Instructor Guide

Nick Huntington-Klein

July 5, 2019

What You Need to Know

Econ 305 consists of two primary components: an R component, and a causal inference component. These are the things that students will need to know by the end of the course, so presumably you'll need to know them before the course starts!

We can think of the R component as having the following learning goals:

- Comfort and familiarity with basic programming and object manipulation in R, and the use of the RStudio environment
- Ability to find and load data
- Ability to manipulate data using the `dplyr` package
- Ability to perform basic statistical calculations in R, including "explaining" one variable with another (i.e. taking means within categories or bins)
- Ability to create and label graphs

We can think of the causal inference component of the class as having the following learning goals:

- Understanding the concept of a data-generating process, and seeing the purpose of data analysis as uncovering that DGP
- Ability to represent a data-generating process in a causal diagram
- Given a causal diagram, ability to determine how a causal effect of interest can be identified
- Conceptual familiarity with each of the causal inference "toolbox" methods: controlling, matching, fixed effects, DID, RDD, IV
- Ability to perform each of the toolbox methods on a very basic level (no regression, standard errors, or hypothesis tests)

In This Document

I will assume that, as an instructor of this class, you either are already familiar with, or can learn by reviewing the material of this course or by other methods…

- Basic use of R and RStudio[1]
- Good practices in graphmaking
- The concepts of DGP and identification
- The toolbox methods

and so will not be covering these.

Instead, I will be covering (and linking to outside resources for):

- What distinguishes this class from others

[1] If you're still new to R, I recommend the videos I have posted as course material, the Florian Heiss textbook which tracks baby-Wooldridge, or any of the great resources available here.

- Rstudio.cloud, which is Rstudio in the cloud, which makes handling assignments easy, and which many students are likely to use since it requires no installation
- RMarkdown, which was used to create all the course slides (and the cheat sheets, and this document). You are, by the way, totally free to use all of my slides if you like. But if you want to modify them at all you'll need to know RMarkdown.
- Some R stuff, `dplyr` in particular, coming from other statistics packages, particularly Stata
- The `dagitty` package and `ggdag`
- Preparing data for student use
- `ggplot2`, which makes nicer-looking graphs than base R and which you may want to use to make graphs for class
- Causal diagrams
- The toolbox methods without regression

What Distinguishes This Class from Other Econometrics Classes?

This class has two main prongs - programming and causal inference. Both are different from the way that they might be typically taught in econometrics classes.

On the programming side:

- The goal of teaching programming skills in this class is *not* to teach students the way to carry out various methods. It is to teach a *general familiarity with how to manipulate data.* This familiarity is then put into action when carrying out methods of interest.
- All programming should be done *as close to the data as possible.* Students should never use a black-box method that they do not understand. Ideally, once they know how to do something in code, they'd be able to replicate it by hand if they had to.
- Putting the first two points together, this is why, for example, when taking means within bins, I have students use the series of commands "divide the data into groups" and "take the mean of Y within each group" rather than, for example, regressing Y on a set of group indicators and taking fitted values.
- By making data *analysis* something that is done with *data manipulation* tools, we prepare students for when they will be faced with data that is not already in ready-to-use format.
- There is a heavy emphasis on the statistical concept of "explaining". In this class, "the part of Y explained by X" is the mean of Y within values of X (or bins of X, if X is continuous). This concept allows calculations to be kept close to the data, and allows causal inference methods to be taught without the use of regression. Regression, as students learn in the last week of my version of the class, is just one way to use X to explain Y. Thinking of regression in this way will help you get into the philosophy of the class, and will also make it easier to understand this class's implementation of the toolbox methods.

On the causal inference side:

- Everything on the causal inference side revolves around the concept of the *data generating process.* Students should learn that there is some "true model" that generated the data, and understand that their task is in using the observed data to uncover that process. In accordance with this (and with some of the programming goals), students will learn to simulate data from a DGP, and then try to recover the DGP they used.
- The data generating process is *not* represented as a series of equations, but rather a series of causal relationships. We don't need linear relationships and we don't need to bother figuring out what a coefficient is. These are *specifications* of the DGP, not the DGP itself, so we don't need them. Similarly, standard errors are not part of the DGP either, so we don't need that.
- Students should understand that removing variation explained by X, and selecting a sample in which there is little to no variation in X, are both ways of controlling for a variable.

- DGPs are represented as causal diagrams. Students are forced to consider how each variable interacts with each other variable. Unobserved variables must be considered individually and explicitly, rather than just being "part of the error term." There is no error term.
- Causal diagrams have a simple list of rules that determine whether a causal effect is identified, and how that identification can occur. This allows all discussion of modeling to center around what identification is and how it can be done.
- Students learn about *collider bias* and other ways in which controlling for a variable *harms* identification.
- Because there is a strong, central, and *general* idea of what identification is, causal inference toolbox methods can be taught quickly as special cases of the general concept of an identification plan.
- The goal is not to teach *best practices* for these methods, but simply to show them *conceptually*, as ways of performing a proper identification, given the causal diagram. Robustness checks should receive little attention unless they can be specifically linked to ways in which our causal diagram might be incorrect.

Rstudio.cloud

Rstudio.cloud is a cloud-based version of Rstudio. At the very least, I recommend letting students know about it, as it is free to use, and allows them to access R and RStudio without having to do any installation, which is easy.

I recommend going beyond that as well, as Rstudio.cloud is designed for use in classroom environments in a few other ways. It allows you to set up shared environments for each assignment or lecture so that students can follow along, or so you can pre-install any packages they may need, or pre-load any data sets, so they don't have to bother with that. Personally I want them learning to install packages and load data anyway, but that might not be where you want to focus your time.

Using Rstudio.cloud in this shared-environment way is not free but it is cheap - $9/month last I checked, and that's just for *you* to make the environment. Once you've paid your $9 and set things up, it's free for them to access it.

I won't go too deeply into the *how* of this, as their documentation does it better than I could. Go to their Guide page and scroll down to "Using Private Spaces in Courses and Workshops."

RMarkdown

Markdown is basically the simplest version of a markup language (think HTML or, at a stretch, LaTeX). RMarkdown is an implementation of Markdown that's integrated with R. If you've ever used Beamer, using RMarkdown for slides will be a very natural transition (it's actually considerably easier to use than Beamer).

I use RMarkdown to make the slides for class because it makes it very easy to incorporate R code into the slides. A "code chunk" can be included in the text like so: `` `code` `` becomes `code`. Or you can include a block of code (the `echo=TRUE, eval=FALSE` means "show (echo) the code back to me, but don't (eval)uate it":

~~~
```{r, echo=TRUE, eval=FALSE} several lines of explanatory code here ```
~~~

```
several lines of explanatory code here
```

Even better, you can have the code actually run in your slides - either showing the code and the result (`echo = TRUE, eval=TRUE`) or just the result (`echo=FALSE, eval=TRUE`). Not only does this automatically update your slides whenever you change a code example, but also it lets students look at the code you used and the result. This can be done in-line: `` ` r 2+2  ` `` becomes 4. You can also do it in multiple-line sections on its own:

~~~
```{r, echo=TRUE, eval=TRUE} 2+2 ```
~~~

becomes

```r
2+2
```

```
## [1] 4
```

RMarkdown can be installed with `install.packages('rmarkdown')` and is very well-integrated into RStudio. You can edit your presentations directly in RStudio, and hit the "Knit" button to build them. Do note that if you want to use RMarkdown to generate PDFs (for example to modify & rebuild my cheat sheets, or if you want to literally make Beamer slides from RMarkdown, which is possible), you'll need to have LaTeX installed (which you could just do with `install.packages('tinytex')` and `tinytex::install_tinytex()` from inside R if you don't have it).

See the code for my slides for examples, and see more information in the [RMarkdown guide])https://rmarkdown.rstudio.com/lesson-1.html), specifically the section on slide presentations. For the slides I made, I used the `revealjs` version, but that's just because I liked the look, they're not necessarily better. If you want to get real fancy, some of the other versions can incorporate "Shiny" tools and you could make your slides interactive.

Adjusting to R and dplyr from Other Software

(Note I'm writing this with Stata users in mind, since I know Stata, but I'll try to keep it general for everyone)

First thing's first: the help functions. `help(whatever)` works well most of the time. The Help pane on the bottom-right in RStudio also has a search function, and of course there's the internet. Also, frustratingly, sometimes instead of `help(whatever)` you need to do `??whatever`. You can see a list all the help files for a particular package with `help(package='packagename')`.

Now onto the meat!

The biggest adjustment coming to R from something like Stata, SPSS, or EViews (although similar to what you might get in Python, with Matlab/Julia somewhere in between) is that in R, you don't so much *run commands* as you *manipulate objects*. When you load in a data set, you've *created a data.frame-type object and stored it in memory*. If you change the values of one of the variables in that data set, you're not *running a command that changes that data*, you're *creating a new object that has had that change-values function applied and overwriting the old version of the object*.

Going even further, the variables *in* a `data.frame` are objects themselves that you can manipulate, too. Formally, `data.frames` are just a `list` of `vectors`.

To see what I mean, let's load in some data, make some changes, and run a regression.

```r
#we'll be using dplyr in a second so may as well load in the tidyverse
library(tidyverse)
#load in data
data(LifeCycleSavings)

#If you put an object on a line by itself, that means "show me this object"
#So if I say LifeCycleSavings here by itself, it will show me all 50 obs
#I'll just pick the first six obs by indexing
LifeCycleSavings[1:6,]
```

```
##               sr pop15 pop75     dpi ddpi
## Australia 11.43 29.35  2.87 2329.68 2.87
## Austria   12.07 23.32  4.41 1507.99 3.93
## Belgium   13.17 23.80  4.43 2108.47 3.82
## Bolivia    5.75 41.89  1.67  189.13 0.22
## Brazil    12.88 42.19  0.83  728.47 4.56
## Canada     8.79 31.72  2.85 2982.88 2.43
```

```
#LifeCycleSavings is just a list of variables, each of which is an object.
#Let's pull one of those objects out and look at it
LifeCycleSavings$pop15
```

```
##  [1] 29.35 23.32 23.80 41.89 42.19 31.72 39.74 44.75 46.64 47.64 24.42
## [12] 46.31 27.84 25.06 23.31 25.62 46.05 47.32 34.03 41.31 31.16 24.52
## [23] 27.01 41.74 21.80 32.54 25.95 24.71 32.61 45.04 43.56 41.18 44.19
## [34] 46.26 28.96 31.94 31.92 27.74 21.44 23.49 43.42 46.12 23.27 29.81
## [45] 46.40 45.25 41.12 28.13 43.69 47.20
```

```
#Basically any calculation we do is just creating an object with the result
#(btw, keep in mind that most base R functions, if given a missing obs (NA)
#will return a NA result, so be sure to do na.rm=TRUE, or whatever the appropriate
#version is for the command you're running)
mean(LifeCycleSavings$pop15,na.rm=TRUE)
```

```
## [1] 35.0896
```

```
#which means we can store it
meanpop15 <- mean(LifeCycleSavings$pop15)
meanpop15
```

```
## [1] 35.0896
```

```
#That also means that, UNLESS we store it, it doesn't stick around
LifeCycleSavings$pop15 - 10
```

```
##  [1] 19.35 13.32 13.80 31.89 32.19 21.72 29.74 34.75 36.64 37.64 14.42
## [12] 36.31 17.84 15.06 13.31 15.62 36.05 37.32 24.03 31.31 21.16 14.52
## [23] 17.01 31.74 11.80 22.54 15.95 14.71 22.61 35.04 33.56 31.18 34.19
## [34] 36.26 18.96 21.94 21.92 17.74 11.44 13.49 33.42 36.12 13.27 19.81
## [45] 36.40 35.25 31.12 18.13 33.69 37.20
```

```
#Note the mean has not dropped by 10
mean(LifeCycleSavings$pop15)
```

```
## [1] 35.0896
```

```
#Now we overwrite that object
LifeCycleSavings$pop15 <- LifeCycleSavings$pop15 - 10
#Now we're cooking
mean(LifeCycleSavings$pop15)
```

```
## [1] 25.0896
```

```
#Even if we're running a regression, we don't "run a regress command"
#so much as we "create a regression object" which has its own
#sub-objects.
my_regression <- lm(pop15~pop75,data=LifeCycleSavings)
#regression objects aren't much to look at
my_regression
```

```
## 
## Call:
## lm(formula = pop15 ~ pop75, data = LifeCycleSavings)
## 
## Coefficients:
## (Intercept)        pop75  
##      39.859       -6.441
```

```
#but regression SUMMARY objects? different story!
summary(my_regression)
```

```
## 
## Call:
## lm(formula = pop15 ~ pop75, data = LifeCycleSavings)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -10.5466  -2.3602   0.3677   2.5504   8.2894 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  39.8593     1.1227   35.50   <2e-16 ***
## pop75        -6.4412     0.4277  -15.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 3.864 on 48 degrees of freedom
## Multiple R-squared:  0.8253, Adjusted R-squared:  0.8217 
## F-statistic: 226.8 on 1 and 48 DF,  p-value: < 2.2e-16
```

```
#and remember, everything is objects
#I don't run the "generate predicted values" function,
#I instead pull out the "fitted values" object already in the regression
my_regression$fitted.values
```

```
##      Australia        Austria        Belgium        Bolivia         Brazil 
##      21.373013      11.453526      11.324702      29.102483      34.513112 
##         Canada          Chile          China       Colombia     Costa Rica 
##      21.501838      31.228088      35.543708      33.031631      32.516333 
##        Denmark        Ecuador        Finland         France        Germany 
##      14.545314      32.194271      24.593626       9.585571      18.281225 
##         Greece      Guatamala       Honduras        Iceland          India 
##      19.891531      34.255463      36.123419      20.020356      33.675753
##        Ireland          Italy          Japan          Korea     Luxembourg 
##      12.870596      17.443866      27.556589      33.997814      15.833559 
##          Malta         Norway    Netherlands    New Zealand      Nicaragua 
##      23.949503      16.220033      18.925348      19.440646      32.065447 
##         Panama       Paraguay           Peru    Philippines       Portugal 
##      32.129859      33.096043      31.614561      32.645157      21.501838 
##   South Africa South Rhodesia          Spain         Sweden    Switzerland 
##      25.173336      30.068667      21.373013      10.616167      15.833559 
##         Turkey        Tunisia United Kingdom  United States      Venezuela 
##      32.902806      32.065447      11.131465      17.765927      34.062227 
##         Zambia        Jamaica        Uruguay          Libya       Malaysia 
##      36.252243      28.716010      22.339197      26.525993      35.608121
```

Once you've got your head around the idea that what you're doing in R is *manipulating objects*, you're like 90% of the way there. The rest is mostly just syntax and learning function names.

Speaking of which, I should briefly discuss the use of the `dplyr` verbs. In class we manipulate data using `dplyr`, which is a package that must be installed and loaded in (either via `library(dplyr)` or `library(tidyverse)`, the tidyverse being a whole buncha packages that represent a certain philosophy of how to use R).

The reasons why I like to use `dplyr` rather than base-R:

- I think it's a more intuitive syntax
- It's very easy to chain together multiple manipulations using pipes (`%>%`)
- It's very easy to create within-group summary statistics
- If there are any missing observations in your data, base-R data manipulation quickly becomes a huge pain
- It has similarities in how it works to (in different ways) Stata data manipulation and SQL. So if there's a transition later to either of those (let's be honest, SQL being more likely), they'll be more ready

If you are coming from Stata, I strongly recommend this guide which puts Stata commands line-by-line up against `dplyr` commands so you can see how they translate. You can also check out my `dplyr` videos or the `dplyr` cheat sheet, both included as course material.

I'll leave the advanced stuff to those sources. The real central things you need to know are:

```
#Select variables with select()
just_pop_vars <- select(LifeCycleSavings,pop15,pop75)
just_pop_vars[1:6,]
```

```
##           pop15 pop75
## Australia 19.35  2.87
## Austria   13.32  4.41
## Belgium   13.80  4.43
## Bolivia   31.89  1.67
## Brazil    32.19  0.83
## Canada    21.72  2.85
```

```
#Select observations with filter()
just_low_pop <- filter(LifeCycleSavings,pop15 < median(LifeCycleSavings$pop15))
just_low_pop[1:6,]
```

```
##      sr pop15 pop75     dpi ddpi
## 1 11.43 19.35  2.87 2329.68 2.87
## 2 12.07 13.32  4.41 1507.99 3.93
## 3 13.17 13.80  4.43 2108.47 3.82
## 4  8.79 21.72  2.85 2982.88 2.43
## 5 16.85 14.42  3.93 2496.53 3.99
## 6 11.24 17.84  2.37 1681.25 4.32
```

```r
#Sort the data with arrange()
arrange(LifeCycleSavings[1:6,],pop15)
```

```
##      sr pop15 pop75     dpi ddpi
## 1 12.07 13.32  4.41 1507.99 3.93
## 2 13.17 13.80  4.43 2108.47 3.82
## 3 11.43 19.35  2.87 2329.68 2.87
## 4  8.79 21.72  2.85 2982.88 2.43
## 5  5.75 31.89  1.67  189.13 0.22
## 6 12.88 32.19  0.83  728.47 4.56
```

```r
#Create new variables with mutate()
LifeCycleSavings <- mutate(LifeCycleSavings,
                           low_pop = pop15 < median(LifeCycleSavings$pop15),
                           pop_minus_10 = pop15 - 10)

#Collapse the data to a set of summary functions with summarize()
summarize(LifeCycleSavings,
          pop15 = mean(pop15),
          sr = min(sr))
```

```
##     pop15  sr
## 1 25.0896 0.6
```

```r
#Chain together multiple commands with the pipe %>%
#which takes an object and pipes it through to be the first
#argument in the next function
LifeCycleSavings %>%
  select(pop15,pop75) %>%
  filter(pop15 < median(LifeCycleSavings$pop15)) %>%
  mutate(the_best_number = 10) %>%
  summarize(best_num = first(the_best_number),
            pop15 = mean(pop15))
```

```
##   best_num   pop15
## 1       10 16.7532
```

```r
#Organize the data into groups with group_by()
#such that all commands are done by-group
LifeCycleSavings %>%
  group_by(low_pop) %>%
  summarize(mean_sr_by_pop = mean(sr))
```

```
## # A tibble: 2 x 2
##   low_pop mean_sr_by_pop
##   <lgl>            <dbl>
## 1 FALSE             7.30
## 2 TRUE             12.0
```

```
#If using group_by(), keep in mind that object
#will STAY GROUPED until you ungroup() it.
#If overwriting your original data, remember to do this
#so you don't accidentally run things grouped later!
LifeCycleSavings <- LifeCycleSavings %>%
  group_by(low_pop) %>%
  mutate(mean_sr_by_pop = mean(sr)) %>%
  ungroup()
head(LifeCycleSavings)
```

```
## # A tibble: 6 x 8
##      sr pop15 pop75   dpi  ddpi low_pop pop_minus_10 mean_sr_by_pop
##   <dbl> <dbl> <dbl> <dbl> <dbl> <lgl>          <dbl>          <dbl>
## 1 11.4   19.4  2.87 2330.  2.87 TRUE            9.35          12.0
## 2 12.1   13.3  4.41 1508.  3.93 TRUE            3.32          12.0
## 3 13.2   13.8  4.43 2108.  3.82 TRUE            3.8           12.0
## 4  5.75  31.9  1.67  189.  0.22 FALSE          21.9            7.30
## 5 12.9   32.2  0.83  728.  4.56 FALSE          22.2            7.30
## 6  8.79  21.7  2.85 2983.  2.43 TRUE           11.7           12.0
```

dagitty and ggdag

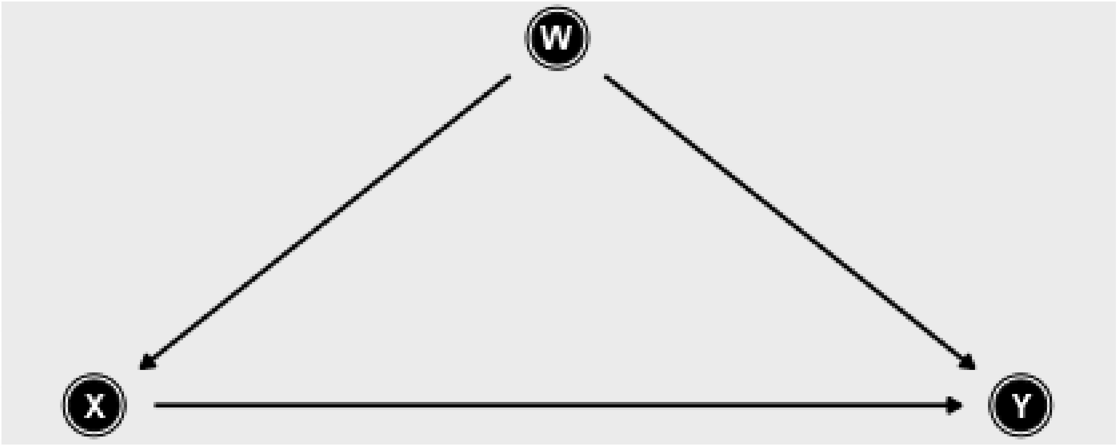
This class will be making use of causal diagrams. Students will be able to draw causal diagrams using dagitty.net, which is a great drawing tool.

You can do this too if you'd like to save the images, put them where you need them, remember filenames, etc.. If you want to put causal diagrams in your slides, you'll probably want to draw them in R using code. The `dagitty` and `ggdag` packages make this possible.

The `dagify` function creates a DAG object (directed acyclic graph, aka causal diagram). Specify an arrow from `X` to `Y` with `Y~X`. You can also, if you like, specify the coordinates of each node. This is a bit tedious but it does make it look nicer. Unfortunately, as far as I can tell there's no way to make the variable names hang *outside* the circles, so you're stuck using very short variable names.

Once you have your DAG object, run it through `tidy_dagitty()`, and then finally plot it using `ggdag()`, as below.

```
dag <- dagify(Y~X+W,
              X~W,
                coords=list(
                x=c(X=1,W=2,Y=3),
                y=c(X=1,W=2,Y=1)
              )) %>% tidy_dagitty()
ggdag(dag,node_size=10)
```



A few extra notes on this (and other graphs you include in RMarkdown): First, the `{r}` call that starts off this code chunk should probably have the `fig.width` and `fig.height` arguments to tell it how big to make the image. Second, if you're on Windows, you should load the `Cairo` package and add `dev='CairoPNG'` as well, or else your results will be fuzzy. That particular code chunk was run with `{r, dev='CairoPNG', echo=TRUE, fig.width=6,fig.height=2.5}`.

Preparing Data for Student Use

R makes it very easy to get data. There are enough datasets just included in regular packages that you should be able to cover all the example problems you want. In RStudio you can just type `data(` and it will autocomplete with a list of all the datasets available in the packages you have loaded up. Check out in particular the `Ecdat` and `wooldridge` packages, and also the `AER` package, which has many data sets from Stock & Watson. This page covers many (although certainly not nearly all) in-package datasets. If you want students to download real, standard data sets, you may also consider this list I've prepared of packages that make it easy to download data from, for example, the World Bank.

You may also be interested in preparing your own data sets. There are several ways to do this.

One is you can just pass them whatever data file you like. R can open a lot of them. Base R has `read.csv` to open CSV files. If you load the `haven` package, you get `read_csv`, `read_stata`, and `read_spss`, among many others. For Excel files there's `read_excel` in the `readxl` package.

If you want to save them the trouble, you can load in the data yourself, and then save it in .Rdata format using the `save()` command. As a bonus, you can actually save *as many objects as you like* in one file this way. Why not pass them two data sets at once? Or a data set and a matrix of parameters? Many options.

ggplot2

The `ggplot2` package (available on its own, or, like `dplyr`, it's also part of the `tidyverse`) is probably the best data-viz tool on the market. In my opinion it's just a shade too hard to use to force the students to use it, but when you're making your own plots you might want to consider making them look that extra shade nicer.

The `ggplot` function (confusingly, the package is `ggplot2`, the function is `ggplot`) is way, way, way too deep for me to go into in detail. I'll instead recommend you to this book chapter by the package author Hadley Wickham. But I will give you a very basic overview and a scatterplot example.

To use `ggplot`, we need a few things: some data, an *aesthetic* (`aes()`), and a *geometry*.

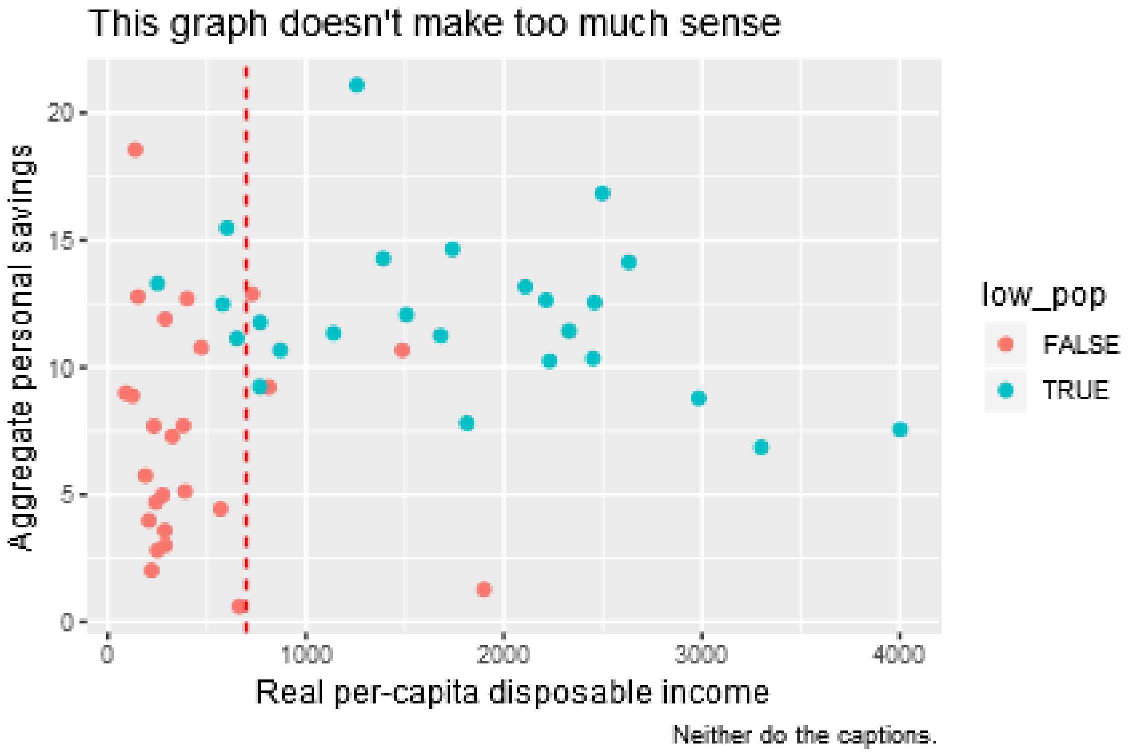
The aesthetic tells `ggplot` the important parts of the graph - what variable is on the x axis? What's on the y? Are we making the color, shape, or size different based on the value of another variable?

The geometry tells `ggplot` what it's doing with the data. Are we plotting points? A line plot? A regression line?

We can then add on multiple pieces and stack them on top of each other by simply adding on (`+`) more geometries.

For example, let's make a scatterplot which colors the points differently by `low_pop`, and puts a vertical red dashed line at the median of the x-axis data. And of course we'll label things properly. Note, by the way, that `color` *inside* the `aes` colors the dots by values of `low_pop`, while `color` *outside* the `aes` but inside the geometry colors the entire line.

```
#let's pipe our data into ggplot, and set our aesthetic
LifeCycleSavings %>% ggplot(aes(x=dpi,y=sr,color=low_pop)) +
  #I want a scatterplot, i.e. a point geometry
  geom_point() +
  #I also want a vertical line. Note that the elements of the
  #aesthetic may change depending on what geometry you're using
  geom_vline(aes(xintercept=median(LifeCycleSavings$dpi)),color='red',linetype='dashed')+
  #and we'll add axis labels
  labs(x='Real per-capita disposable income',
       y='Aggregate personal savings',
       title='This graph doesn\'t make too much sense',
       caption='Neither do the captions.')
```



Causal Diagrams

Causal diagrams are actually pretty simple. Just write down all the relevant variables including unobserved ones (or a reasonably edited set of them), theorize which variables cause which others, and draw arrows from cause to effect. If two variables share an unobserved common cause, then give that common cause a name (maybe "U"?) and then draw arrows from that common cause. Now you've got a structural model!

The basic rules of causal diagrams are covered in the Causal Diagrams Cheat Sheet and the Dagitty cheat sheet (available as course material). I'll also recommend the books Causal Inference: the Mixtape by Cunningham (free), Counterfactuals and Causal Inference by Morgan & Winship, or, if you don't mind a very dismissive tone about any other way of doing causal inference, The Book of Why by Pearl for more high-level explanations (which may actually help, after all, the stuff you know best is at a higher level of explanation anyway).

I'll just add a few things here:

1. Causal diagrams do not specify a functional form, so they can be a little difficult to use them when functional form is an important part of identification. This largely comes up in the case of interaction terms. Formally, `X -> Y <- Z` allows for an interaction between `X` and `Z`. But that's not exactly stellar pedagogy. I just toss `X*Z` on there as its own variable, even though that's not formally correct. Similarly, I have a `X > Cutoff` variable on its own when doing RDD.
2. Causal diagrams are largely just another way of representing the potential outcomes-style modeling we generally do in our heads as economists. It just forces us to write it down explicitly, and greatly simplifies the rules for determining identification. Don't be looking for this to be an entirely different concept of causality - you'll confuse yourself! It's not as new as it seems.
3. The rules for identification are really as easy as they seem, and can be translated into language you may be more comfortable with. If you want to identify the effect of X on Y, a "back door" is any path you can walk from X to Y that contains an arrow that *causes X*. An open (uncontrolled) back-door path makes you unidentified. You can think of this as X being correlated with the error term, or your effect being biased by [a variable on the open back-door path]. Identifying the effect of X on Y by closing all back doors is basically selection on observables.
4. Similarly, a variable Z on a causal diagram can be an instrument if it causes X (relevance), and all paths from Z to Y either go through X or are closed by controlling for them (validity, or validity conditional on controls).
5. The diagram also makes it clear that you don't want to control for variables caused by X, as they're on a *front-door path*, not the *back-door paths* that cause problems. We know this as post-treatment bias.
6. Two things that may be a bit more new are the "front-door criterion" which doesn't have a whole lot of real-world applications, and collider bias, which does. Basically, if there's a variable on a path which has arrows on both sides pointing towards it, that variable is a collider. For example, your eye color is caused by your mom's eye color and your dad's. So on the path `MomEyes -> YourEyes <- DadEyes`, `YourEyes` is a collider. **If a path has a collider on it, it's pre-closed, and controlling for that variable opens the path back up, potentially destroying your identification.** In the eyes example, mom's eyes and dad's eyes are uncorrelated. But *given* your eye color, they are correlated - if you have brown eyes, then mom and dad can't both be blue. So you have to be careful what you control for! The closest concept we have is sample selection bias - we know that height causes your basketball skill to improve. But if we take a sample of NBA players (in effect, controlling for getting on an NBA team, which is caused by both your height and your basketball skill, i.e. a collider), we find no relationship between height and skill; the estimate is biased.

The Toolbox Without Regression

These are each covered in more detail in the lecture slides. But I'll have a quick recap here, and some additional notes aimed at you since you *are* familiar with the regression versions.

Controlling for a variable

To get the effect of X on Y while controlling for W, we split W up into bins (if necessary), and then we Frisch-Waugh-Lovell the thing, taking the means of X and Y within values of W, and subtracting out those means. Note the `cut()` function allows you to split W into bins, and `dplyr`'s `group_by` makes taking means within values of W easy. Taking it back to the lingo of the class, we're finding what part of the X/Y relationship is *explained* by W, and then removing that part so as to close a back door.

Note that this does make it kind of tricky to control for *more than one thing*. So you are boxed in a bit in that you have to limit yourself to using a single control.

The course is also explicit that this is only *one way* to control for a variable. Even beyond all the other ways to explain X and Y with W, anything that ensures we get rid of variation in W works, including selecting a sample in which there is no variation in W, or matching.

Fixed effects

Given the above approach to controlling, fixed effects falls right into our laps. It's free. Just control for identity. Done. Describe it in those terms, even, since that's what we're doing: bundling up all the individual-level time-invariant variables into a single back door and closing that sucker. This concept of fixed effects also lets you make very explicit what FE does *not* control for (anything time-varying within individual - these should end up on your causal graph, back doors unclosed!). Consider showing the same causal diagram before and after you bundle up all those individual-level back door variables into one.

Matching

I do teach matching, even though it's not super econ-y, because it reinforces this idea of there being other ways to control (it is, in effect, a way of selecting a sample in which the controls don't vary), and makes it very easy to segue into talking about treated and untreated groups.

Rather than use the matching methods you may be more familiar with like PSM, IPW, or nearest-neighbor Mahalanobis matching, this course uses Coarsened Exact Matching, which is actually a newer method that is getting a lot of attention generally because it does very good things in large data sets. But we're mostly interested in it because you can do it by hand.

It's dead simple. Coarsen (bin) any continous variables. Then look for *exact* matches on your set of matching variables. Toss anyone without a match. Compare means.

Note that, because this is so easy, this is the course's stock way of controlling for multiple things at once.

Difference in Differences

Conceptually, the way this is introduced is first as an event study. However, event studies have problems in that time might be a confounder. So you have to introduce different groups so as to be able to control for time. Of course, then you have group membership on a back door so you have to control for that too.

It's basically another extension of controlling. This time we control by differencing things out. We remove variation in the controls by literally subtracting it out. First controlling for time by subtracting within-group before-after differences, then controlling for group by subtracting between-group differences in those before-after differences.

Regression Discontinuity

This is the toughie without regression, as you might expect given the name. But basically what we're doing is just doing a zero-order polynomial RDD (i.e., flat), and then picking a bandwidth. Once you do that, you're just comparing means above and below the cutoff. Which is actually great, as it forces you to focus just on the concept of how RDD is supposed to identify things, and makes it conceptually easy to do something like a balance test if you like. Sure, it's a low-power way to do RDD, but we're not doing a hypothesis test anyway.

To distinguish RDD from other methods, it's a good idea to show an unmeasured confounder on the causal diagram that the RDD will let you ignore. Can't do that with controls!

Instrumental Variables

This is pitched as the *opposite* of controlling for a variable. You use variation in the experiment to explain X and Y, but instead of subtracting out that variation, you ONLY use that variation. It's a very clean conceptualization of IV!

Once you've done this, if Z is binary, you've just got a Wald estimator, so calculate that. If Z is binned at multiple levels, get the correlation between explained X and explained Y, just like we'd get the correlation between residual X and residual Y when controlling.

Causal diagrams make it easy to understand how controls really work in IV. Maybe consider adding a back door for your IV so you can make it a valid IV by controlling. There's a reason that first stage has all the controls of the second!