, it says something about the economics profession that a random stranger could walk uninvited into a live seminar and ask, "How did you cluster your standard errors?", and it would likely draw approving nods from audience members.

The good news is that there are *lots* of ways to get nonstandard errors in R. For many years, these have been based on the excellent **sandwich** package (link). However, here I'll demonstrate using the **estimatr** package (link), which is both fast and provides convenient aliases for the standard regression functions. Some examples follow below.

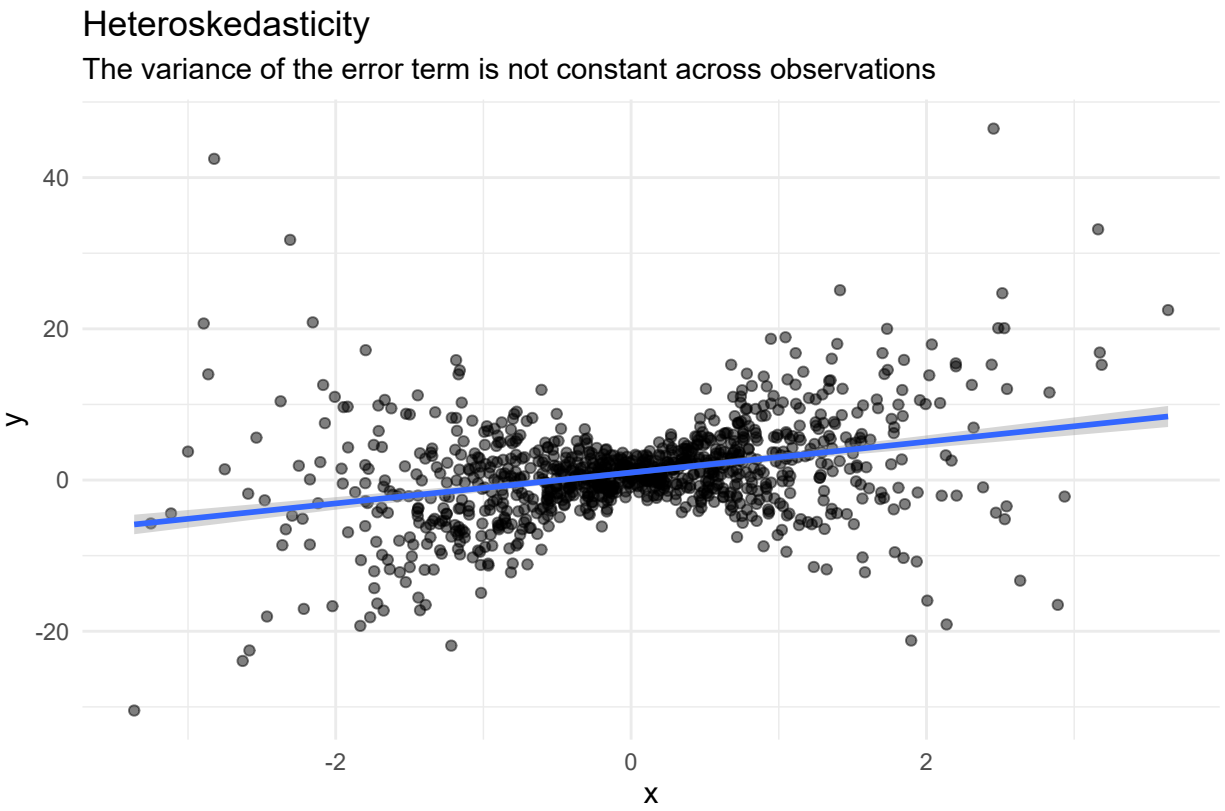
Robust standard errors

One of the primary reasons that you might want to use robust standard errors is to account for heteroskedasticity. What is heteroskedasticity, well it is when the variance of the error term is not constant across observations. This is a problem because it violates one of the key assumptions of OLS regression, namely that the error term is homoskedastic (i.e. constant variance).[1] I present an example below with fake data (cause it is easier than forcing it out of real data.)

```
# Create an example of heteroskedasticity

# This creates a simple regression, but with variance that increases with x
hetero_df <- tibble(
  x = rnorm(1000),
  y = 1 + 2 * x + rnorm(1000, sd = 1 + 5 * abs(x))
)

# Plot the data
hetero_df %>%
  ggplot(aes(x = x, y = y)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", se = TRUE) +
  labs(
    title = "Heteroskedasticity",
    subtitle = "The variance of the error term is not constant across observations",
    caption = "Source: Gen AI (GitHub CoPilot) helped me write this example, but I still used my brain
  )
```

[1] See Causal Inference: The Mixtape for more details.

```
## `geom_smooth()` using formula = 'y ~ x'
```



Source: Gen AI (GitHub CoPilot) helped me write this example, but I still used my brain too.

There are many ways to deal with heteroskedasticity, but one of the most common is to use robust standard errors or "HC1" heteroskedasticity-consistent standard errors. **fixest** and `feols()` has an argument, `vcov` (for variance-covariance matrix) that you can set as "HC1" or "hetero" to generate standard errors. I'll illustrate below!

```
ols_robust = feols(kfr_p25 ~ poor_share1990, data = opp_atlas, vcov = "HC1")
ols_robust = feols(kfr_p25 ~ poor_share1990, data = opp_atlas, vcov = "hetero")
# tidy(ols1_robust, conf.int = TRUE) ## Could tidy too
ols_robust
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,836
## Standard-errors: Heteroskedasticity-robust
##                 Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)     0.462206   0.002361 195.7588 < 2.2e-16 ***
## poor_share1990 -0.196645   0.012822 -15.3366 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.060279   Adj. R2: 0.062417
```

If you use the package **estimatr**, there is a function `lm_robust()`, which defaults to using Eicker-Huber-White robust standard errors, commonly referred to as "HC2" standard errors. You can easily specify alternate methods using the `se_type =` and `vcov` argument.[2] For example, you can specify Stata robust standard errors (`reg y x, robust`) if you want to replicate code or results from that language. (See here for more details on why this isn't the default and why Stata's robust standard errors differ from those in R and Python. tl;dr: A few years ago several people realized Stata was reporting different SEs than they expected.) See Grant's notes for more advanced treatment of robustness.

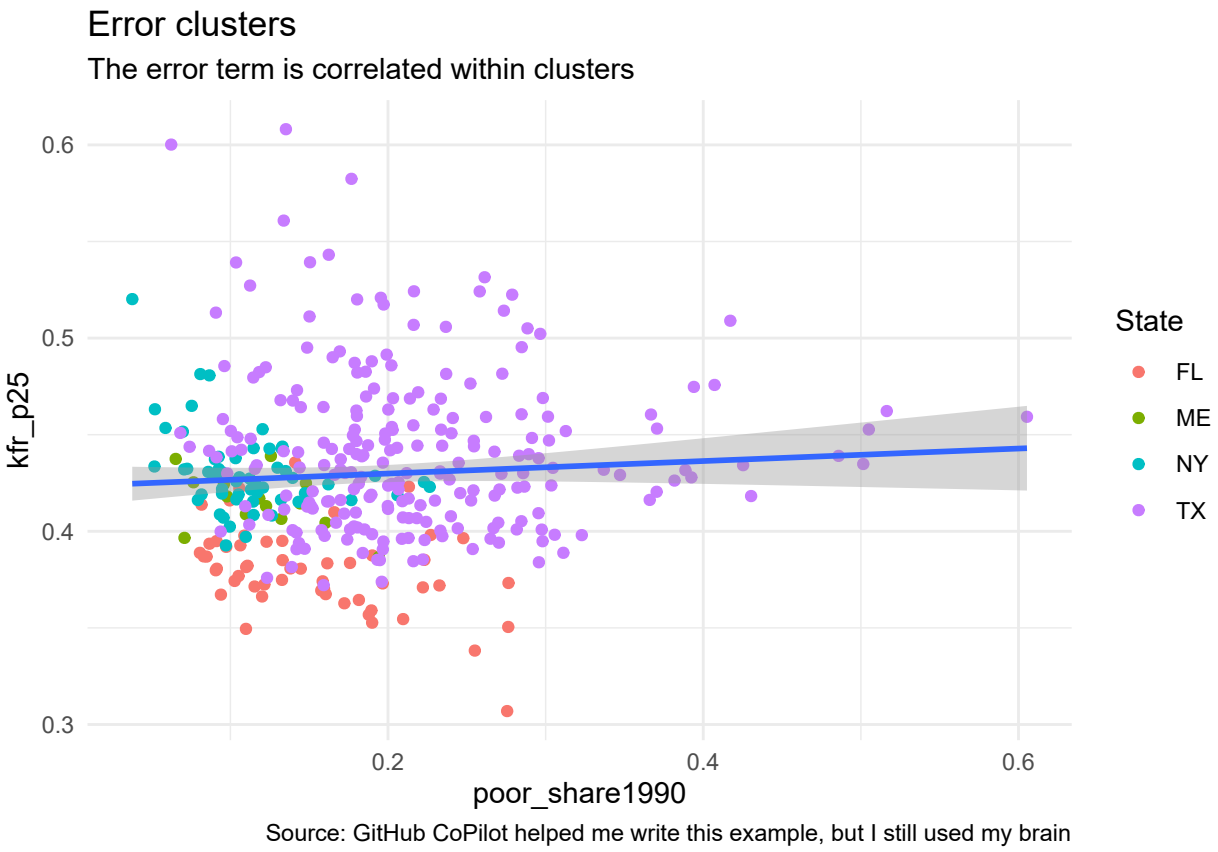
[2] See the package documentation for a full list of options.

Clustered standard errors

Another way that standard errors can violate homoskedasticity is for the error to be "clustered" by groups in the data. A classic example is students in the same classroom will all have the same teacher and so their learning outcomes will be correlated. For example, all of you have the same teacher (me), so you will all learn the same coding habits. This is a problem because it violates the assumption that the error term is independent across observations. Here's an example with the Opportunity Atlas data.

```
# Let's look at just a few states, so we can easily see clusters
filter(opp_atlas, state_abb %in% c('FL','NY','TX','ME')) %>%
  ggplot(aes(x=poor_share1990,y=kfr_p25)) +
  geom_point(aes(col=state_abb)) +
  geom_smooth(method='lm') +
  labs(
    col = 'State',
    title = "Error clusters",
    subtitle = "The error term is correlated within clusters",
    caption = "Source: GitHub CoPilot helped me write this example, but I still used my brain"
  )
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Clustered standard errors is an issue that most commonly affects panel data. As such, I'm going to hold off discussing clustering until we get to the panel data section below. But here's a quick example of clustering with `fixest::feols`:

```
fixest_cluster <- feols(kfr_p25 ~ poor_share1990, data = opp_atlas, cluster = ~state)
```

Manipulating variables

Standardize variables Regression coefficients can be tricky to understand! The units may be strange or the coefficients may be hard to interpret. One way to make coefficients easier to interpret is to manipulate the variables to be more informative. For example, you might want to standardize the variables to return a correlation coefficient.

The statistical equation for a correlation coefficient is:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\hat{\sigma}_x^2, \hat{\sigma}_y^2}$$

Where x_i and y_i are the individual observations, and $\bar{x}$ and $\bar{y}$ are the means of the variables. The $\hat{\sigma}_x^2$ and $\hat{\sigma}_y^2$ are the sample variances of the variables. Let's change the variables `kfr_p25` and `poor_share1990` to be standardized and then run a regression.

```
opp_atlas <- opp_atlas %>%
  mutate(
    kfr_p25_std = (kfr_p25-mean(kfr_p25))/sd(kfr_p25),
    poor_share1990_std = (poor_share1990-mean(poor_share1990))/sd(poor_share1990)
  )
```

Take a look at the variables to see if they are standardized.

```
fixest_corr <- feols(kfr_p25_std ~ poor_share1990_std, data = opp_atlas)
summary(fixest_corr)
```

```
## OLS estimation, Dep. Var.: kfr_p25_std
## Observations: 2,836
## Standard-errors: IID
##                          Estimate Std. Error       t value  Pr(>|t|)
## (Intercept)         1.850000e-16   0.018182  1.020000e-14         1
## poor_share1990_std -2.504958e-01   0.018186 -1.377438e+01 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.967947   Adj. R2: 0.062417
```

Is that a correlation coefficient? Maybe? That'd be neat! How can we find out? Well use the `cor()` function and see if the coefficients are the same.

```
cor(opp_atlas$kfr_p25, opp_atlas$poor_share1990)
```

```
## [1] -0.2504958
```

They are! This is a handy trick to standardized two radically different variables to tell you about the overall correlation when you're less interested in the individual coefficients. This regression trick also returns standard errors which are really nice to have.

Convert variables to logs Another useful trick is to log variables! You can do this directly or use the `log()` function.

```
tidy(feols(log(kfr_p25) ~ log(poor_share1990), data = opp_atlas))
```

```
## # A tibble: 2 x 5
##   term                estimate std.error statistic  p.value
##   <chr>                  <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)          -0.990    0.0105      -93.9 0
## 2 log(poor_share1990)  -0.0705   0.00533     -13.2 6.88e-39
```

Dummy variables and interaction terms

For the next section, we'll need to create a dummy variable. We'll create two: one for whether a county is in the South and one for having positive job growth.

```
# Get the regions and do minor cleaning
opp_atlas <- mutate(opp_atlas,
  in_south = ifelse(state_abb %in% c("AL", "AR",
    "FL", "GA", "KY", "LA", "MS", "NC", "OK",
     "SC", "TN", "TX", "VA", "WV"), "South", "North"))
```

Dummy variables as *factors*

Dummy variables are a core component of many regression models. However, these can be a pain to create in some statistical languages, since you first have to tabulate a whole new matrix of binary variables and then append it to the original data frame. In contrast, R has a very convenient framework for creating and evaluating dummy variables in a regression: Simply specify the variable of interest as a factor.[3]

Here's an example where we explicitly tell R that "in_south" is a factor. Since I don't plan on reusing this model, I'm just going to print the results to screen rather than saving it to my global environment.

```
summary(feols(kfr_p25 ~ poor_share1990 + as.factor(in_south), data = opp_atlas))
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,836
## Standard-errors: IID
##                            Estimate Std. Error   t value   Pr(>|t|)
## (Intercept)                0.464157   0.002336 198.69988  < 2.2e-16 ***
## poor_share1990            -0.050497   0.013948  -3.62032 0.00029939 ***
## as.factor(in_south)South -0.058671   0.002227 -26.34129  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.054025   Adj. R2: 0.246608
```

Okay, I should tell you that I'm actually making things more complicated than they need to be with the heavy-handed emphasis on factors. R is "friendly" and tries to help whenever it thinks you have misspecified a function or variable. While this is something to be aware of, normally It Just Works™. A case in point is that we don't actually *need* to specify a string (i.e. character) variable as a factor in a regression. R will automatically do this for you regardless, since it's the only sensible way to include string variables in a regression.

```
## Use the non-factored version of "in_south" instead; R knows it must be ordered
## for it to be included as a regression variable
summary(feols(kfr_p25 ~ poor_share1990 + in_south, data = opp_atlas))
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,836
## Standard-errors: IID
##                 Estimate Std. Error   t value   Pr(>|t|)
## (Intercept)     0.464157   0.002336 198.69988  < 2.2e-16 ***
## poor_share1990 -0.050497   0.013948  -3.62032 0.00029939 ***
## in_southSouth  -0.058671   0.002227 -26.34129  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.054025   Adj. R2: 0.246608
```

What happens if I use job growth quartile as a dummy variable? First, let's check the values of this value. It is uneven because we removed missing values before (a bit haphazardly, I might add.)

[3] Factors are variables that have distinct qualitative levels, e.g. "male", "female", "non-binary", etc.

```r
table(opp_atlas$job_growth_quartile)
```

```
##
##   1   2   3   4
## 675 725 718 712
```

```r
summary(feols(kfr_p25 ~ poor_share1990 + job_growth_quartile, data = opp_atlas))
```

```
## NOTE: 6 observations removed because of NA values (RHS: 6).
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,830
## Standard-errors: IID
##                      Estimate Std. Error   t value  Pr(>|t|)
## (Intercept)          0.453041   0.004139 109.45594 < 2.2e-16 ***
## poor_share1990      -0.188190   0.014663 -12.83415 < 2.2e-16 ***
## job_growth_quartile  0.003158   0.001048   3.01317 0.0026082 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.059972   Adj. R2: 0.066759
```

We can fix this by telling R that job_growth_quartile is a factor.

```r
summary(feols(kfr_p25 ~ poor_share1990 + as.factor(job_growth_quartile), data = opp_atlas))
```

```
## NOTE: 6 observations removed because of NA values (RHS: 6).
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,830
## Standard-errors: IID
##                                  Estimate Std. Error   t value  Pr(>|t|)
## (Intercept)                      0.456668   0.003782 120.73991 < 2.2e-16 ***
## poor_share1990                  -0.189521   0.014818 -12.79030 < 2.2e-16 ***
## as.factor(job_growth_quartile)2  0.003675   0.003262   1.12688 0.2598904
## as.factor(job_growth_quartile)3  0.003922   0.003327   1.17886 0.2385515
## as.factor(job_growth_quartile)4  0.010368   0.003315   3.12730 0.0017821 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.059959   Adj. R2: 0.066527
```

Interaction effects

Like dummy variables, R provides a convenient syntax for specifying interaction terms directly in the regression model without having to create them manually beforehand.[4] You can use any of the following expansion operators:

- `x1:x2` "crosses" the variables (equivalent to including only the x1 × x2 interaction term)
- `x1/x2` "nests" the second variable within the first (equivalent to `x1 + x1:x2`; more on this later)
- `x1*x2` includes all parent and interaction terms (equivalent to `x1 + x2 + x1:x2`)

As a rule of thumb, if not always, it is generally advisable to include all of the parent terms alongside their interactions. This makes the `*` option a good default.

For example, we might wonder whether the relationship between a location's income mobility for children in the 25th percentile and its 1990 poverty rate differs by region. That is, we want to run a regression of the form,

[4] Although there are very good reasons that you might want to modify your parent variables before doing so (e.g. centering them). As it happens, Grant McDermott has strong feelings that interaction effects are most widely misunderstood and misapplied concept in econometrics. However, that's a topic for another day.

$$KFR_{P25} = \beta_0 + \beta_1 D_{South} + \beta_2 + \beta_3 D_{South} \times 1990PovertyShare$$

To implement this in R, we simply run the following,

```
ols_ie = feols(kfr_p25 ~ in_south * poor_share1990, data = opp_atlas)
summary(ols_ie)
```

```
## OLS estimation, Dep. Var.: kfr_p25
## Observations: 2,836
## Standard-errors: IID
##                              Estimate Std. Error    t value  Pr(>|t|)
## (Intercept)                  0.462703   0.003309 139.841639 < 2.2e-16 ***
## in_southSouth               -0.055816   0.005113 -10.916960 < 2.2e-16 ***
## poor_share1990              -0.039824   0.022149  -1.798030  0.072279 .
## in_southSouth:poor_share1990 -0.017690   0.028515  -0.620363  0.535069
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.054021   Adj. R2: 0.246445
```

Presentation

Tables

Regression tables There are loads of different options here.

Fixest actually provides its own table function, `etable()` (for "estimation table"), which is a bit like Stata's `esttab` or `outreg` command. It's a bit more flexible than the default `summary()` function, but it only works with **fixest** list objects.

```
etable(ols_fixest,ols_fixest_filter,ols_robust,fixest_cluster,fixest_corr,ols_ie)
```

```
##                                    ols_fixest   ols_fixest_filter
## Dependent Var.:                       kfr_p25             kfr_p25
##
## Constant                        5.602*** (0.4015)  0.4622*** (0.0026)
## poor_share1990                   -0.0043 (0.0153) -0.1966*** (0.0143)
## poor_share1990_std
## in_southSouth
## in_southSouth x poor_share1990
## ------------------------------ ----------------- -------------------
## S.E. type                                     IID                 IID
## Observations                                3,219               2,836
## R2                                        2.42e-5             0.06275
## Adj. R2                                  -0.00029             0.06242
##
##                                     ols_robust      fixest_cluster
## Dependent Var.:                       kfr_p25             kfr_p25
##
## Constant                         0.4622*** (0.0024)  0.4622*** (0.0106)
## poor_share1990                  -0.1966*** (0.0128) -0.1966*** (0.0471)
## poor_share1990_std
## in_southSouth
## in_southSouth x poor_share1990
## ------------------------------ ------------------- -------------------
## S.E. type                      Heteroskedast.-rob.           by: state
## Observations                                 2,836               2,836
## R2                                         0.06275             0.06275
```

```
## Adj. R2                                    0.06242             0.06242
##
##                                       fixest_corr              ols_ie
## Dependent Var.:                       kfr_p25_std             kfr_p25
##
## Constant                          1.85e-16 (0.0182)  0.4627*** (0.0033)
## poor_share1990                                        -0.0398. (0.0221)
## poor_share1990_std             -0.2505*** (0.0182)
## in_southSouth                                        -0.0558*** (0.0051)
## in_southSouth x poor_share1990                          -0.0177 (0.0285)
## ---------------------------- ------------------- --------------------
## S.E. type                                     IID                 IID
## Observations                                2,836               2,836
## R2                                        0.06275             0.24724
## Adj. R2                                   0.06242             0.24644
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Hmm… how can I clean that up a bit? Well `etable()` has a number of options including `dict` where you can assign variable names.

```
dict = c("poor_share1990"="Poverty Rate (1990)",
         'poor_share1990_std'='Std. Poverty Rate (1990)',
         'kfr_p25'='Income Mobility (25th Percentile)',
         'kfr_p25_std'='Std. Income Mobility (25th Percentile)',
         'in_south'='In South')

etable(ols_fixest,ols_fixest_filter,ols_robust,fixest_cluster,fixest_corr, dict = dict)
```

```
##                                              ols_fixest
## Dependent Var.:          Income Mobility (25th Percentile)
##
## Constant                                  5.602*** (0.4015)
## Poverty Rate (1990)                        -0.0043 (0.0153)
## Std. Poverty Rate (1990)
## ------------------------    ------------------------------
## S.E. type                                            IID
## Observations                                       3,219
## R2                                               2.42e-5
## Adj. R2                                         -0.00029
##
##                                         ols_fixest_filter
## Dependent Var.:          Income Mobility (25th Percentile)
##
## Constant                                  0.4622*** (0.0026)
## Poverty Rate (1990)                      -0.1966*** (0.0143)
## Std. Poverty Rate (1990)
## ------------------------    ------------------------------
## S.E. type                                            IID
## Observations                                       2,836
## R2                                               0.06275
## Adj. R2                                          0.06242
##
##                                              ols_robust
## Dependent Var.:          Income Mobility (25th Percentile)
```

```
##
## Constant                                  0.4622*** (0.0024)
## Poverty Rate (1990)                      -0.1966*** (0.0128)
## Std. Poverty Rate (1990)
## ------------------------    ------------------------------
## S.E. type                        Heteroskedasticity-robust
## Observations                                         2,836
## R2                                                 0.06275
## Adj. R2                                            0.06242
##
##                                              fixest_cluster
## Dependent Var.:          Income Mobility (25th Percentile)
##
## Constant                                  0.4622*** (0.0106)
## Poverty Rate (1990)                      -0.1966*** (0.0471)
## Std. Poverty Rate (1990)
## ------------------------    ------------------------------
## S.E. type                                        by: state
## Observations                                         2,836
## R2                                                 0.06275
## Adj. R2                                            0.06242
##
##                                               fixest_corr
## Dependent Var.:          Std. Income Mobility (25th Percentile)
##
## Constant                                    1.85e-16 (0.0182)
## Poverty Rate (1990)
## Std. Poverty Rate (1990)                  -0.2505*** (0.0182)
## ------------------------         ------------------------------
## S.E. type                                                  IID
## Observations                                             2,836
## R2                                                     0.06275
## Adj. R2                                                0.06242
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There are tons of other tools to use too!

As a note, here's what you need to do in Rmarkdown to get it to work in a PDF format. You must specify to output as a latex file, then you can report "asis" and ask `etable()` to output a tex file.

```
etable(ols_fixest,ols_fixest_filter,ols_robust,fixest_cluster,fixest_corr, dict = dict, tex=TRUE)
```

Dependent Variables:	Income Mobility (25th Percentile)				Std. Income Mobility (25th Percentile)
Model:	(1)	(2)	(3)	(4)	(5)
Variables					
Constant	5.602***	0.4622***	0.4622***	0.4622***	1.85×10^{-16}
	(0.4015)	(0.0026)	(0.0024)	(0.0106)	(0.0182)
Poverty Rate (1990)	-0.0043	-0.1966***	-0.1966***	-0.1966***	
	(0.0153)	(0.0143)	(0.0128)	(0.0471)	
Std. Poverty Rate (1990)					-0.2505***
					(0.0182)
Fit statistics					
Observations	3,219	2,836	2,836	2,836	2,836
R^2	2.42×10^{-5}	0.06275	0.06275	0.06275	0.06275
Adjusted R^2	-0.00029	0.06242	0.06242	0.06242	0.06242

*Signif. Codes: ***: 0.01, **: 0.05, *: 0.1*

Alternatively, you can save this file to a tex file and then include it in your LaTeX document using `input()` or `include()`.

```
etable(ols_fixest,ols_fixest_filter,ols_robust,fixest_cluster,fixest_corr, dict = dict, tex=TRUE,file="
```

Other neat tricks with `etable()` is that you can specify a "note" and "label" for the table. Also, you can change standard error calculations for all the models at once.

```
etable(ols_fixest,ols_fixest_filter,ols_robust,fixest_cluster,fixest_corr, dict = dict, notes="This is a
```

```
##                                            ols_fixest
## Dependent Var.:          Income Mobility (25th Percentile)
##
## Constant                                5.602*** (0.4015)
## Poverty Rate (1990)                      -0.0043 (0.0153)
## Std. Poverty Rate (1990)
## ________________________    ______________________________
## S.E. type                                           IID
## Observations                                      3,219
## R2                                              2.42e-5
## Adj. R2                                        -0.00029
##
##                                         ols_fixest_filter
## Dependent Var.:          Income Mobility (25th Percentile)
##
## Constant                               0.4622*** (0.0026)
## Poverty Rate (1990)                   -0.1966*** (0.0143)
## Std. Poverty Rate (1990)
## ________________________    ______________________________
## S.E. type                                           IID
## Observations                                      2,836
## R2                                              0.06275
## Adj. R2                                         0.06242
##
##                                            ols_robust
## Dependent Var.:          Income Mobility (25th Percentile)
##
## Constant                               0.4622*** (0.0026)
## Poverty Rate (1990)                   -0.1966*** (0.0143)
```

```
## Std. Poverty Rate (1990)
## ________________________    ______________________________
## S.E. type                                            IID
## Observations                                       2,836
## R2                                               0.06275
## Adj. R2                                          0.06242
##
##                                           fixest_cluster
## Dependent Var.:          Income Mobility (25th Percentile)
##
## Constant                                  0.4622*** (0.0026)
## Poverty Rate (1990)                      -0.1966*** (0.0143)
## Std. Poverty Rate (1990)
## ________________________    ______________________________
## S.E. type                                            IID
## Observations                                       2,836
## R2                                               0.06275
## Adj. R2                                          0.06242
##
##                                            fixest_corr
## Dependent Var.:          Std. Income Mobility (25th Percentile)
##
## Constant                                    1.85e-16 (0.0182)
## Poverty Rate (1990)
## Std. Poverty Rate (1990)                 -0.2505*** (0.0182)
## ________________________        ______________________________
## S.E. type                                              IID
## Observations                                         2,836
## R2                                                 0.06275
## Adj. R2                                            0.06242
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Other tools Another great tool for creating and exporting regression tables is the **modelsummary** package (link). It is extremely flexible and handles all manner of models and output formats. **modelsummary** also supports automated coefficient plots and data summary tables, which I'll get back to in a moment. The documentation is outstanding and you should read it, but here is a bare-boned example just to demonstrate.

```
# library(modelsummary) ## Already loaded

## Note: msummary() is an alias for modelsummary() add variable names
msummary(list("lm"=ols_lm, "fixest"=ols_fixest,"filter"=ols_fixest_filter,"robust"=ols_robust,"cluster"=
         stars=TRUE, ## Output type
         coef_map = c("poor_share1990"="Poverty Rate (1990)",
          "(Intercept)"="Constant"),
          title="Relationship between Average Income Percentile in 2015 of Children born in 25th percen
```

One nice thing about **modelsummary** is that it plays very well with R Markdown and will automatically coerce your tables to the format that matches your document output: HTML, LaTeX/PDF, RTF, etc. Of course, you can also specify the output type if you aren't using R Markdown and want to export a table for later use. Finally, you can even specify special table formats like *threepartable* for LaTeX and, provided that you have called the necessary packages in your preamble, it will render correctly (see example here).

Table 1: Relationship between Average Income Percentile in 2015 of Children born in 25th percentile and 1990 poverty rate

	lm	fixest	filter	robust	cluster
Poverty Rate (1990)	−0.004	−0.004	−0.197***	−0.197***	−0.197***
	(0.015)	(0.015)	(0.014)	(0.013)	(0.047)
Constant	5.602***	5.602***	0.462***	0.462***	0.462***
	(0.402)	(0.402)	(0.003)	(0.002)	(0.011)
Num.Obs.	3219	3219	2836	2836	2836
R2	0.000	0.000	0.063	0.063	0.063
R2 Adj.	0.000	0.000	0.062	0.062	0.062
AIC	29 018.8	29 016.8	−7879.2	−7879.2	−7879.2
BIC	29 037.1	29 029.0	−7867.3	−7867.3	−7867.3
Log.Lik.	−14 506.412				
RMSE	21.92	21.92	0.06	0.06	0.06
Std.Errors		IID	IID	Heteroskedasticity-robust	by: state

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

	North (N=1581)		South (N=1255)			
	Mean	Std. Dev.	Mean	Std. Dev.	Diff. in Means	Std. Error
kfr_p25	0.5	0.1	0.4	0.0	-0.1	0.0
poor_share1990	0.1	0.1	0.2	0.1	0.1	0.0
ann_avg_job_growth_2004_2013	0.0	0.0	0.0	0.0	0.0	0.0

Summary tables A variety of summary tables — balance, correlation, etc. — can be produced by the companion set of `modelsummary::datasummary*()` functions. Again, you should read the documentation to see all of the options. But here's an example of a very simple balance table using a subset of our "humans" data frame.

```
datasummary_balance(~ in_south,
                    data = opp_atlas %>%
                    select(kfr_p25:ann_avg_job_growth_2004_2013,in_south))
```

Another package that I like a lot in this regard is **vtable** (link). Not only can it be used to construct descriptive labels like you'd find in Stata's "Variables" pane, but it is also very good at producing the type of "out of the box" summary tables that economists like. For example, here's the equivalent version of the above balance table.

```
# library(vtable) ## Already loaded

## An additional argument just for formatting across different output types of
## this .Rmd document
otype = ifelse(knitr::is_latex_output(), 'return', 'kable')

## vtable::st() is an alias for sumtable()
vtable::st(opp_atlas %>%
  select(kfr_p25:ann_avg_job_growth_2004_2013, in_south),
   group = 'in_south',
   out = otype)
```

```
##                       Variable     N    Mean    SD     N    Mean    SD
## 1                     in_south North               South
## 2                      kfr_p25  1581    0.46 0.061  1255     0.4 0.044
```

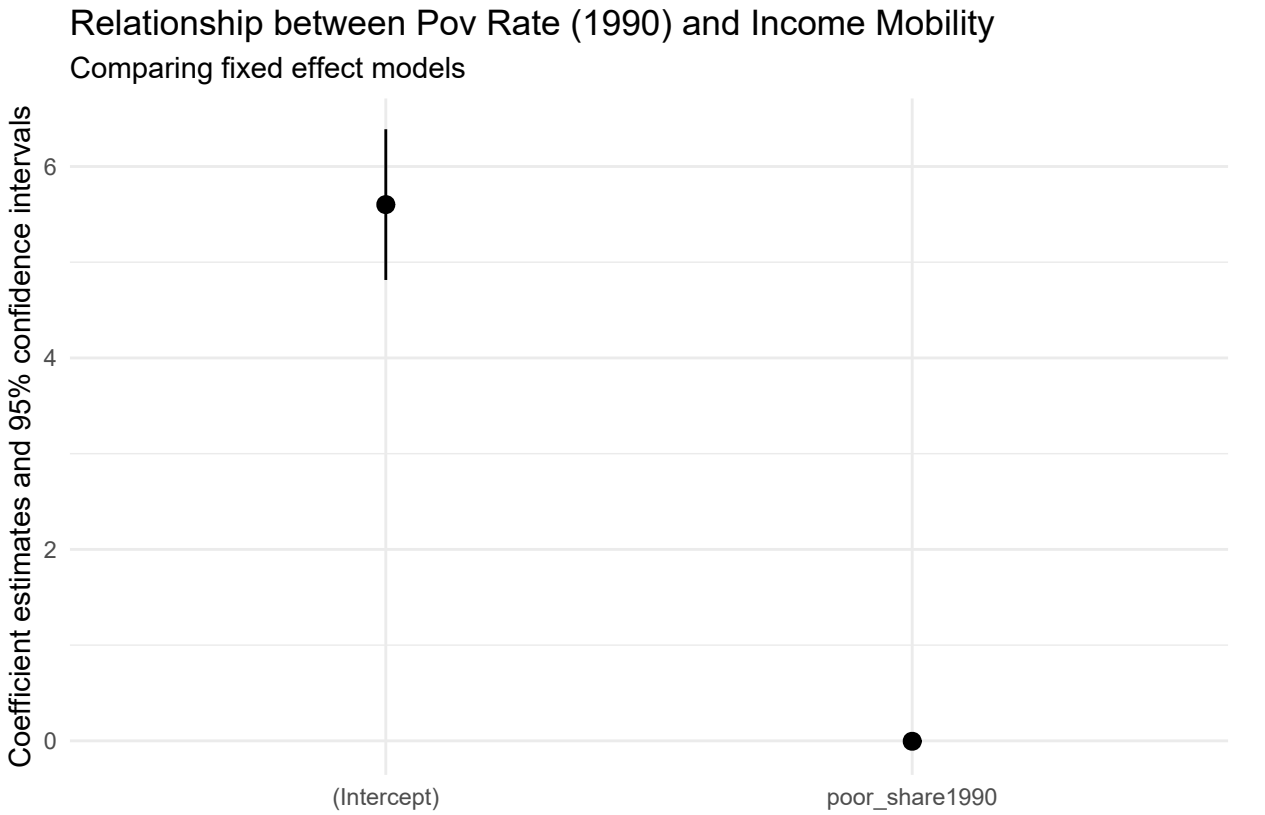
```
## 3               poor_share1990  1581    0.14 0.061  1255     0.2 0.085
## 4 ann_avg_job_growth_2004_2013  1576 -0.0029 0.014  1255 -0.0022 0.016
```

Lastly, Stata users in particular might like the `qsu()` and `descr()` functions from the lightning-fast **collapse** package (link).

Figures

Coefficient plots We've already worked through an example of how to extract and compare model coefficients here. I use this "manual" approach to visualizing coefficient estimates all the time. However, our focus on **modelsummary** in the preceding section provides a nice segue to another one of the package's features: `modelplot()`. Consider the following, which shows both the degree to which `modelplot()` automates everything and the fact that it readily accepts regular **ggplot2** syntax.

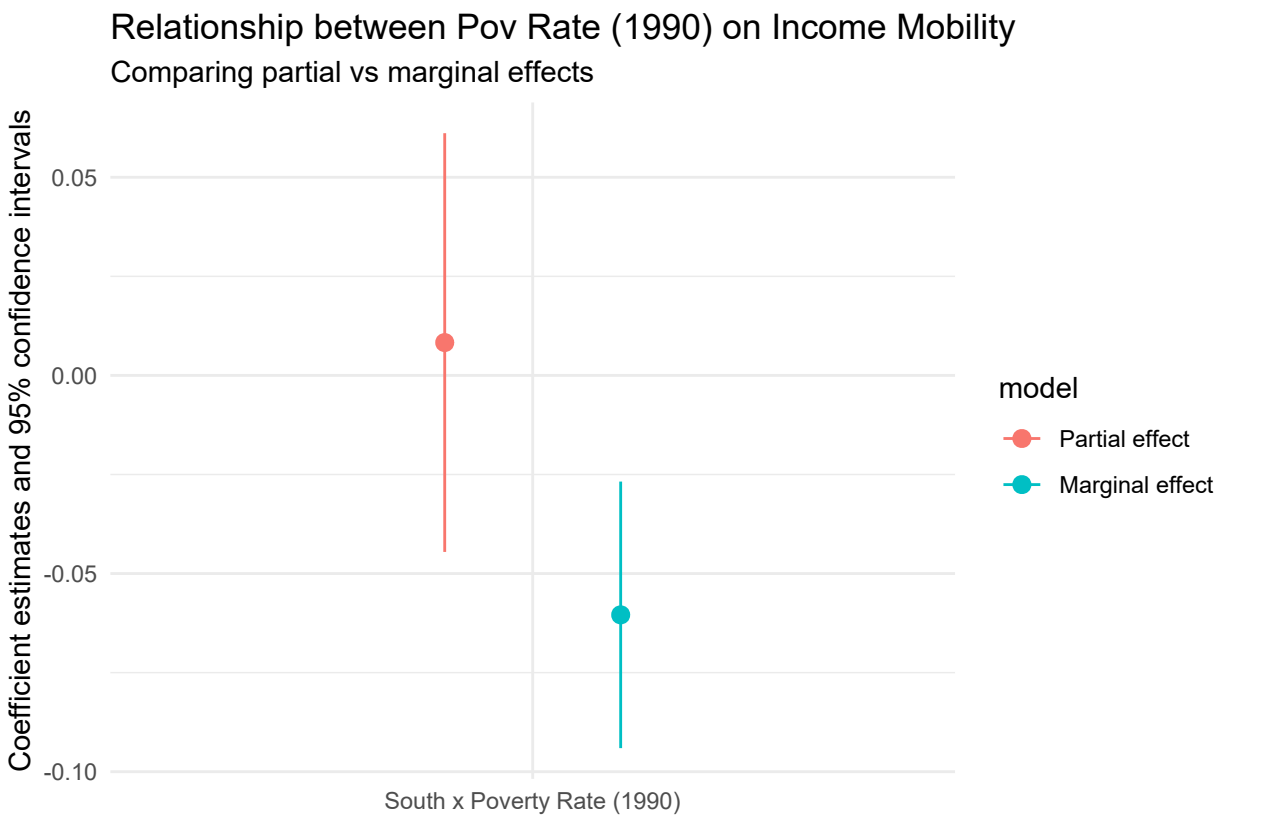
```
# library(modelsummary) ## Already loaded
mods = list('No clustering' = summary(ols_fixest, se = 'standard'))

modelplot(mods) +
  ## You can further modify with normal ggplot2 commands...
  coord_flip() +
  labs(
    title = "Relationship between Pov Rate (1990) and Income Mobility",
    subtitle = "Comparing fixed effect models"
    )
```



Or, here's another example where we compare the (partial) In South × Poverty Share coefficient from our earlier interaction model, with the (full) marginal effect that we obtained later on.

```
ie_mods = list('Partial effect' = ols_ie, 'Marginal effect' = ols_ie_marg2)
```

```r
modelplot(ie_mods, coef_map = c("in_southSouth:poor_share1990" = "South x Poverty Rate (1990)")) +
  coord_flip() +
  labs(
    title = "Relationship between Pov Rate (1990) on Income Mobility",
    subtitle = "Comparing partial vs marginal effects"
    )
```



Further resources

- Ed Rubin has outstanding teaching notes for econometrics with R on his website. This includes both undergrad- and graduate-level courses. Seriously, check them out.
- Several introductory texts are freely available, including *Introduction to Econometrics with R* (Christoph Hanck *et al.*), *Using R for Introductory Econometrics* (Florian Heiss), and *Modern Dive* (Chester Ismay and Albert Kim).
- Tyler Ransom has a nice cheat sheet for common regression tasks and specifications.
- Itamar Caspi has written a neat unofficial appendix to this lecture, *recipes for Dummies*. The title might be a little inscrutable if you haven't heard of the `recipes` package before, but basically it handles "tidy" data preprocessing, which is an especially important topic for machine learning methods. We'll get to that later in course, but check out Itamar's post for a good introduction.
- I promised to provide some links to time series analysis. The good news is that R's support for time series is very, very good. The Time Series Analysis task view on CRAN offers an excellent overview of available packages and their functionality.
- Lastly, for more on visualizing regression output, I highly encourage you to look over Chapter 6 of Kieran Healy's *Data Visualization: A Practical Guide*. Not only will learn how to produce beautiful and effective model visualizations, but you'll also pick up a variety of technical tips.