Lab 4

Ray Caraher

2025-04-14

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/Teachi
data_directory <- file.path(base_directory, 'Data')
results_directory <- file.path(base_directory, 'Results')
```

Synthetic Controls

Synthetic Controls, Matching, and DiD

Today we are going to take a closer look at the difference between a DiD and synthetic control model.

We are going to use California's 1988 minimum wage increase as an example.

Getting data

Let's read in the data and take a look.

```
empwage <- read_dta(file.path(data_directory, "emp_wage_data.dta"))
glimpse(empwage)
```

```
## Rows: 4,284
## Columns: 42
## $ statenum          <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## $ quarterdate       <dbl> 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
## $ year              <dbl> 1980, 1980, 1980, 1980, 1981, 1981, 1981, 1981,
## $ qtr               <dbl> 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4,
## $ overall_emp       <dbl> 0.6237059, 0.6227944, 0.6029442, 0.6123795, 0.60
## $ age_group_sh1     <dbl> 0.13001385, 0.12324983, 0.12576827, 0.12388597,
## $ age_group_sh2     <dbl> 0.2463664, 0.2466077, 0.2594528, 0.2656242, 0.26
## $ age_group_sh3     <dbl> 0.2180452, 0.2252479, 0.2170429, 0.2099088, 0.22
## $ age_group_sh4     <dbl> 0.1650004, 0.1778168, 0.1805827, 0.1773866, 0.16
## $ age_group_sh5     <dbl> 0.1737192, 0.1657912, 0.1568459, 0.1503734, 0.15
## $ age_group_sh6     <dbl> 0.06685495, 0.06128659, 0.06030744, 0.07282100,
## $ division          <dbl> 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
## $ region            <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
## $ quarter           <dbl> 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4,
## $ race_share1       <dbl> 0.7237110, 0.7279011, 0.7314317, 0.7217314, 0.74
## $ race_share2       <dbl> 0.2733353, 0.2655723, 0.2643476, 0.2731366, 0.25
## $ race_share3       <dbl> 0.0029536844, 0.0065265722, 0.0042207381, 0.0051
## $ hispanic_share    <dbl> 0.0061911955, 0.0053420719, 0.0013335996, 0.0052
## $ married_share     <dbl> 0.6359439, 0.6434138, 0.6375374, 0.6302283, 0.62
```

Data

We can see that this data contains information at the state-quarter level on:

- demographic population shares (by age group, race, and ethnicity)
- overall and teen wages and employment
- the state-level minimum wage

CA's minimum wage

Let's take a look at how California's minimum wage changed overtime:

We can use the count command to do so.

```
empwage |>
  filter(stateabb == "CA") |>
  count(year, MW) |>
  filter(n < 4)
```

```
## # A tibble: 10 x 3
##     year    MW     n
##    <dbl> <dbl> <int>
##  1  1988  3.35     2
##  2  1988  4.25     2
##  3  1996  4.25     3
##  4  1996  4.75     1
##  5  1997  4.84     1
##  6  1997  5        1
##  7  1997  5.05     1
##  8  1997  5.15     1
##  9  1998  5.36     1
## 10  1998  5.75     3
```

CA's minimum wage

We can see that CA increased it's minimum wage several times starting in 1988.

In this exercise, we will focus on this initial increase between 1988 and 1992, when there was a federal minimum wage imposed.

```
empwage <- empwage %>%
  mutate(yr_qtr =  year + (qtr / 10))

empwage <- empwage %>%
  filter(yr_qtr >= 1982.1 & yr_qtr <= 1990.1)

empwage <- empwage %>%
  mutate(overall_logemp = log(overall_emp),
         teen_logemp = log(teen_emp))
```

Synthetic control design

Synthetic control research design excels as an alternative to DiD when there is *one* treated unit, *many* control units, and a reasonably long time span.

The synthetic control method constructs a single *counter-factual* control unit comprised on a weighted average of all other control units.

The identifying assumption is that this weighted-average-of-controls counter-factual is a good representation of what would have happened in the treated unit *had the intervention not occurred*.

Synthetic control design and CA's minimum wage

Let's now create a good setup to use the synthetic control method to estimate the effect of the minimum wage on wage and employment outcomes.

In other words, let's drop all other units that recieve a "treatment" (i.e., an increase in the MW) between 1982q1 and 1992q1.

```
empwage <- empwage %>%
  group_by(stateabb) %>%
  arrange(yr_qtr) %>%
  mutate(mw_change = MW - lag(MW))

empwage <- empwage %>%
  group_by(stateabb) %>%
  arrange(yr_qtr) %>%
  mutate(control = case_when(sum(mw_change, na.rm = T) == 0 ~ 1,
                             TRUE ~ 0)) %>%
  ungroup()

empwage_ca <- empwage %>%
  filter(control == 1 | stateabb == "CA")
```

Synthetic control design and CA's minimum wage

There appear to be about 35 states which did not increase the MW during this period. This will be our pool of control units ("donors").

```
count(empwage_ca, stateabb, control) |>
  arrange(control)
```

```
## # A tibble: 36 x 3
##    stateabb control     n
##    <chr>      <dbl> <int>
##  1 CA             0    33
##  2 AK             1    33
##  3 AL             1    33
##  4 AR             1    33
##  5 AZ             1    33
##  6 CO             1    33
##  7 DE             1    33
##  8 FL             1    33
##  9 GA             1    33
## 10 ID             1    33
## # i 26 more rows
```

TWFE estimate

Let's first estimate the basic TWFE DiD estimates so we can later compare to the synthetic controls.

```
empwage_ca <- empwage_ca %>%
  mutate(post = case_when(yr_qtr >= 1988.3 ~ 1,
                          TRUE ~ 0),
         treated = case_when(stateabb == "CA" ~ 1,
                             TRUE ~ 0),
         treat = treated * post)

did1_lteenemp <- feols(teen_logemp ~ treat |
                         stateabb + yr_qtr,
                       cluster = "stateabb",
                       data = empwage_ca)

did1_lteenwage <- feols(teen_logwage ~ treat |
                         stateabb + yr_qtr,
                       cluster = "stateabb",
                       data = empwage_ca)
```

TWFE estimate

Let's first estimate the basic TWFE DiD estimates so we can later compare to the synthetic controls.

```
did1_ltotemp <- feols(overall_logemp ~ treat |
                        stateabb + yr_qtr,
                      cluster = "stateabb",
                      data = empwage_ca)

did1_ltotwage <- feols(overall_logwage ~ treat |
                         stateabb + yr_qtr,
                       cluster = "stateabb",
                       data = empwage_ca)
```

TWFE estimate

Let's first estimate the basic TWFE DiD estimates so we can later compare to the synthetic controls.

```
did_tab <- bind_rows(as_tibble(did1_lteenemp$coeftable),
                     as_tibble(did1_lteenwage$coeftable),
                     as_tibble(did1_ltotemp$coeftable),
                     as_tibble(did1_ltotwage$coeftable))

did_tab <- did_tab |>
  mutate(Outcome = c("Teen emp.", "Teen wage", "Overall emp.", "Overall wage"))

did_tab <- did_tab |>
  select(Outcome, everything())
```

TWFE estimate

Let's first estimate the basic TWFE DiD estimates so we can later compare to the synthetic controls.

```
did_tab
```

```
## # A tibble: 4 x 5
##   Outcome       Estimate `Std. Error` `t value` `Pr(>|t|)`
##   <chr>            <dbl>        <dbl>     <dbl>      <dbl>
## 1 Teen emp.     0.0241        0.0107      2.26    2.99e- 2
## 2 Teen wage     0.0932        0.0112      8.29    8.92e-10
## 3 Overall emp. -0.000698      0.00324    -0.215   8.31e- 1
## 4 Overall wage  0.0248        0.00896     2.77    8.98e- 3
```

Synthetic Control Estimates in R

Now, let's estimate a synthetic control in R.

We will use a combination of averaged pre-treatment outcomes as well as covariates for this estimate.

The best implementation of synthetic controls in R (so far) is from the `tidysynth` package.

Let's install it take a look at how it works:

```
#install.packages("tidysynth")

library(tidysynth)
```

The tidysynth package

The `tidysynth` package works by iteratively, in that we first "initialize" our synthetic control object, then use its functions to add control variables, before ultimately using it to calculate the weights for the synthetic control.

Let's go ahead and create the initial object:

```
time_vars <- distinct(empwage_ca, yr_qtr) |>
  mutate(time_var = 1:n())

empwage_ca <- empwage_ca |>
  left_join(time_vars)
```

```
## Joining with `by = join_by(yr_qtr)`
```

```
synth <- empwage_ca |>
  synthetic_control(outcome = teen_logwage, # outcome
                    unit = stateabb, # unit index in the panel data
                    time = time_var, # time index in the panel data
                    i_unit = "CA", # unit where the intervention occurred
                    i_time = 27, # time period when the intervention occurred
                    generate_placebos = F
                    )
```

The tidysynth package

The `tidysynth` package stores the data as a complex set of nested lists. Let's look at what this is like:

```
synth$.outcome[[1]]
```

```
## # A tibble: 27 x 2
##    time_unit    CA
##        <int> <dbl>
##  1         1  1.35
##  2         2  1.41
##  3         3  1.36
##  4         4  1.35
##  5         5  1.38
##  6         6  1.36
##  7         7  1.39
##  8         8  1.36
##  9         9  1.44
## 10        10  1.40
## # i 17 more rows
```

The tidysynth package

The `tidysynth` package stores the data as a complex set of nested lists. Let's look at what this is like:

```r
synth$.outcome[[2]]
```

```
## # A tibble: 27 x 36
##    time_unit    AK    AL    AR    AZ    CO    DE    FL    GA    ID    IL
##        <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
##  1         1  1.55  1.22  1.28  1.25  1.25  1.24  1.30  1.24  1.23  1.30  1.
##  2         2  1.63  1.19  1.23  1.27  1.36  1.16  1.27  1.24  1.28  1.24  1.
##  3         3  1.63  1.27  1.23  1.19  1.30  1.22  1.27  1.25  1.22  1.30  1.
##  4         4  1.54  1.32  1.29  1.06  1.25  1.26  1.28  1.31  1.24  1.24  1.
##  5         5  1.67  1.15  1.25  1.29  1.20  1.30  1.28  1.21  1.20  1.19  1.
##  6         6  1.72  1.52  1.18  1.34  1.31  1.22  1.29  1.27  1.13  1.29  1.
##  7         7  1.72  1.31  1.31  1.34  1.32  1.15  1.25  1.27  1.18  1.36  1.
##  8         8  1.81  1.31  1.22  1.36  1.30  1.23  1.30  1.33  1.25  1.29  1.
##  9         9  1.68  1.27  1.13  1.28  1.27  1.29  1.30  1.33  1.18  1.25  1.
## 10        10  1.70  1.26  1.36  1.24  1.31  1.25  1.30  1.36  1.28  1.29  1.
## # i 17 more rows
## # i 24 more variables: KS <dbl>, KY <dbl>, LA <dbl>, MD <dbl>, MI <dbl>,
## #   MO <dbl>, MS <dbl>, MT <dbl>, NC <dbl>, NE <dbl>, NJ <dbl>, NM <dbl>,
## #   NV <dbl>, NY <dbl>, OH <dbl>, OK <dbl>, SC <dbl>, SD <dbl>, TN <dbl>,
## #   TX <dbl>, UT <dbl>, VA <dbl>, WV <dbl>, WY <dbl>
```

Adding control variables

Using tidysynth, we add control variables using the `generate_predictor()` function in a pipe, building off the initial object.

We specify the time window, the type of aggregation we want to, and the variables we want to do it to.

Let's first do the pre-treatment mean of the outcome variable:

```
synth <- synth |>
  generate_predictor(time_window = 1:26,
                     mean_teen_logwage = mean(teen_logwage, na.rm = T)
                     )
```

Adding control variables

Let's next do the pre-treatment means of the industry shares and demographic shares variables.

Let's first do this for all the age-share variables.

```
synth <- synth |>
  generate_predictor(time_window = 1:26,
                     mean_age_group_sh1 = mean(age_group_sh1, na.rm = T),
                     mean_age_group_sh2 = mean(age_group_sh2, na.rm = T),
                     mean_age_group_sh3 = mean(age_group_sh3, na.rm = T),
                     mean_age_group_sh4 = mean(age_group_sh4, na.rm = T),
                     mean_age_group_sh5 = mean(age_group_sh5, na.rm = T),
                     mean_age_group_sh6 = mean(age_group_sh6, na.rm = T)
                     )
```

Adding control variables

Now, the race, ethnicity, gender, and high-school degree shares:

```
synth <- synth |>
  generate_predictor(time_window = 1:26,
                     mean_race_share1 = mean(race_share1, na.rm = T),
                     mean_race_share2 = mean(race_share2, na.rm = T),
                     mean_race_share3 = mean(race_share3, na.rm = T),
                     mean_hispanic_share = mean(hispanic_share, na.rm = T),
                     mean_gender_share = mean(gender_share, na.rm = T),
                     mean_hsl_share = mean(hsl_share, na.rm = T)
                     )
```

Adding control variables

Now, the industry shares:

```
synth <- synth |>
  generate_predictor(time_window = 1:26,
                     mean_emp_sh_ind1 = mean(emp_sh_ind1, na.rm = T),
                     mean_emp_sh_ind2 = mean(emp_sh_ind2, na.rm = T),
                     mean_emp_sh_ind3 = mean(emp_sh_ind3, na.rm = T),
                     mean_emp_sh_ind4 = mean(emp_sh_ind4, na.rm = T),
                     mean_emp_sh_ind5 = mean(emp_sh_ind5, na.rm = T),
                     mean_emp_sh_ind6 = mean(emp_sh_ind6, na.rm = T),
                     mean_emp_sh_ind7 = mean(emp_sh_ind7, na.rm = T),
                     mean_emp_sh_ind8 = mean(emp_sh_ind8, na.rm = T),
                     mean_emp_sh_ind9 = mean(emp_sh_ind9, na.rm = T)
                     )
```

Adding control variables

Let's take a look at how `tidysynth` stores these predictor variables:

```
synth$.predictors[[1]]
```

```
## # A tibble: 22 x 2
##    variable                CA
##    <chr>                <dbl>
##  1 mean_teen_logwage   1.42
##  2 mean_age_group_sh1  0.0918
##  3 mean_age_group_sh2  0.286
##  4 mean_age_group_sh3  0.255
##  5 mean_age_group_sh4  0.170
##  6 mean_age_group_sh5  0.134
##  7 mean_age_group_sh6  0.0635
##  8 mean_gender_share   0.490
##  9 mean_hispanic_share 0.196
## 10 mean_hsl_share      0.550
## # i 12 more rows
```

Adding control variables

Let's take a look at how `tidysynth` stores these predictor variables:

```
synth$.predictors[[2]]
```

```
## # A tibble: 22 x 36
##    variable          AK      AL      AR      AZ      CO      DE      FL      GA
##    <chr>          <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <
##  1 mean_teen_l~ 1.67   1.30    1.27    1.33    1.33    1.31    1.34    1.35    1.
##  2 mean_age_gr~ 0.0879 0.107   0.108   0.0939  0.0875  0.0951  0.0928  0.102   0.
##  3 mean_age_gr~ 0.307  0.260   0.261   0.299   0.274   0.279   0.261   0.270   0.
##  4 mean_age_gr~ 0.294  0.232   0.221   0.242   0.280   0.237   0.231   0.247   0.
##  5 mean_age_gr~ 0.174  0.174   0.178   0.162   0.170   0.167   0.170   0.172   0.
##  6 mean_age_gr~ 0.103  0.154   0.158   0.132   0.132   0.149   0.158   0.144   0.
##  7 mean_age_gr~ 0.0338 0.0725  0.0735  0.0711  0.0564  0.0723  0.0863  0.0657  0.
##  8 mean_gender~ 0.497  0.475   0.477   0.489   0.492   0.478   0.478   0.476   0.
##  9 mean_hispan~ 0.0200 0.00324 0.00639 0.164   0.0988  0.0146  0.110   0.0112  0.
## 10 mean_hsl_sh~ 0.586  0.723   0.740   0.586   0.529   0.640   0.648   0.681   0.
## # i 12 more rows
## # i 26 more variables: IL <dbl>, IN <dbl>, KS <dbl>, KY <dbl>, LA <dbl>,
## #   MD <dbl>, MI <dbl>, MO <dbl>, MS <dbl>, MT <dbl>, NC <dbl>, NE <dbl>,
## #   NJ <dbl>, NM <dbl>, NV <dbl>, NY <dbl>, OH <dbl>, OK <dbl>, SC <dbl>,
## #   SD <dbl>, TN <dbl>, TX <dbl>, UT <dbl>, VA <dbl>, WV <dbl>, WY <dbl>
```

Adding control variables

Let's double check that `tidysynth` is correctly returning the right mean values:

```
empwage_ca |>
  filter(time_var <= 26) |>
  filter(stateabb == "AL") |>
  summarise(mean(teen_logwage, na.rm = T))
```

```
## # A tibble: 1 x 1
##   `mean(teen_logwage, na.rm = T)`
##                             <dbl>
## 1                            1.30
```

```
empwage_ca |>
  filter(time_var <= 26) |>
  filter(stateabb == "AZ") |>
  summarise(mean(age_group_sh1, na.rm = T))
```

```
## # A tibble: 1 x 1
##   `mean(age_group_sh1, na.rm = T)`
##                              <dbl>
## 1                           0.0939
```

Estimate the synthetic control

To actually calculate the weights and synthetic control, we use the `generate_weights()` function followed by the `generate_control()` function. We first need to specify the optimization window.

For now, we will use the whole pre-treatment period, but if you want to leave out some periods before treatment as a validation exercise, you can have an optimization window that ends early.

```
teenwage_out <- synth |>
  generate_weights(optimization_window = 1:26) |>
  generate_control()
```

Plotting the synthetic control

The `tidysynth` has a bunch of nice, built-in packages to plot and work with the synthetic control results.

You can look up the documentation for these.

But for now, we will do it manually so we know what we are looking at.

Let's first grab the synthetic control and treated outcomes:

```
teenwage_synth <- as_tibble(teenwage_out$.synthetic_control[[1]])

teenwage_synth <- teenwage_synth |>
  left_join(time_vars, by = c("time_unit" = "time_var"))

teenwage_synth
```

```
## # A tibble: 33 x 4
##    time_unit real_y synth_y yr_qtr
##        <int>  <dbl>   <dbl>  <dbl>
##  1         1   1.35    1.35  1982.
##  2         2   1.41    1.38  1982.
##  3         3   1.36    1.35  1982.
##  4         4   1.35    1.36  1982.
##  5         5   1.38    1.36  1983.
```

Plotting the synthetic control

Now, let's plot it!

```
teenwage_synth_l <- teenwage_synth |>
  pivot_longer(cols = c("real_y", "synth_y"),
               names_to = "type", values_to = "teen_logwage")

p1 <- ggplot(teenwage_synth_l) +
  geom_line(aes(x = yr_qtr, y = teen_logwage, color = type))
```

Plotting the synthetic control

```r
print(p1)
```

Synthetic control vs. unweighted controls

Overall, not a bad fit!

Let's compare the synthetic control fit to the simple, unweighted outcome for the control states

```
control_teenwage <- empwage_ca |>
  filter(control == 1) |>
  group_by(yr_qtr) |>
  summarise(unwght_y = mean(teen_logwage, na.rm = T)) |>
  ungroup()

teenwage_synth <- teenwage_synth |>
  left_join(control_teenwage)
```

```
## Joining with `by = join_by(yr_qtr)`
```

```
teenwage_synth_l <- teenwage_synth |>
  pivot_longer(cols = c("real_y", "synth_y", "unwght_y"),
               names_to = "type", values_to = "teen_logwage")

p1 <- ggplot(teenwage_synth_l) +
  geom_line(aes(x = yr_qtr, y = teen_logwage, color = type))
```

Synthetic control vs. unweighted controls

```
print(p1)
```

Comparing to the DiD

How do we actually compare the synthetic control estimates to the DiD estimates?

One way is to find the average *post-treatment difference* between the treated state and the synthetic control.

```
teenwage_synth <- teenwage_synth |>
  mutate(diff = real_y - synth_y)

teenwage_synth |>
  filter(yr_qtr >= 1988.3) |>
  summarise(treat_effect = mean(diff))
```

```
## # A tibble: 1 x 1
##   treat_effect
##          <dbl>
## 1       0.0374
```

Comparing to the DiD

Compared to our TWFE estimate of `0.09`, the synthetic control estimate is a much more modest increase of about `0.04`.

Who gets the most weight?

It is easy to get the weights from the synthetic control model:

```
state_weights <- teenwage_out$.unit_weights[[1]]

state_weights <- state_weights |>
  rename(state = unit)

state_weights
```

```
## # A tibble: 35 x 2
##    state      weight
##    <chr>       <dbl>
##  1 AK    0.178
##  2 AL    0.000000109
##  3 AR    0.000000116
##  4 AZ    0.000000106
##  5 CO    0.0485
##  6 DE    0.000000180
##  7 FL    0.000000885
##  8 GA    0.000000336
##  9 ID    0.0302
## 10 IL    0.0129
## # i 25 more rows
```

Mapping the weights

In R, there are many ways to generate a map.

The easiest way to map the US is with the `usmap` function.

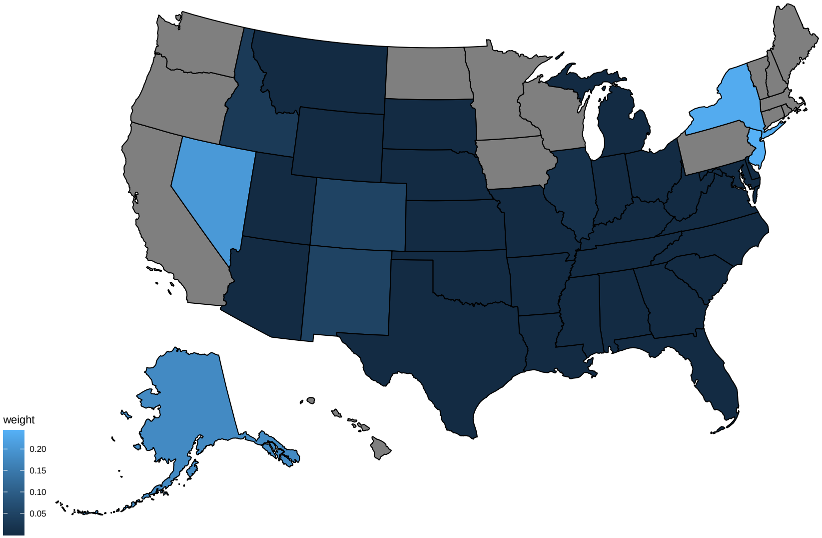
Look at the documentation to make a prettier looking map for the write-up.

```
#install.packages(usmap)
library(usmap)

m1 <- plot_usmap(data = state_weights, values = "weight")
```

Mapping the weights

```
print(m1)
```



Constructing Placebos

How do we conduct inference in the case of synthetic controls?

One common way is to leverage *placebo treatments*.

In this exercise, we run the synthetic control **as if** each control state in our sample is the one being treated.

If the magnitude of our treatment in our actually treated state is much larger than our placebo treatments (usually trimming the placebos to get rid of really bad fits), then we can consider our effect to be not due to random variation, and therefore statistically significant.

Constructing Placebos

We can do this using our code above, but making sure to loop over a list of all states!

For now, let's do this with CA and just a few other states.

Constructing Placebos

```r
state_list <- distinct(empwage_ca, stateabb) |>
  pull(stateabb)

state_list <- state_list[1:5]

placebo_synth_tab <- tibble()
```

Constructing Placebos

```r
for (i in seq_along(state_list)) {

  state <- state_list[i]

  synth <- empwage_ca |>
  synthetic_control(outcome = teen_logwage, # outcome
                    unit = stateabb, # unit index in the panel data
                    time = time_var, # time index in the panel data
                    i_unit = state, # unit where the intervention occurred
                    i_time = 27, # time period when the intervention occurred
                    generate_placebos = F
                    ) |>
      generate_predictor(time_window = 1:26,
                     mean_teen_logwage = mean(teen_logwage, na.rm = T),
                     mean_age_group_sh1 = mean(age_group_sh1, na.rm = T),
                     mean_age_group_sh2 = mean(age_group_sh2, na.rm = T),
                     mean_age_group_sh3 = mean(age_group_sh3, na.rm = T),
                     mean_age_group_sh4 = mean(age_group_sh4, na.rm = T),
                     mean_age_group_sh5 = mean(age_group_sh5, na.rm = T),
                     mean_age_group_sh6 = mean(age_group_sh6, na.rm = T),
                     mean_race_share1 = mean(race_share1, na.rm = T),
                     mean_race_share2 = mean(race_share2, na.rm = T),
                     mean_race_share3 = mean(race_share3, na.rm = T),
                     mean_hispanic_share = mean(hispanic_share, na.rm = T),
                     mean_gender_share = mean(gender_share, na.rm = T),
                     mean_hsl_share = mean(hsl_share, na.rm = T),
                    mean_emp_sh_ind1 = mean(emp_sh_ind1, na.rm = T),
                     mean_emp_sh_ind2 = mean(emp_sh_ind2, na.rm = T),
                     mean_emp_sh_ind3 = mean(emp_sh_ind3, na.rm = T),
                     mean_emp_sh_ind4 = mean(emp_sh_ind4, na.rm = T),
                     mean_emp_sh_ind5 = mean(emp_sh_ind5, na.rm = T),
                     mean_emp_sh_ind6 = mean(emp_sh_ind6, na.rm = T),
                     mean_emp_sh_ind7 = mean(emp_sh_ind7, na.rm = T),
                     mean_emp_sh_ind8 = mean(emp_sh_ind8, na.rm = T),
                     mean_emp_sh_ind9 = mean(emp_sh_ind9, na.rm = T)
                     ) |>
```

Constructing Placebos

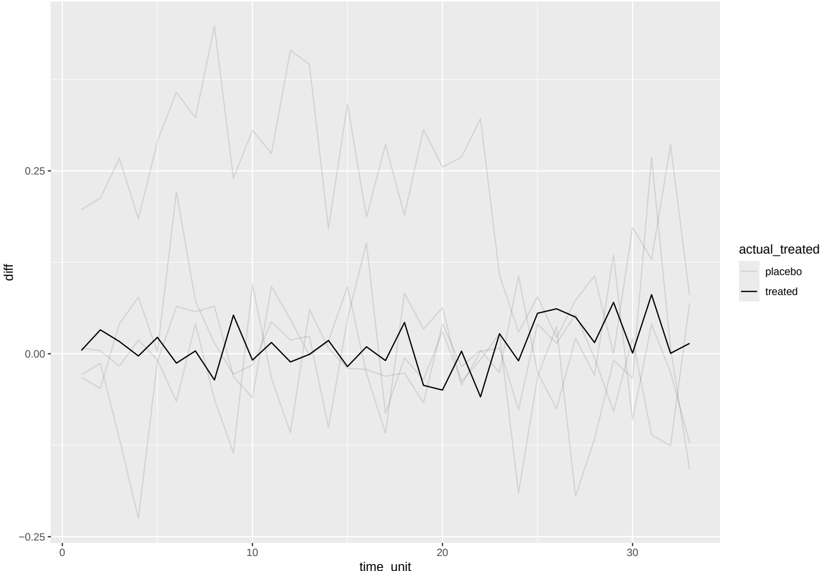
```
placebo_synth_tab <- placebo_synth_tab |>
  mutate(diff = real_y - synth_y,
         actual_treated = case_when(unit == "CA" ~ "treated",
                                    TRUE ~ "placebo"))

p1 <- placebo_synth_tab |>
  ggplot() +
  geom_line(aes(x = time_unit, y = diff,
                group = unit, alpha = actual_treated))
```

Constructing Placebos

```
print(p1)
```

```
## Warning: Using alpha for a discrete variable is not advised.
```



Constructing Placebos

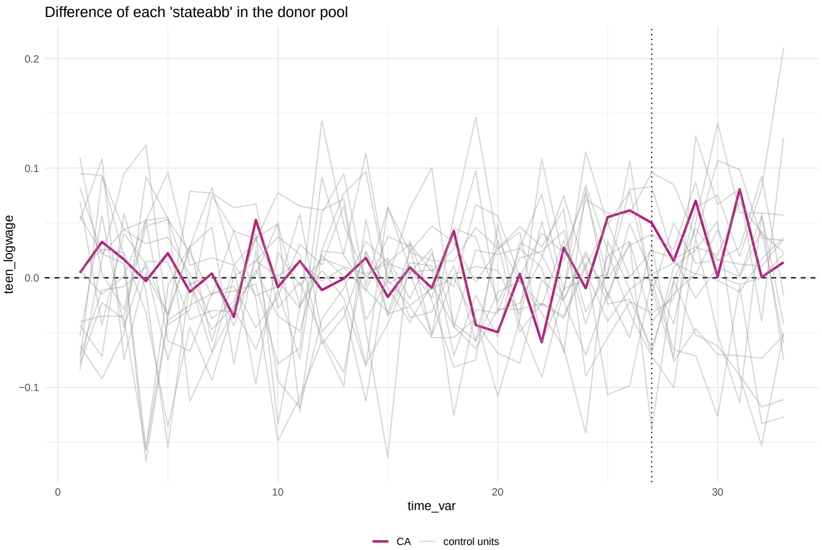
Rather than doing this by hand for all possible control states, the `tidysynth` package can do it for us!

```
synth <- synth |>
        generate_predictor(time_window = 1:26,
                     mean_teen_logwage = mean(teen_logwage, na.rm = T),
                     mean_age_group_sh1 = mean(age_group_sh1, na.rm = T),
                     mean_age_group_sh2 = mean(age_group_sh2, na.rm = T),
                     mean_age_group_sh3 = mean(age_group_sh3, na.rm = T),
                     mean_age_group_sh4 = mean(age_group_sh4, na.rm = T),
                     mean_age_group_sh5 = mean(age_group_sh5, na.rm = T),
                     mean_age_group_sh6 = mean(age_group_sh6, na.rm = T),
                     mean_race_share1 = mean(race_share1, na.rm = T),
                     mean_race_share2 = mean(race_share2, na.rm = T),
                     mean_race_share3 = mean(race_share3, na.rm = T),
                     mean_hispanic_share = mean(hispanic_share, na.rm = T),
                     mean_gender_share = mean(gender_share, na.rm = T),
                     mean_hsl_share = mean(hsl_share, na.rm = T),
                    mean_emp_sh_ind1 = mean(emp_sh_ind1, na.rm = T),
                     mean_emp_sh_ind2 = mean(emp_sh_ind2, na.rm = T),
                     mean_emp_sh_ind3 = mean(emp_sh_ind3, na.rm = T),
                     mean_emp_sh_ind4 = mean(emp_sh_ind4, na.rm = T),
                     mean_emp_sh_ind5 = mean(emp_sh_ind5, na.rm = T),
                     mean_emp_sh_ind6 = mean(emp_sh_ind6, na.rm = T),
                     mean_emp_sh_ind7 = mean(emp_sh_ind7, na.rm = T),
                     mean_emp_sh_ind8 = mean(emp_sh_ind8, na.rm = T),
                     mean_emp_sh_ind9 = mean(emp_sh_ind9, na.rm = T)
                     )
```

Constructing Placebos

```
synth_out <- synth |>
      generate_weights(optimization_window = 1:26) |>
    generate_control()

synth_out |> plot_placebos()
```



Calculating P-Values

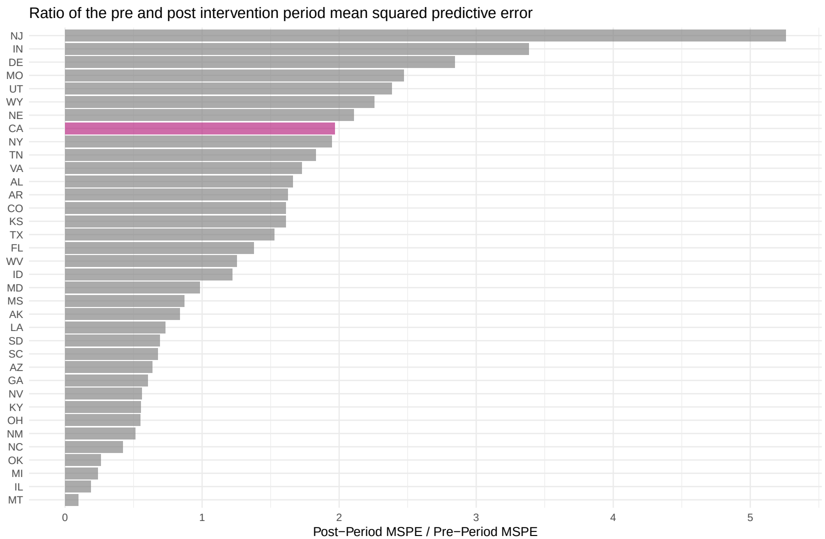
After we calculate the RMSE ratio for the actually treated unit and the placebo tests, we *rank* them by size to get a *pseudo p-value*.

If the actually treated unit's RMSE ratio is in the top percentile of the distribution of all RMSE ratios (say the top 10%), then we can say this estimate statistically significant.

Calculating P-Values

The `tidysynth` package will return the RMSE ratios in a figure as well using the `plot_mspe_ratio()` function.

```
synth_out |> plot_mspe_ratio()
```



Calculating P-Values

We can also use the `grab_significance()` function to see the rankings and associated p-value for the actually treated unit.

This RMSE ratio method of inference implies that the estimated effect of CA's minimum wage increase **did not** have a statistically significant effect on teen employment.

```
synth_out |> grab_significance()
```

```
## # A tibble: 36 x 8
##    unit_name type    pre_mspe post_mspe mspe_ratio  rank fishers_exact_pvalu
##    <chr>     <chr>      <dbl>     <dbl>      <dbl> <int>                <dbl
##  1 NJ        Donor    0.00819   0.0431        5.26     1                0.027
##  2 IN        Donor    0.00302   0.0102        3.38     2                0.055
##  3 DE        Donor    0.00435   0.0124        2.84     3                0.083
##  4 MO        Donor    0.00399   0.00987       2.47     4                0.111
##  5 UT        Donor    0.00356   0.00850       2.38     5                0.139
##  6 WY        Donor    0.0115    0.0260        2.26     6                0.167
##  7 NE        Donor    0.00255   0.00538       2.11     7                0.194
##  8 CA        Treated  0.00101   0.00198       1.97     8                0.222
##  9 NY        Donor    0.00276   0.00537       1.95     9                0.25
## 10 TN        Donor    0.00370   0.00678       1.83    10                0.278
## # i 26 more rows
## # i 1 more variable: z_score <dbl>
```

Combining Synthetic Controls and DiD

We see now that the synthetic control method offers a lot of utility, especially when we have only one treated unit.

But is there a way to combine the benefits of the synthetic control method with the standard DiD method?

Synthetic DiD

The synthetic DiD method estimates the standard DiD estimate **but using the weights from the synthetic control**.

The synthetic DiD method uses the *unit weights* from the synthetic control method above as well as **time weights** to give more weight to the pre-treatment periods that best fit.

Synthetic DiD in R

We can use the `synthdid` package in R to implement this method.

Since this is an in-development package, we need to install it from github. Install the `devtools` package if you have not yet done so.

```r
#install.packages("devtools")
#devtools::install_github("synth-inference/synthdid")
library(synthdid)
```

Synthetic DiD in R

We use the `panel_matrices()` function from `synthdid` to get the data in the correct form.

(Hint: make sure the class of object is a `data.frame` and not a `tibble`).

The method the `synthdid` package uses is to estimate the synthetic control using *pre-treatment outcome values*.

```
empwage_ca <- empwage_ca |>
  mutate(teen_logemp = as.numeric(teen_logemp),
         treat = as.logical(treat))

empwage_ca <- as.data.frame(empwage_ca)

setup <- panel.matrices(panel = as.data.frame(empwage_ca),
                        unit = "stateabb", time = "time_var",
                        outcome = "teen_logemp",
                        treatment = "treat")
```

Synthetic DiD in R

The `setup` object now contains all objects needed to run the `synthdid_estimat()` function.

Let's take a look at one of them:

```
setup$Y
```

```
##              1          2          3          4          5          6
## AK -1.0081962 -0.7273728 -0.6289909 -0.8517815 -1.0251937 -0.7791227 -0.5543
## AL -1.2929312 -1.1793571 -1.2341935 -1.1227144 -1.5162633 -1.1726885 -1.0470
## AR -1.2133995 -0.8442482 -0.8000836 -0.9253152 -1.1333850 -0.9169848 -0.7501
## AZ -0.7793198 -0.7700494 -0.7511143 -0.7691967 -0.7937306 -0.8546383 -0.6650
## CO -0.7890146 -0.6328306 -0.5448869 -0.6935204 -0.6996565 -0.6924205 -0.4500
## DE -0.9774293 -0.9554051 -0.6265315 -0.9257820 -1.0924189 -0.8099136 -0.7344
## FL -0.9325055 -0.7751482 -0.7526732 -0.7959629 -0.8806242 -0.7706282 -0.6985
## GA -0.9042336 -0.8736010 -0.8313586 -0.9662616 -0.9063002 -0.8760937 -0.7834
## ID -0.8318124 -0.6036627 -0.5577134 -0.9228904 -0.9249200 -0.6972987 -0.5382
## IL -0.9280728 -0.8132880 -0.6388296 -0.8049551 -0.9504480 -0.8800036 -0.6849
## IN -1.0905274 -0.8336256 -0.7230820 -1.0261382 -1.0392510 -0.8329032 -0.6950
## KS -0.7351025 -0.4652166 -0.4691918 -0.6021589 -0.6123627 -0.5249327 -0.5028
## KY -1.0115417 -0.9701820 -0.6896040 -1.0420095 -1.1939360 -0.9768864 -0.7763
## LA -1.2002695 -1.1907511 -0.9627468 -1.3504183 -1.3305941 -1.0768800 -0.9164
## MD -1.0080420 -0.8069361 -0.6769087 -1.0286375 -1.0118630 -0.6932645 -0.5489
## MI -1.0204558 -0.8443355 -0.6869813 -0.8897343 -0.9271311 -0.8242867 -0.6970
## MO -0.9184572 -0.7318141 -0.7135642 -0.9549940 -0.8966334 -0.7298988 -0.7086
## MS -1.2973616 -1.3882500 -1.2456637 -1.2540113 -1.2639359 -1.4871428 -1.1188
```

Synthetic DiD in R

The `setup` object now contains all objects needed to run the `synthdid_estimat()` function.

Let's take a look at one of them:

```
setup$N0
```

```
## [1] 35
```

```
setup$T0
```

```
## [1] 26
```

Synthetic DiD in R

Now we can run the actual synthetic DiD estimate:

```
synthdid_est <- synthdid_estimate(setup$Y, setup$N0, setup$T0)
synthdid_est
```

```
## synthdid: 0.065 +- NA. Effective N0/N0 = 27.4/35~0.8. Effective T0/T0 = 5.4/
```

Synthetic DiD in R

To get the standard error, the `synthdid` package uses a similar placebo method to the standard synthetic control.

We can compute this standard error using the `vcov(..., method = "placebo")` function (it may take a while).

The 95% confidence interval suggests the point estimate is not statistically significant.

```
synthdid_est_se <- sqrt(vcov(synthdid_est, method='placebo'))
synthdid_est_se
```

```
##           [,1]
## [1,] 0.0507795
```

```
ci_95 <- c(synthdid_est + 1.96 * synthdid_est_se, synthdid_est - 1.96 * synthdi
ci_95
```

```
## [1]  0.1648792 -0.0341764
```

Getting the unit weights

We can also look at the weights the synthetic control unit receives using the `synthdid_controls()` function.

The `weight.type = "omega"` argument tells the function to return the unit weights, and giving the `mass` argument a really big value tells it to return the weights for units that recieved even near-zero weights (not always useful with a large number of donors).

```
unit_weights <- data.frame(synthdid_controls(
  synthdid_est, weight.type = "omega", mass = 10e12)) |>
  rownames_to_column("stateabb") |>
  rename(unit_wght = estimate.1)
unit_weights
```

```
##    stateabb   unit_wght
## 1        OK 0.065635600
## 2        FL 0.053802340
## 3        CO 0.051360433
## 4        NY 0.048781840
## 5        AL 0.047372516
## 6        MI 0.043380290
## 7        TX 0.038948654
## 8        NM 0.038753851
## 9        NJ 0.037719497
```

Getting the time weights

We can also do the same to get the time weights setting the `weight.type` argument to "lambda".

Note the weights are only calculated for the *pre-treatment* periods.

```
time_weights <- data.frame(synthdid_controls(synthdid_est,
                                             weight.type = "lambda",
                                             mass = 1000000)) %>%
  rownames_to_column("time_var") %>%
  rename(time_wght = estimate.1) %>%
  mutate(time_var = as.numeric(time_var))
time_weights
```

```
##    time_var   time_wght
## 1        26 0.337036530
## 2        23 0.156688740
## 3        24 0.150664158
## 4        18 0.091646708
## 5        15 0.075971374
## 6        21 0.057941432
## 7        20 0.048459450
## 8        10 0.046852027
## 9        19 0.028138775
## 10        1 0.005044503
## 11        8 0.001556302
## 12       25 0.000000000
```

Replicating the synthdid estimate

Let's see if we can use the weights to double-check the `synthdid` estimate:

First, let's merge in our weights and make sure to give post-treatment periods a time weight of 1 and treated units (i.e., CA) a unit weight of 1.

```
empwage_ca <- empwage_ca |>
  left_join(unit_weights, by = "stateabb") |>
  left_join(time_weights, by = "time_var")

 empwage_ca <-  empwage_ca  %>%
  mutate(unit_wght = case_when(treated == 1 ~ 1,
                               TRUE ~ unit_wght))

 empwage_ca <- empwage_ca %>%
  mutate(time_wght = case_when(post == 1 ~ 1,
                               TRUE ~ time_wght))
```

Replicating the synthdid estimate

Now, we combine the unit and time weights for each observation before entering them into a TWFE model.

```
empwage_ca <- empwage_ca |>
   mutate(comb_wght = unit_wght * time_wght)

synthdid_est_2 <- feols(teen_logemp ~ treat | stateabb + time_var,
              weights = ~comb_wght,
             data = empwage_ca)
```

```
## NOTE: 540 observations removed because of 0-weight.
```

```
summary(synthdid_est_2)
```

```
## OLS estimation, Dep. Var.: teen_logemp
## Observations: 648
## Weights: comb_wght
## Fixed-effects: stateabb: 36,  time_var: 18
## Standard-errors: Clustered (stateabb)
##           Estimate Std. Error t value           Pr(>|t|)
## treatTRUE 0.065351   0.006526 10.0137 0.0000000000081787 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 0.009503     Adj. R2: 0.809807
##                  Within R2: 0.030939
```