Data Science for Economists

Lecture 8: Regression analysis in R

Grant R. McDermott

University of Oregon | EC 607

Contents

Today's lecture is about the bread-and-butter tool of applied econometrics and data science: regression analysis. My goal is to give you a whirlwind tour of the key functions and packages. I'm going to assume that you already know all of the necessary theoretical background on causal inference, asymptotics, etc. This lecture will *not* cover any of theoretical concepts or seek to justify a particular statistical model. Indeed, most of the models that we're going to run today are pretty silly. We also won't be able to cover some important topics. For example, I'll only provide the briefest example of a Bayesian regression model and I won't touch times series analysis at all. (Although, I will provide links for further reading at the bottom of this document.) These disclaimers aside, let's proceed…

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**, **rstanarm**
- Already used: **tidyverse**, **hrbrthemes**, **listviewer**

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,rstanarm)
## 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())
```

Panel models

Fixed effects with the fixest package

The simplest (and least efficient) way to include fixed effects in a regression model is, of course, to use dummy variables. However, it isn't very efficient or scalable. What's the point learning all that stuff about the Frisch-Waugh-Lovell, within-group transformations, etc. etc. if we can't use them in our software routines? Again, there are several options to choose from here. For example, many of you are probably familiar with the excellent **lfe** package (link), which offers near-identical functionality to the popular Stata library, **reghdfe** (link). However, for fixed effects models in R, I am going to advocate that you look no further than the **fixest** package (link).

fixest is relatively new on the scene and has quickly become one of my absolute favourite packages. It has an *boatload* of functionality built in to it: support for nonlinear models, high-dimensional fixed effects, multiway clustering, multi-model estimation, LaTeX tables, etc, etc. It is also insanely fast... as in, up to orders of magnitude faster than **lfe** or **reghdfe**. I won't be able to cover all of **fixest**'s features in depth here — see the introductory vignette for a thorough walkthrough — but I hope to least give you a sense of why I am so enthusiastic about it. Let's start off with a simple example before moving on to something slightly more demanding.

Simple FE model The package's main function is `fixest::feols()`, which is used for estimating linear fixed effects models. The syntax is such that you first specify the regression model as per normal, and then list the fixed effect(s) after a `|`. An example may help to illustrate. Let's say that we again want to run our simple regression of mass on height, but this time control for species-level fixed effects.[1]

```
# library(fixest) ## Already loaded

ols_fe = feols(mass ~ height | species, data = starwars) ## Fixed effect(s) go after the "|"
ols_fe
```

```
## OLS estimation, Dep. Var.: mass
## Observations: 58
## Fixed-effects: species: 31
## Standard-errors: Clustered (species)
##        Estimate Std. Error t value  Pr(>|t|)
## height 0.974876   0.044291 22.0105 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 9.69063     Adj. R2: 0.99282
##                 Within R2: 0.662493
```

Note that the resulting model object has automatically clustered the standard errors by the fixed effect variable (i.e. species). We'll explore some more options for adjusting standard errors in **fixest** objects shortly. But to preview things, you can specify the standard errors you want at estimation time... or you can adjust the standard errors for any existing model via `summary.fixest()`. For example, here are two equivalent ways to specify vanilla (iid) standard errors:

[1] Since we specify "species" in the fixed effects slot below, `feols()` will automatically coerce it to a factor variable even though we didn't explicitly tell it to.

Specify SEs at estimation time.

```r
feols(mass ~ height | species,
      data = starwars, se = 'standard')
```

```
## OLS estimation, Dep. Var.: mass
## Observations: 58
## Fixed-effects: species: 31
## Standard-errors: IID
##        Estimate Std. Error t value   Pr(>|t|)
## height 0.974876   0.136463  7.1439 1.3797e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 9.69063     Adj. R2: 0.99282
##                 Within R2: 0.662493
```

Adjust SEs of an existing model (`ols_fe`) on the fly.

```r
summary(ols_fe,
        se = 'standard')
```

```
## OLS estimation, Dep. Var.: mass
## Observations: 58
## Fixed-effects: species: 31
## Standard-errors: IID
##        Estimate Std. Error t value   Pr(>|t|)
## height 0.974876   0.136463  7.1439 1.3797e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0
## RMSE: 9.69063     Adj. R2: 0.99282
##                 Within R2: 0.662493
```

Before continuing, let's quickly save a "tidied" data frame of the coefficients for later use. I'll use iid standard errors again, if only to show you that the `broom::tidy()` method for `fixest` objects also accepts an `se` argument. This basically just provides another convenient way for you to adjust standard errors for your models on the fly.

```r
# coefs_fe = tidy(summary(ols_fe, se = 'standard'), conf.int = TRUE) ## same as below
coefs_fe = tidy(ols_fe, se = 'standard', conf.int = TRUE)
```

High dimensional FEs and multiway clustering As I already mentioned above, **fixest** supports (arbitrarily) high-dimensional fixed effects and (up to fourway) multiway clustering. To see this in action, let's add "homeworld" as an additional fixed effect to the model.

```r
## We now have two fixed effects: species and homeworld
ols_hdfe = feols(mass ~ height | species + homeworld, data = starwars)
ols_hdfe
```

```
## OLS estimation, Dep. Var.: mass
## Observations: 55
## Fixed-effects: species: 30,  homeworld: 38
## Standard-errors: Clustered (species)
##        Estimate Std. Error t value Pr(>|t|)
## height 0.755844   0.332888 2.27057  0.03078 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 7.45791     Adj. R2: 1.00768
##                 Within R2: 0.487231
```

Easy enough, but the standard errors of the above model are automatically clustered by species, i.e. the first fixed effect variable. Let's go a step further and cluster by both "species" and "homeworld". [2] **fixest** provides several ways for us to do this — via the `se` or `cluster` arguments — and, again, you can specify your clustering strategy at estimation time, or adjust the standard errors of an existing model on-the-fly. I'll (re)assign the model to the same `ols_hdfe` object, but you could, of course, create a new object if you so wished.

```r
## Cluster by both species and homeworld

## These next few lines all do the same thing. Pick your favourite!

## Specify desired SEs at estimation time...
# ols_hdfe = feols(mass ~ height | species + homeworld, se = 'twoway', data = starwars)
# ols_hdfe = feols(mass ~ height | species + homeworld, cluster = c('species', 'homeworld'), data = sta
# ols_hdfe = feols(mass ~ height | species + homeworld, cluster = ~ species + homeworld, data = starwar
#
```

[2] I make no claims to this is a particularly good or sensible clustering strategy, but just go with it.

```r
##... or, adjust the SEs of an existing model on the fly
# ols_hdfe = summary(ols_hdfe, se = 'twoway')
# ols_hdfe = summary(ols_hdfe, cluster = c('species', 'homeworld'))
ols_hdfe = summary(ols_hdfe, cluster = ~ species + homeworld) ## I'll go with this one

ols_hdfe
```

```
## OLS estimation, Dep. Var.: mass
## Observations: 55
## Fixed-effects: species: 30,  homeworld: 38
## Standard-errors: Clustered (species & homeworld)
##        Estimate Std. Error t value   Pr(>|t|)
## height 0.755844   0.116416 6.49263 4.1625e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 7.45791     Adj. R2: 1.00768
##                 Within R2: 0.487231
```

Comparing our model coefficients We'll get to model presentation at the very end of the lecture. For now, I want to quickly flag that **fixest** provides some really nice, built-in functions for comparing models. For example, you can get regression tables with `fixest::etable()`.

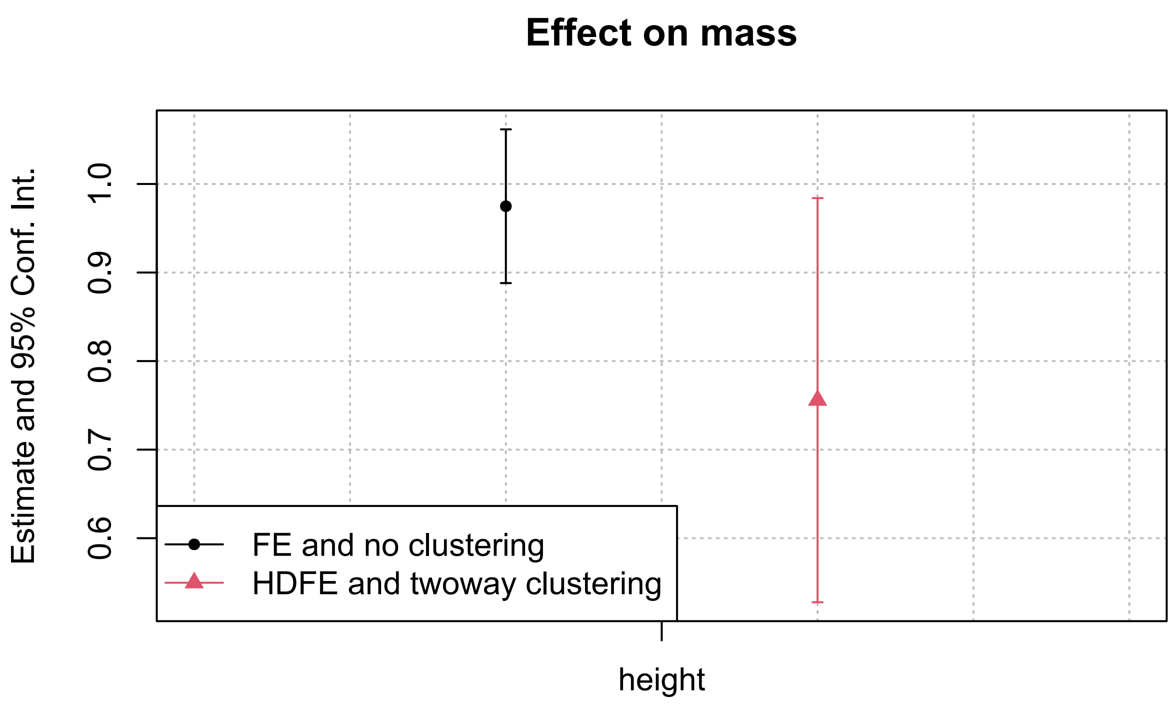
```r
etable(ols_fe, ols_hdfe)
```

```
##                              ols_fe           ols_hdfe
## Dependent Var.:                mass               mass
##
## height          0.9749*** (0.0443) 0.7558*** (0.1164)
## Fixed-Effects:  ------------------ ------------------
## species                        Yes                Yes
## homeworld                       No                Yes
## _______________ __________________ __________________
## S.E.: Clustered        by: species  by: spec. & home.
## Observations                    58                 55
## R2                         0.99672            0.99815
## Within R2                  0.66249            0.48723
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Similarly, the `fixest::coefplot()` function for plotting estimation results:

```r
coefplot(list(ols_fe, ols_hdfe))

## Add legend (optional)
legend("bottomleft", col = 1:2, lwd = 1, pch = c(20, 17),
       legend = c("FE and no clustering", "HDFE and twoway clustering"))
```

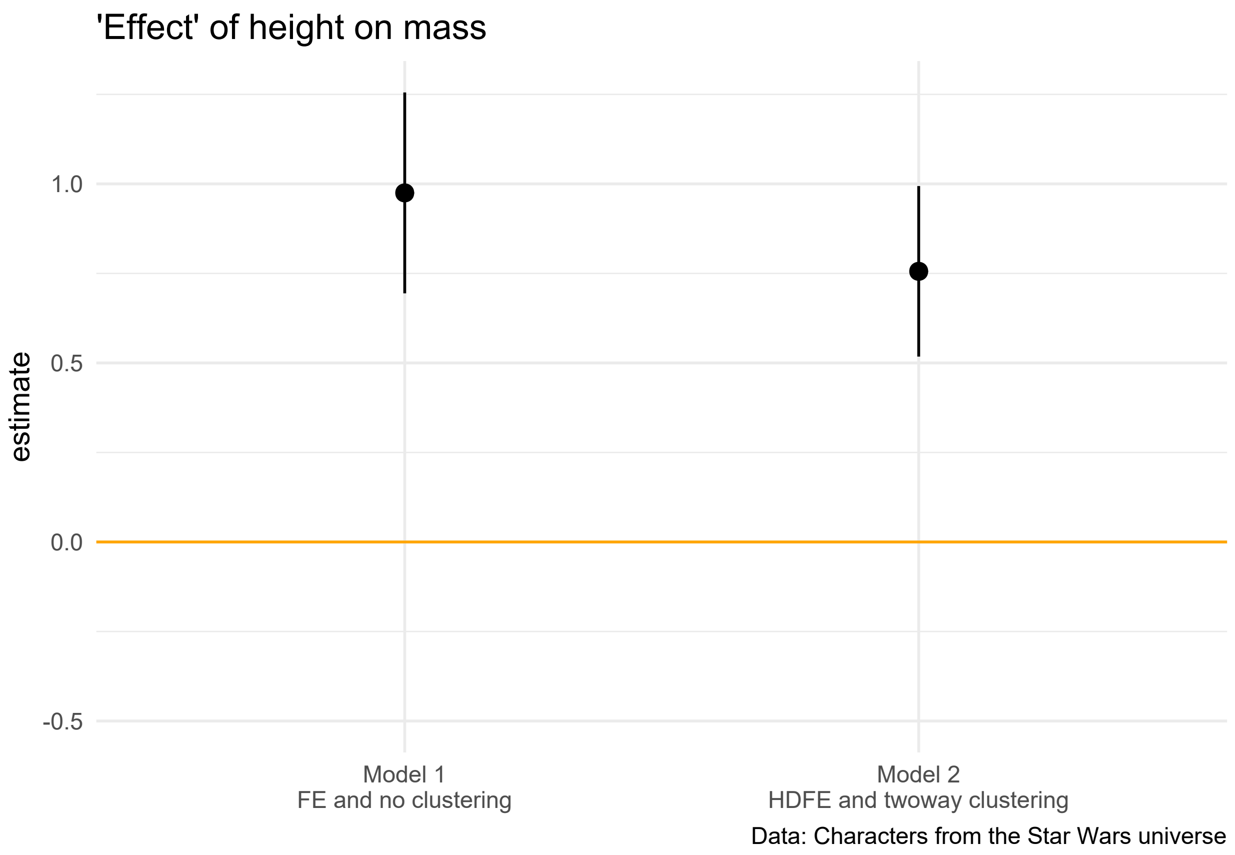


`coefplot()` is especially useful for tracing the evolution of treatment effects over time, as in a difference-in-differences setup (see Examples). However, I realise some people may find it a bit off-putting that it produces base R plots, rather than a **ggplot2** object. We'll get to an automated **ggplot2** coefficient plot solution further below with `modelsummary::modelplot()`. Nevertheless, let me close this out this section by demonstrating the relative ease with which you can do this "manually". Consider the below example, which leverages the fact that we have saved (or can save) regression models as data frames with `broom::tidy()`. As I suggested earlier, this makes it simple to construct our own bespoke coefficient plots.

```r
# library(ggplot2) ## Already loaded

## First get tidied output of the ols_hdfe object
coefs_hdfe = tidy(ols_hdfe, conf.int = TRUE)

bind_rows(
  coefs_fe %>% mutate(reg = "Model 1\nFE and no clustering"),
  coefs_hdfe %>% mutate(reg = "Model 2\nHDFE and twoway clustering")
  ) %>%
  ggplot(aes(x=reg, y=estimate, ymin=conf.low, ymax=conf.high)) +
  geom_pointrange() +
  labs(Title = "Marginal effect of height on mass") +
  geom_hline(yintercept = 0, col = "orange") +
  ylim(-0.5, NA) + ## Added a bit more bottom space to emphasize the zero line
  labs(
    title = "'Effect' of height on mass",
    caption = "Data: Characters from the Star Wars universe"
    ) +
  theme(axis.title.x = element_blank())
```



FWIW, we'd normally expect our standard errors to blow up with clustering. Here that effect appears to be outweighed by the increased precision brought on by additional fixed effects. Still, I wouldn't put too much thought into it. Our clustering choice doesn't make much sense and I really just trying to demonstrate the package syntax.

Aside on standard errors We've now seen the various options that **fixest** has for specifying different standard error structures. In short, you invoke either of the `se` or `cluster` arguments. Moreover, you can choose to do so either at estimation time, or by adjusting the standard errors for an existing model post-estimation (e.g. with `summary.fixest(mod, cluster = ...)`). There are two additional points that I want to draw your attention to.

First, if you're coming from another statistical language, adjusting the standard errors post-estimation (rather than always at estimation time) may seem slightly odd. But this behaviour is actually extremely powerful, because it allows us to analyse the effect of different error structures *on-the-fly* without having to rerun the entire model again. **fixest** is already the fastest game in town, but just think about the implied time savings for really large models.[3] I'm a huge fan of the flexibility, safety, and speed that on-the-fly standard error adjustment offers us. I even wrote a whole blog post about it if you'd like to read more.

Second, reconciling standard errors across different software is a much more complicated process than you may realise. There are a number of unresolved theoretical issues to consider — especially when it comes to multiway clustering — and package maintainers have to make a number of arbitrary decisions about the best way to account for these. See here for a detailed discussion. Luckily, Laurent (the **fixest** package author) has taken the time to write out a detailed vignette about how to replicate standard errors from other methods and software packages.[4]

[3]To be clear, adjusting the standard errors via, say, `summary.fixest()` completes instantaneously.

[4]If you want a deep dive into the theory with even more simulations, then this paper by the authors of the **sandwich** paper is another excellent resource.

Random and mixed effects

Fixed effects models are more common than random or mixed effects models in economics (in my experience, anyway). I'd also advocate for Bayesian hierachical models if we're going down the whole random effects path. However, it's still good to know that R has you covered for random effects models through the **plm** (link) and **nlme** (link) packages.[5] I won't go into detail , but click on those links if you would like to see some examples.

Instrumental variables

As you would have guessed by now, there are a number of ways to run instrumental variable (IV) regressions in R. I'll walk through three different options using the `ivreg::ivreg()`, `estimatr::iv_robust()`, and `fixest::feols()` functions, respectively. These are all going to follow a similar syntax, where the IV first-stage regression is specified in a multi-part formula (i.e. where formula parts are separated by one or more pipes, `|`). However, there are also some subtle and important differences, which is why I want to go through each of them. After that, I'll let you decide which of the three options is your favourite.

The dataset that we'll be using for this section describes cigarette demand for the 48 continental US states in 1995, and is taken from the **ivreg** package. Here's a quick a peek:

```r
data("CigaretteDemand", package = "ivreg")
head(CigaretteDemand)
```

```
##        packs   rprice  rincome  salestax   cigtax  packsdiff  pricediff
## AL 101.08543 103.9182 12.91535 0.9216975 26.57481 -0.1418075 0.09010222
## AR 111.04297 115.1854 12.16907 5.4850193 36.41732 -0.1462808 0.19998082
## AZ  71.95417 130.3199 13.53964 6.2057067 42.86964 -0.3733741 0.25576681
## CA  56.85931 138.1264 16.07359 9.0363074 40.02625 -0.5682141 0.32079587
## CO  82.58292 109.8097 16.31556 0.0000000 28.87139 -0.3132622 0.22587189
## CT  79.47219 143.2287 20.96236 8.1072834 48.55643 -0.3184911 0.18546746
##    incomediff salestaxdiff  cigtaxdiff
## AL 0.18222144    0.1332853 -3.62965832
## AR 0.15055894    5.4850193  2.03070663
## AZ 0.05379983    1.4004856 14.05923036
## CA 0.02266877    3.3634447 15.86267924
## CO 0.13002974    0.0000000  0.06098283
## CT 0.18404197   -0.7062239  9.52297455
```

Now, assume that we are interested in regressing the number of cigarettes packs consumed per capita on their average price and people's real incomes. The problem is that the price is endogenous, because it is simultaneously determined by demand and supply. So we need to instrument for it using cigarette sales tax. That is, we want to run the following two-stage IV regression.

$$Price_i = \pi_0 + \pi_1 SalesTax_i + v_i \qquad \text{(First stage)}$$

$$Packs_i = \beta_0 + \beta_2 \widehat{Price_i} + \beta_1 RealIncome_i + u_i \qquad \text{(Second stage)}$$

Option 3: `fixest::feols()`

Finally, we get back to the `fixest::feols()` function that we've already seen above. Truth be told, this is the IV option that I use most often in my own work. In part, this statement reflects the fact that I work mostly with panel data and will invariably be using **fixest** anyway. But I also happen to like its IV syntax a lot. The key thing is to specify the IV first-stage as a separate formula in the *final* slot of the model call.[6] For example, if we had `fe` fixed effects, then the model call would be `y ~ ex | fe | en ~ in`. Since we don't have any fixed effects in our current cigarette demand example, the first-stage will come directly after the exogenous variables:

[5]As I mentioned above, **plm** also handles fixed effects (and pooling) models. However, I prefer **fixest** and **lfe** for the reasons already discussed.

[6]This closely resembles Stata's approach to writing out the IV first-stage, where you specify the endogenous variable(s) and the instruments together in a slot.

```
# library(fixest) ## Already loaded

iv_feols =
  feols(
    log(packs) ~ log(rincome) | ## y ~ ex
      log(rprice) ~ salestax,   ## en ~ in (IV first-stage; must be the final slot)
    data = CigaretteDemand
    )
# summary(iv_feols, stage = 1) ## Show the 1st stage in detail
iv_feols
```

```
## TSLS estimation, Dep. Var.: log(packs), Endo.: log(rprice), Instr.: salestax
## Second stage: Dep. Var.: log(packs)
## Observations: 48
## Standard-errors: IID
##                  Estimate Std. Error   t value   Pr(>|t|)
## (Intercept)      9.430658   1.358366  6.942648 1.2395e-08 ***
## fit_log(rprice) -1.143375   0.359486 -3.180583 2.6617e-03 **
## log(rincome)     0.214515   0.268585  0.798687 4.2867e-01
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.183555   Adj. R2: 0.393109
## F-test (1st stage), log(rprice): stat = 45.2  , p = 2.655e-8, on 1 and 45 DoF.
##                        Wu-Hausman: stat =  1.102, p = 0.299559, on 1 and 44 DoF.
```

Again, I emphasise that the IV first-stage must always come last in the `feols()` model call. Just to be pedantic — but also to demonstrate how easy **fixest**'s IV functionality extends to panel settings — here's a final `feols()` example. This time, I'll use a panel version of the same US cigarette demand data that includes entries from both 1985 and 1995. The dataset originally comes from the **AER** package, but we can download it from the web as follows. Note that I'm going to modify some variables to make it better comparable to our previous examples.

```
## Get the data
url = 'https://vincentarelbundock.github.io/Rdatasets/csv/AER/CigarettesSW.csv'
cigs_panel =
  read.csv(url, row.names = 1) %>%
  mutate(
    rprice = price/cpi,
    rincome = income/population/cpi
    )
head(cigs_panel)
```

```
##   state year   cpi population    packs    income  tax     price     taxs
## 1    AL 1985 1.076    3973000 116.4863  46014968 32.5 102.18167 33.34834
## 2    AR 1985 1.076    2327000 128.5346  26210736 37.0 101.47500 37.00000
## 3    AZ 1985 1.076    3184000 104.5226  43956936 31.0 108.57875 36.17042
## 4    CA 1985 1.076   26444000 100.3630 447102816 26.0 107.83734 32.10400
## 5    CO 1985 1.076    3209000 112.9635  49466672 31.0  94.26666 31.00000
## 6    CT 1985 1.076    3201000 109.2784  60063368 42.0 128.02499 51.48333
##      rprice  rincome
## 1  94.96438 10.76387
## 2  94.30762 10.46817
## 3 100.90962 12.83046
## 4 100.22058 15.71332
## 5  87.60842 14.32619
## 6 118.98234 17.43861
```

Let's run a panel IV now, where we'll explicitly account for year and state fixed effects.

```
iv_feols_panel =
  feols(
    log(packs) ~ log(rincome) |
      year + state |            ## Now include FEs slot
      log(rprice) ~ taxs,       ## IV first-stage still comes last
    data = cigs_panel
  )
# summary(iv_feols_panel, stage = 1) ## Show the 1st stage in detail
iv_feols_panel
```

```
## TSLS estimation, Dep. Var.: log(packs), Endo.: log(rprice), Instr.: taxs
## Second stage: Dep. Var.: log(packs)
## Observations: 96
## Fixed-effects: year: 2,  state: 48
## Standard-errors: Clustered (year)
##                   Estimate Std. Error       t value   Pr(>|t|)
## fit_log(rprice) -1.279349   2.11e-15 -6.071802e+14 1.0485e-15 ***
## log(rincome)     0.443422   1.41e-15  3.138717e+14 2.0283e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.044789     Adj. R2: 0.92791
##                  Within R2: 0.533965
## F-test (1st stage), log(rprice): stat = 111.0    , p = 7.535e-14, on 1 and 46 DoF.
##                      Wu-Hausman: stat =   6.02154, p = 0.018161 , on 1 and 44 DoF.
```

Good news, our coefficients are around the same magnitude. But the increased precision of the panel model has yielded gains in statistical significance.

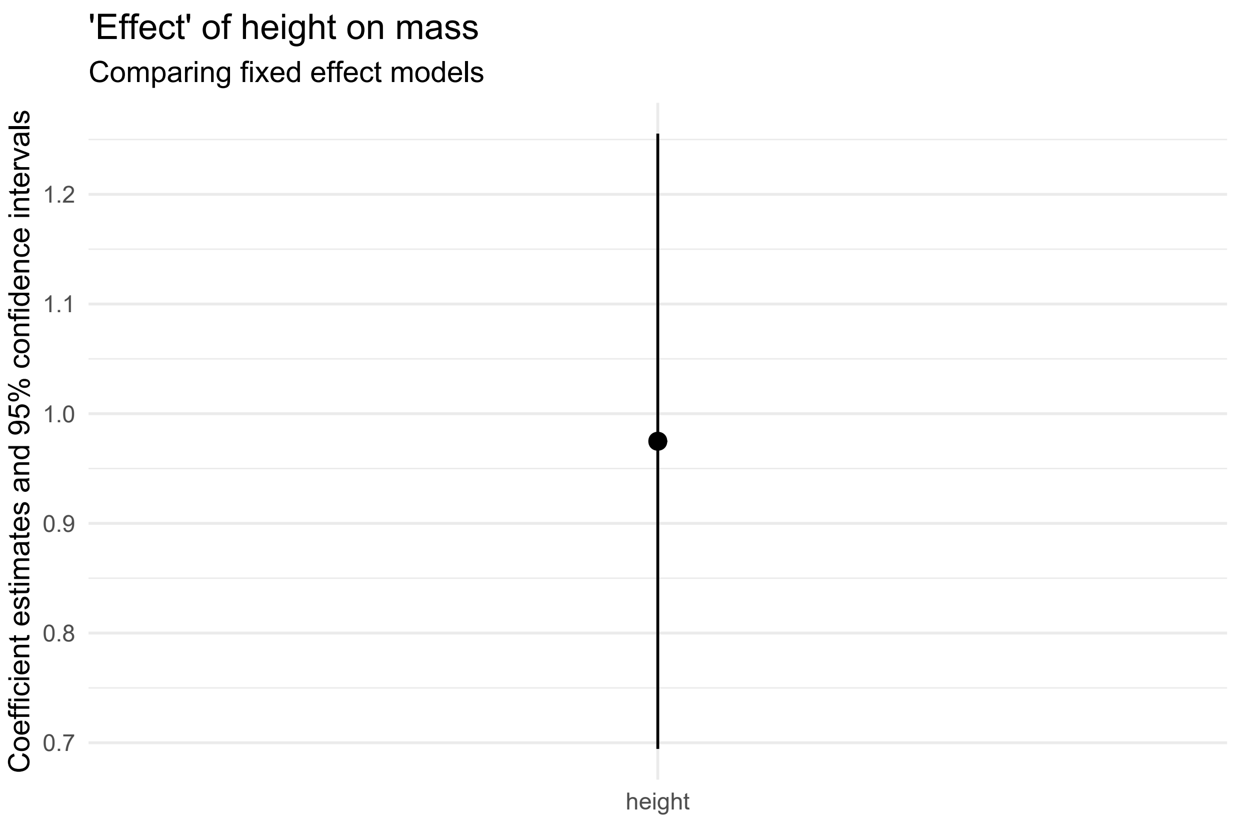
Presentation

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('FE, no clustering' = summary(ols_fe, se = 'standard'))

modelplot(mods) +
  ## You can further modify with normal ggplot2 commands...
  coord_flip() +
  labs(
    title = "'Effect' of height on mass",
    subtitle = "Comparing fixed effect models"
    )
```



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.