Lab 3

Ray Caraher

2025-03-03

Setup

Setting up our script

Before we get into any real coding, let's make sure that the preamble for our code looks good. Here is how I set it up:

A bunch of text detailing how the loading of the packages should print when this is run.

```
## Load packages

library(haven)
library(stargazer)
library(fixest)
library(tidyverse)
## Set options

options(scipen = 999)

## Clear environment

rm(list = ls())

## Set directories

base_directory <- '/Users/rcaraher/Library/CloudStorage/OneDrive-UniversityofMassachusetts/Academic/Teach
data_directory <- file.path(base_directory, 'Data')
results_directory <- file.path(base_directory, 'Results')
```

Optimization and Poisson

Count Data and Poisson Regression

If our data primarily a small number of integers rather than a continuous variable, OLS is consistent but not very precise.

Often makes more sense to estimate a Poisson regression.

Let's first grab our data:

Getting data

We can directly read data into R from a URL:

```
award_data <- read.csv("https://stats.idre.ucla.edu/stat/data/poisson_sim.csv")
```

Data

Let's look at the data:

```
glimpse(award_data)
```

```
## Rows: 200
## Columns: 4
## $ id          <int> 45, 108, 15, 67, 153, 51, 164, 133, 2, 53, 1, 128, 16, 10
## $ num_awards <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ prog        <int> 3, 1, 3, 3, 3, 1, 3, 3, 3, 3, 3, 2, 3, 3, 3, 1, 1, 3, 2,
## $ math        <int> 41, 41, 44, 42, 40, 42, 46, 40, 33, 46, 40, 38, 44, 37, 4
```

Data

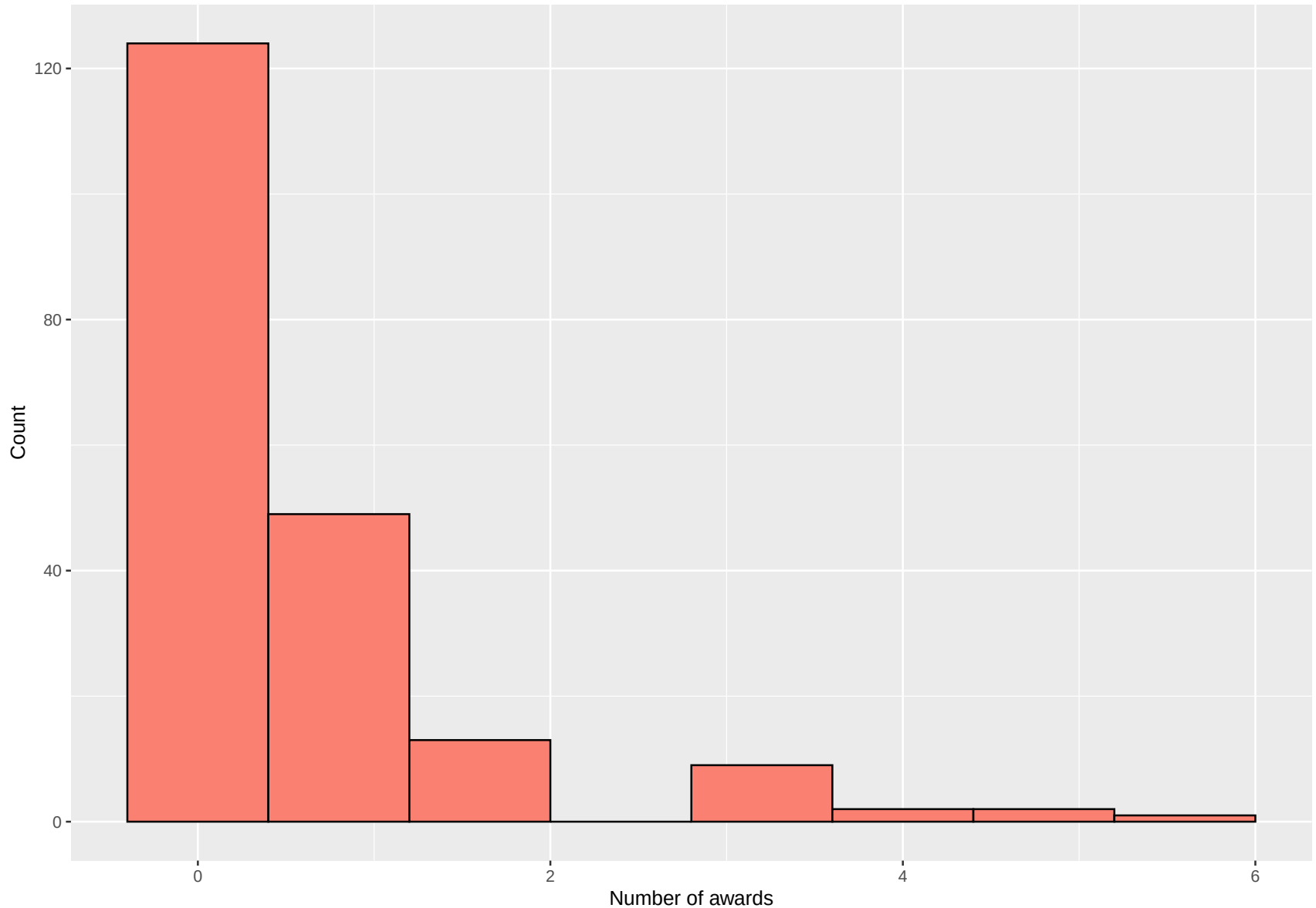
Let's look at the data:

```
p1 <- ggplot(award_data) +
    geom_histogram(aes(x = num_awards),
                   binwidth = 0.8,
                  color = "black", fill = "salmon") +
  labs(x = "Number of awards", y = "Count")
```

Data

Let's look at the data:

```
print(p1)
```



Data

This data looks like a good candidate for a Poisson regression.

Why? Small number of heavily skewed-right integers.

OLS

Let's first look at what the OLS regression looks like:

```
award_data <- award_data %>%
  mutate(prog1 = ifelse(prog == 1, 1, 0),
         prog2 = ifelse(prog == 2, 1, 0),
         prog3 = ifelse(prog == 3, 1, 0)
  )

summary(lm(num_awards ~ prog2 + prog3 + math, data = award_data))
```

```
##
## Call:
## lm(formula = num_awards ~ prog2 + prog3 + math, data = award_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7311 -0.5618 -0.1537  0.2851  4.4126
##
## Coefficients:
##             Estimate Std. Error t value      Pr(>|t|)
## (Intercept) -2.195504   0.411417  -5.336 0.00000026046 ***
## prog2        0.478613   0.168956   2.833        0.0051 **
## prog3        0.212506   0.187433   1.134        0.2583
## math         0.047889   0.007773   6.161 0.00000000403 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Poisson Regression

Let's compare that to the Poisson regression!

```
poisson_reg1 <- glm(num_awards ~ prog2 + prog3 + math,
                    family = "poisson", data = award_data)
summary(poisson_reg1)
```

```
##
## Call:
## glm(formula = num_awards ~ prog2 + prog3 + math, family = "poisson",
##     data = award_data)
##
## Coefficients:
##             Estimate Std. Error z value             Pr(>|z|)
## (Intercept) -5.24712    0.65845  -7.969 0.0000000000000016 ***
## prog2        1.08386    0.35825   3.025              0.00248 **
## prog3        0.36981    0.44107   0.838              0.40179
## math         0.07015    0.01060   6.619 0.0000000000362501 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 287.67  on 199  degrees of freedom
## Residual deviance: 189.45  on 196  degrees of freedom
## AIC: 373.5
##
```

Poisson Regression

How do we interpret these estimates? The estimate on `prog2` suggests that being enrolled in program 2 increases a student's (log) expected number of awards by about 1.1.

How Does Poisson Work?

Poisson regression works by maximizing a likelihood function.

The likelihood function asks the question: given various different parameter values, how *likely* is it that we observe the data that we do?

The likelihood function for a sample of N observations of y_i distributed Poisson with mean and variance $exp(X_i'\beta)$ is

$$L = \prod_{i=1}^{N} \frac{(e^{X_i'\beta})^{y_i} exp(-e^{X_i'\beta})}{y_i!}.$$

Poisson Regression Steps

In reality, it turns out to be much easier to maximize the log of this function.

So, the steps to implement this estimation are:

1. Choose a starting set of parameters
2. Use the parameters to calculate the poisson term (to the left of the product sign) for each observation
3. Sum the log of all these terms (i.e., i through N)
4. Sum all those terms to calculate the log-likelihood value
5. Find the parameter values which minimize the *negative* of the log-likelihood value

Optimization

There is no closed-form solution for the expression above, so it is necessary to find the minimized parameter values using computational optimization.

We will use the `optim()` function in R, using a custom function that we write as the objective function, and some arbitrary starting values (usually zeros are fine).

Coding the Objective Function

Our objective function is simply the Poisson log-likelihood function discussed earlier, where we input the parameter values and our data, and return the negative of the log-likelihood value.

```
log_likelihood <- function(parameters, data){

  b0 <- parameters[1] ## Take regression coefficients as inputs
  b1 <- parameters[2]
  b2 <- parameters[3]
  b3 <- parameters[4]

  data <- data %>%  ## Compute poisson value for each row
    mutate(exp_val = exp(b0 + b1*prog2 + b2*prog3 + b3*math)
    )

  data <- data %>%
    mutate(poi = (exp(-exp_val) * (exp_val^num_awards)) /
             (factorial(num_awards))
    )

  ll <- sum(log(data$poi))  ## Calculate log-likelihood value

  return(-ll) ## Return negative of log-likelihood value

}
```

Optimizing the Value

We use the `optim()` function to find the minimum and compute the standard errors using the Hessian matrix

```
poisson_mle <- optim(rep(0, 4), log_likelihood, data = award_data,
                     hessian = T, method = "BFGS")
mle_se <- poisson_mle$hessian %>%
  solve() %>%
  diag() %>%
  sqrt()

mle_results <- data.frame("Coefficient" = c("(Intercept)",
                                            "prog2", "prog3", "math"),
                          "Estimate" = poisson_mle$par,
                          "Std.Error" = mle_se)
```

Poisson Results

We can take a look at the final log-likelihood value and its associated parameter values:

```
poisson_mle$value
```

```
## [1] 182.7533
```

```
mle_results
```

```
##   Coefficient    Estimate  Std.Error
## 1 (Intercept) -5.21824026 0.65712018
## 2       prog2  1.08415504 0.35766862
## 3       prog3  0.36532344 0.44067969
## 4        math  0.06967923 0.01057573
```

Optimization Notes

If your optimiztion is not running or returning errors, you can try the following:

1. Different starting parameter values

- Even if the function does run, it is sometimes good to try a few different ones and make sure the parameter estimates are the same in case there is a local minimum

2. A different optimization method

- The default (BFGS) works well, but look at the documentation to see if the others may be more appropriate

DiD Estimators

Overview of DiD

DiD (Difference-in-Differences) is the workhorse of contemporary causal inference in applied microeconomics.

The basic idea is to compare changes in a control unit to changes in a treated unit.

The difference between those two changes (hence, Difference-in-Differences) can (sometimes) be interpreted as a *causal* effect.

Basic DiD

Let's look at the classic Card and Krueger minimum wage study.

First let's load the data:

```
ck <- read_csv("../ck_1994.csv")
```

```
## Rows: 820 Columns: 18
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): chain, state, period
## dbl (15): rest_id, southj, centralj, shore, pa1, pa2, empft, emppt, nmgrs, s...
##
## 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.
```

```
glimpse(ck)
```

```
## Rows: 820
## Columns: 18
## $ rest_id  <dbl> 46, 46, 49, 49, 506, 506, 56, 56, 61, 61, 62, 62, 445, 445, 4~
## $ chain    <chr> "BK", "BK", "KFC", "KFC", "KFC", "KFC", "Wendys", "Wendys", "~
## $ state    <chr> "PA", "PA", "PA", "PA", "PA", "PA", "PA", "PA", "PA", "PA", "~
## $ southj   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ centralj <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ shore    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ pa1      <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0~
## $ pa2      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1~
## $ period   <chr> "pre", "post", "pre", "post", "pre", "post", "pre", "post", "~
## $ empft    <dbl> 30.0, 3.5, 6.5, 0.0, 3.0, 3.0, 20.0, 0.0, 6.0, 28.0, 0.0, NA,~
## $ emppt    <dbl> 15.0, 35.0, 6.5, 15.0, 7.0, 7.0, 20.0, 36.0, 26.0, 3.0, 31.0,~
## $ nmgrs    <dbl> 3, 3, 4, 4, 2, 4, 4, 2, 5, 6, 5, NA, 3, 5, 5, 6, 5, 2, 2, 2, ~
## $ stwage   <dbl> NA, 4.30, NA, 4.45, NA, 5.00, 5.00, 5.25, 5.50, 4.75, 5.00, N~
## $ pentree  <dbl> 16.5, 16.5, 13.0, 13.0, 10.0, 11.0, 12.0, 12.0, 12.0, 12.0, 1~
## $ fte      <dbl> 40.50, 24.00, 13.75, 11.50, 8.50, 10.50, 34.00, 20.00, 24.00,~
```

Basic DiD

Our outcome of interest is `fte` (full-time employment).

Our policy change occurs in NJ in period two (`state == NJ` and `period == post`).

Let's compare the mean difference in NJ (treated) to the mean difference in PA (control):

```
treated_pre <- ck |>
  filter(state == "NJ" & period == "pre") |>
  summarise(mean_fte = mean(fte, na.rm = T)) |>
  pull(mean_fte)

treated_post <- ck |>
  filter(state == "NJ" & period == "post") |>
  summarise(mean_fte = mean(fte, na.rm = T)) |>
  pull(mean_fte)
```

Basic DiD

```r
control_pre <- ck |>
  filter(state == "PA" & period == "pre") |>
  summarise(mean_fte = mean(fte, na.rm = T)) |>
  pull(mean_fte)

control_post <- ck |>
  filter(state == "PA" & period == "post") |>
  summarise(mean_fte = mean(fte, na.rm = T)) |>
  pull(mean_fte)

diff_treated <- treated_post - treated_pre
diff_control <- control_post - control_pre

did <- diff_treated - diff_control

print(did)
```

```
## [1] 2.753606
```

DiD in regression form

We will never estimate a DiD like this. We will instead use a regression.

One common method to estimate a DiD:

$$y = \beta_1 \mathit{Treated}_i + \beta_2 \mathit{Post}_t + \beta_3(\mathit{Treated}_i * \mathit{Post}_t) + \epsilon_{it}$$

where *Treated* is 1 if the unit is in the treatment group, *Post* is 1 if the time period is after the intevention, and the β_3 term gives the DiD estimate.

DiD in regression form

Let's estimate it this way:

```
did1 <- feols(fte ~ treated + post + treated * post,
              data = ck)
```

```
## NOTE: 26 observations removed because of NA values (LHS: 26).
```

```
summary(did1)
```

```
## OLS estimation, Dep. Var.: fte
## Observations: 794
## Standard-errors: IID
##              Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)  23.33117    1.07187 21.76679 < 2.2e-16 ***
## treated      -2.89176    1.19352 -2.42288  0.015622 *
## post         -2.16558    1.51585 -1.42862  0.153507
## treated:post  2.75361    1.68841  1.63089  0.103313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 9.3819   Adj. R2: 0.003632
```

DiD with Fixed Effects

Much more common (especially in social sciences) is to estimate the DiD with the following regression:

$$y = \beta_1 D_{it} + \gamma_i + \tau_(t) + \epsilon_{it}$$

where D_{it} equals 1 if unit i is treated at time t, γ_i is the unit fixed effect, and τ_t is the time fixed effect.

This is called the Two-Way Fixed Effects (TWFE) estimator and is the most important regression model in applied microeconomics.

DiD with Fixed Effects

Let's first create our D_{it} variable and call it `treat`, the estimate the regression.

```
ck <- ck |>
  mutate(treat = case_when(state == "NJ" & period == "post" ~ 1,
                           TRUE ~ 0))

twfe <- feols(fte ~ treat |
                state + time,
              data = ck)
```

```
## NOTE: 26 observations removed because of NA values (LHS: 26).
```

```
summary(twfe)
```

```
## OLS estimation, Dep. Var.: fte
## Observations: 794
## Fixed-effects: state: 2,  time: 2
## Standard-errors: Clustered (state)
##       Estimate         Std. Error        t value               Pr(>|t|)
## treat  2.75361 0.0000000000000405 68016987722947 0.00000000000000093597 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 9.3819     Adj. R2: 0.003632
##                Within R2: 0.003356
```

Advanced DiD Models

In economics, we will often have multiple treated units and control units, often studying some state-level policy changes such as the minimum wage.

Econometric problems can arise when using the TWFE model, especially when the roll out of these treatments is not simultaneous.

In these cases, we need to rely on alternative ways of estimating DiD models, which also allow us to estimate the DiD for different time periods (e.g., 3 years after treatment, 2 years before, etc.).

While there are important differences between the estimators, they all share the same similarity in that they are more careful about choosing which units treated units are being compared to and rule out *forbidden comparisons:* comparing newly treated units to already treated units.

Advanced DiD Model 1: LP-DiD

Similar (often identical) to the stacked DiD approach in Cengiz et al. (2019).

To do LP-DiD estimates in R, we first need to download the package.

Since this package is hosted on GitHub, it is a slightly different code:

```
install.packages("devtools")
devtools::install_github("alexCardazzi/lpdid")
```

```
library(lpdid)
```

Advanced DiD Model 1: CS-DiD

We will also need to grab the `did` package to do CS-DiD.

```
install.packages("did")
```

```
library(did)
```

A Note on Data Shape

Depending on which package you are using, these functions may require you to have the data, and especially the treatment variables, organized in certain ways.

In general, the data will all need to be in panel format, like that below:

```
panel_ex = tibble(
  state = c("NC", "NC", "MA", "MA", "CA", "CA"),
  year = c(2012, 2013, 2012, 2013, 2012, 2013),
  y = c(123, 561, 95, 21, 123, 313),
  treated = c(0, 0, 0, 0, 1, 1),
  post = c(0, 1, 0, 1, 0, 1),
  treat = c(0, 0, 0, 0, 0, 1)
)
```

A Note on Data Shape

```
knitr::kable(panel_ex, format = "markdown")
```

state	year	y	treated	post	treat
NC	2012	123	0	0	0
NC	2013	561	0	1	0
MA	2012	95	0	0	0
MA	2013	21	0	1	0
CA	2012	123	1	0	0
CA	2013	313	1	1	1

A Note on Data Shape

It is also useful (and necessary) to have variables which denote the treatment cohort (i.e., the year in which a unit is treated) (for CS-DID), as well as a lagged treatment variable which equals 1 only in the initial year of treatment.

```
panel_ex = tibble(
  state = c("NC", "NC", "MA", "MA", "CA", "CA"),
  year = c(2012, 2013, 2012, 2013, 2012, 2013),
  y = c(123, 561, 95, 21, 123, 313),
  treated = c(0, 0, 0, 0, 1, 1),
  post = c(0, 1, 0, 1, 0, 1),
  treat = c(0, 0, 0, 0, 0, 1),
  cohort = c(0, 0, 0, 0, 2013, 2013)
)
```

A Note on Data Shape

```
knitr::kable(panel_ex, format = "markdown")
```

state	year	y	treated	post	treat	cohort
NC	2012	123	0	0	0	0
NC	2013	561	0	1	0	0
MA	2012	95	0	0	0	0
MA	2013	21	0	1	0	0
CA	2012	123	1	0	0	2013
CA	2013	313	1	1	1	2013

Generating a data set

Let's first generate a test data set so we can know for sure if these estimators are working. Let's say there are 50 states and 15 years of data. Let's say states number 1 to 15 are group 1, and states number 16 - 30 are group 2. Assume states 31 - 50 are group 0 (never treated). Assume groups 1 and 2 are treated at year 9.

```
state <- 1:50
year <- 1:15

sim_df <- expand_grid(state, year)
```

Generating a data set

Let's create some identifying variables to help us with this estimation.

```
sim_df <- sim_df |>
  mutate(group = case_when(state <= 15 ~ 1,
                           state >= 16 & state <= 30 ~ 2,
                           state >= 31 ~ 0))

sim_df <- sim_df |>
  mutate(treated = case_when(group == 0 ~ 0,
                             TRUE ~ 1))

sim_df <- sim_df |>
  mutate(cohort = case_when(group == 0 ~ 0,
                            group == 1 ~ 9,
                            group == 2 ~ 9))

sim_df <- sim_df |>
  mutate(treat = case_when(cohort == 0 ~ 0,
                           year >= cohort ~ 1,
                           TRUE ~ 0))

sim_df <- sim_df |>
  mutate(treatshock = case_when(cohort == 0 ~ 0,
                           year == cohort ~ 1,
                           TRUE ~ 0))
```

Generating a data set

Now let's generate the outcome variable.

Let's say the true treatment effect for group 1 is 0.2 and is 0.1 for group 2 (heterogeneous treatment effects). Both groups are treated in year 9.

Then, the outcome is equal to:

$$y_{st} = \beta_s \, Treat_{st} + \mu s + \tau_t + \epsilon_{st}$$

where we draw the μ_s, τ_t, and ϵ_{st} from a normal distribution.

Remember to set the seed before generating random numbers!

Generating a data set

```
set.seed(94578161)

mu <- tibble(state = 1:50) |>
  mutate(mu_s = rnorm(50, mean = 0, sd = 0.1))


tau <- tibble(year = 1:15) |>
  mutate(tau_t = rnorm(15, mean = 0, sd = 0.05))

sim_df <- sim_df |>
  left_join(mu, by = "state") |>
  left_join(tau, by = "year")

sim_df <- sim_df |>
  mutate(eps_st = rnorm(nrow(sim_df), mean = 0, sd = 0.05))

sim_df <- sim_df |>
  mutate(beta = case_when(group == 1 ~ 0.2,
                          group == 2 ~ 0.1,
                          group == 0 ~ 0
                          ))

sim_df <- sim_df |>
  mutate(y = beta * treat + mu_s + tau_t + eps_st)
```

Estimating the Treatment Effect

Let's now test our models and see if we can get the right treatment effect.

First, a standard TWFE model:

```
stwfe <- feols(y ~ treat |
                 state + year,
               data = sim_df)
summary(stwfe)
```

```
## OLS estimation, Dep. Var.: y
## Observations: 750
## Fixed-effects: state: 50,  year: 15
## Standard-errors: Clustered (state)
##       Estimate Std. Error t value  Pr(>|t|)
## treat 0.147344   0.011904 12.3777 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.05219     Adj. R2: 0.858417
##                 Within R2: 0.322542
```

Estimating DTWFE

Estimating DTWFE is the most complicated syntax-wise. We need to create a dummy variable which equals 1 at each time of treatment, and 0 otherwise (or if a control unit).

Let's first load another helper function.

```
install.packages("fastDummies")
```

```
library(fastDummies)
```

```
## Thank you for using fastDummies!

## To acknowledge our work, please cite the package:

## Kaplan, J. & Schlegel, B. (2023). fastDummies: Fast Creation of Dummy (Binar
```

Estimating DTWFE

```
sim_df <- sim_df |>
  mutate(cohort_t = year - cohort,
         cohort_t = case_when(group == 0 ~ 0,
                              TRUE ~ cohort_t))
sim_df <- dummy_cols(sim_df, select_columns = "cohort_t")

sim_df <- sim_df |>
  rename_with(~gsub("cohort_t_-", "cohort_t_m", .x))

sim_df <- sim_df |>
  mutate(cohort_t_0 = case_when(group == 0 ~ 0,
                                TRUE ~ cohort_t_0))

dtwfe_est <- feols(y ~ cohort_t_m3 + cohort_t_m2 + cohort_t_m1 +
                     cohort_t_0 + cohort_t_1 + cohort_t_2 + cohort_t_3 + cohort
                     state + year,
                   data = sim_df)
```

Estimating DTWFE

Let's look at the results!

```
summary(dtwfe_est)
```

```
## OLS estimation, Dep. Var.: y
## Observations: 750
## Fixed-effects: state: 50,  year: 15
## Standard-errors: Clustered (state)
##              Estimate Std. Error   t value        Pr(>|t|)
## cohort_t_m3 -0.070145   0.017320 -4.049906 0.0001820784582 ***
## cohort_t_m2 -0.047009   0.015648 -3.004070 0.0041884439649 **
## cohort_t_m1 -0.007591   0.017376 -0.436880 0.6641179938058
## cohort_t_0   0.109426   0.014645  7.471964 0.0000000012363 ***
## cohort_t_1   0.122820   0.017274  7.110000 0.0000000044817 ***
## cohort_t_2   0.104416   0.015850  6.587851 0.0000000288691 ***
## cohort_t_3   0.093046   0.018089  5.143663 0.0000047178368 ***
## cohort_t_4   0.086072   0.017550  4.904425 0.0000107285070 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.056606     Adj. R2: 0.831723
##                  Within R2: 0.203039
```

Estimating DTWFE

We then need to re-base the event-study estimates to be relative to t minus 1.

```
m1_eff <- dtwfe_est$coeftable$Estimate[3]

dtwfe_sim <- tidy(dtwfe_est) |>
  select(term, estimate) |>
  mutate(t_eff = str_remove(term, "cohort_t_"),
         t_eff = as.numeric(str_replace(t_eff, "m", "-"))) |>
  mutate(estimate = estimate - m1_eff)
```

Estimating LP-DiD

Implementing these new DiD estimators is easy to do once installed, but be sure to look at the documentation to understand what exactly each argument does/needs.

Let's use LP-DiD and estimate an event study:

```
lpdid1 <- lpdid(sim_df,
                window = c(-3, 4),
                y = "y",
                unit_index = "state",
                time_index = "year",
                treat_status = "treatshock"
                )
```

```
##      Estimate Std. Error   t value           Pr(>|t|)
## 1 -0.06255338 0.02218198 -2.820009 0.0024011158576146
## 2 -0.03941769 0.01989465 -1.981321 0.0237776223787132
## 3  0.00000000 0.00000000       NaN                NaN
## 4  0.11701790 0.02108282  5.550392 0.0000000142514822
## 5  0.13041095 0.02156598  6.047067 0.0000000007375324
## 6  0.11200770 0.01969666  5.686635 0.0000000064783456
## 7  0.10063762 0.02496816  4.030637 0.0000278129117691
## 8  0.09366364 0.02662347  3.518086 0.0002173359945015
```

CS-DiD

Now let's estimate the model using the CS-DiD estimator.

We need to first estimate the group-time effects, then call a different function to aggregate those effects into an event study framework.

```
csdid1 <- att_gt(yname = "y",
                 tname = "year",
                 idname = "state",
                 gname = "cohort",
                 control_group = "nevertreated",
                 allow_unbalanced_panel = T,
                 clustervars = "state",
                 base_period = "universal",
                 data = sim_df)
```

CS-DiD

```r
csdid_est <- aggte(csdid1, type = "dynamic",
                   min_e = -3,
                   max_e = 4,
                   na.rm = T)

csdid_est
```

```
##
## Call:
## aggte(MP = csdid1, type = "dynamic", min_e = -3, max_e = 4, na.rm = T)
##
## Reference: Callaway, Brantly and Pedro H.C. Sant'Anna.  "Difference-in-Differences with Multiple Time P
##
##
## Overall summary of ATT's based on event-study/dynamic aggregation:
##     ATT    Std. Error     [ 95%  Conf. Int.]
##  0.1107        0.0184     0.0747      0.1468 *
##
##
## Dynamic Effects:
##  Event time Estimate Std. Error [95% Simult.  Conf. Band]
##          -3  -0.0626     0.0208       -0.1141     -0.0110 *
##          -2  -0.0394     0.0184       -0.0849      0.0061
##          -1   0.0000         NA            NA          NA
##           0   0.1170     0.0217        0.0634      0.1707 *
##           1   0.1304     0.0221        0.0756      0.1852 *
##           2   0.1120     0.0190        0.0651      0.1590 *
##           3   0.1006     0.0258        0.0367      0.1645 *
##           4   0.0937     0.0257        0.0300      0.1573 *
## ---
## Signif. codes: `*' confidence band does not cover 0
##
## Control Group:  Never Treated,  Anticipation Periods:  0
## Estimation Method:  Doubly Robust
```