6-4 Regression Discontinuity Design - Solutions

April 09, 2024

Regression Discontinuity

```
# install packages
# ----------
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```
# load
pacman::p_load(# Tidyverse packages including dplyr and ggplot2
               tidyverse,
               rdd,         # regression discontinuity design library
               ggthemes,
               tidymodels,  # machine learning workflow (R's version of Python's sklearn)
               here)

# run here to set working directory
here()
```

```
## [1] "/Users/Dora/git/Computational-Social-Science-Training-Program/6 Causal Inference"
```

```
# set seed
# ----------
set.seed(1)
```

Definition

In social sciences, a regression discontinuity design is a quasi-experimental pretest-posttest design that elicits the causal effects of interventions by assigning a cutoff or threshold above or below which an intervention is assigned. By comparing observations lying closely on either side of the threshold, it is possible to estimate the average treatment effect in environments in which randomization is unfeasible.

Treatment Using a Running Variable

In an ideal experiment, we would be able to randomly assign our units to treatment and control. However, as we've seen, this is not always possible in social science contexts. Let's consider a classic question from political science: do incumbent politicians enjoy an incumbency advantage? In other words, do incumbents garner a higher vote share than they would if they were running for the first time?

To explore this question, we are going to use data taken from Lee (2008). We have a few variables to define:

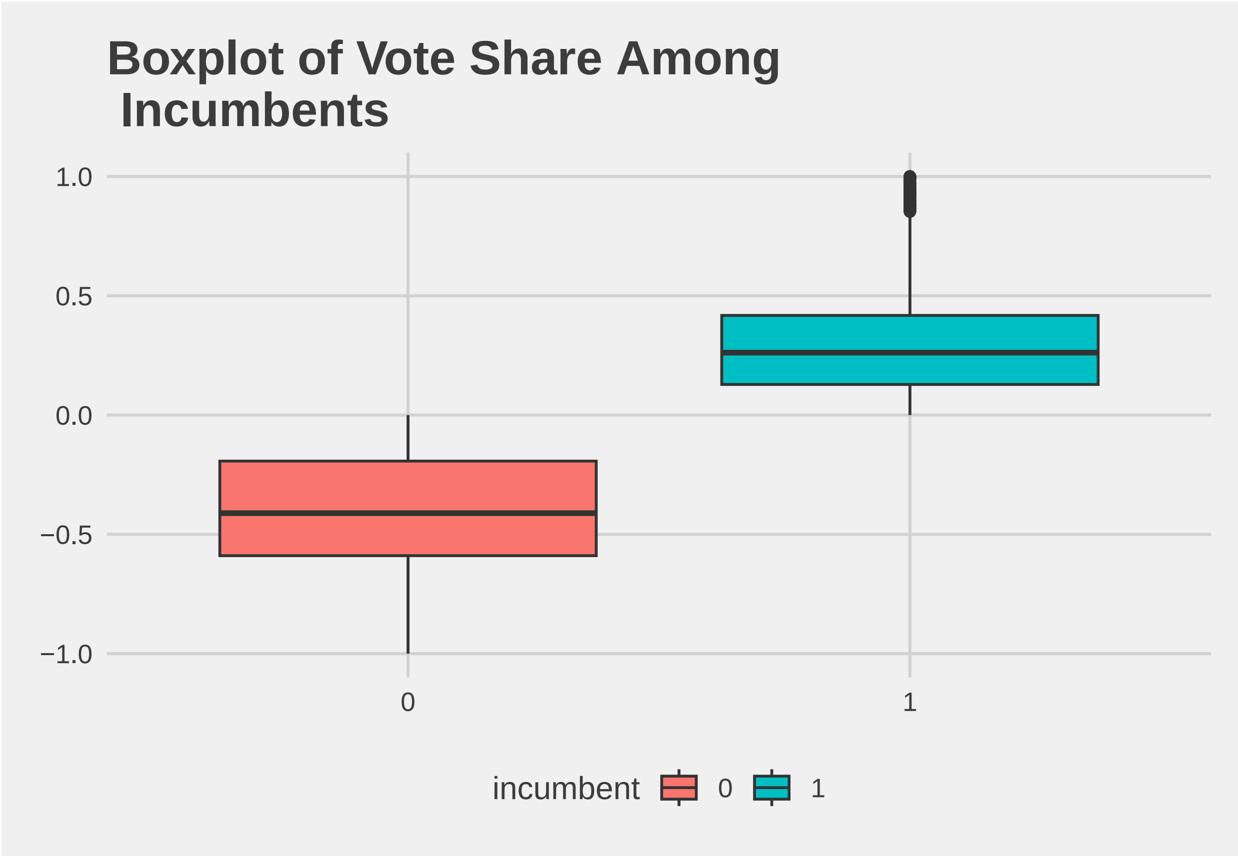
- `difshare`: Normalized proportion of vote share the party received in the last election
- `yearel`: Year of current election
- `myoutcomenext`: 0/1 binary for whether the candidate won re-election
- `win_relection`: "win"/"lose" whether the candidate won re-election
- `incumbent`: 0/1 for whether the candidate is an incumbent

```r
# load data
# ----------
elections <- read_csv('../../data/indiv_final.csv') %>%
  # add two new columns
  mutate(win_reelection = case_when(myoutcomenext == 1 ~ 'win',
                                    TRUE ~ 'lose')) %>%
  mutate(incumbent = case_when(difshare > 0  ~ 1,
                               TRUE ~ 0))
```

```
## Rows: 24937 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): yearel, myoutcomenext, difshare
##
## 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.
```

Suppose we are interested in the effect of incumbency on the probability of winning re-election. We might look at the distribution of vote shares by incumbency. Let's look at this boxplot:

```r
#
# visualize shares of incumbency
# ---------------------------------------------------------------
elections %>%
  mutate(incumbent = as_factor(incumbent)) %>% # as_factor to treat as integers instead of numeric
  ggplot() +
  geom_boxplot(aes(x = incumbent,
                   y = difshare,
                   group = incumbent,
                   fill = incumbent)) +
  ggtitle('Boxplot of Vote Share Among \n Incumbents') +
  theme_fivethirtyeight()
```



Looks like incumbents enjoy a pretty significant advantage! Let's investigate this further by this using a linear probability model:

```
#
# linear probability model
# ---------------------------------------------------------------

# linear probability model (instead of logit)
election_lm <- lm(myoutcomenext ~ incumbent,
                  data = elections)

# view
summary(election_lm)
```

```
##
## Call:
## lm(formula = myoutcomenext ~ incumbent, data = elections)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.78141 -0.01401 -0.01401  0.21859  0.98599
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.014012   0.002259   6.203 5.61e-10 ***
## incumbent   0.767395   0.003576 214.604  < 2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2765 on 24935 degrees of freedom
## Multiple R-squared:  0.6488, Adjusted R-squared:  0.6487
## F-statistic: 4.606e+04 on 1 and 24935 DF,  p-value: < 2.2e-16
```

```r
# another option is to use model summary
# ----------
#library(modelsummary)
#modelsummary(election_lm)
```

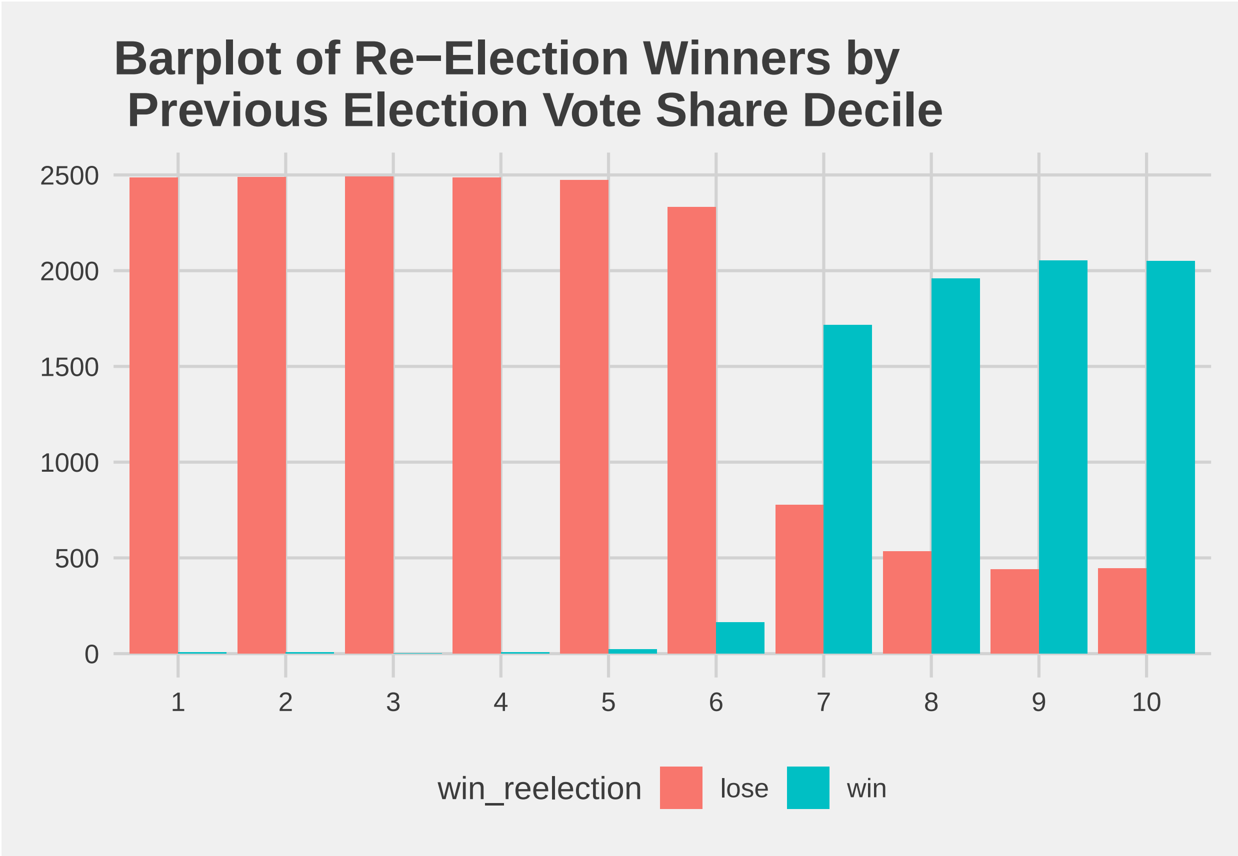
Incumbency is a statistically significant covariate!

What might be the problem with using .77 as the coefficient in this case? Let's take at the distribution of previous elections vote shares between winners and losers.

```r
#
# visualize distribution of winners and losers based on previous election
# ----------------------------------------------------------------
elections %>%

  # process
  # ----------
  mutate(incumbent = as_factor(incumbent)) %>%        # as_factor makes incumbent an integer
  mutate(decile = as_factor(ntile(difshare, 10))) %>% # calculates deciles of difshare

  # plot
  # ----------
  ggplot() +
  geom_bar(aes(y = decile,    # bar plot counts number of people who win re-election by decile
               fill = win_reelection),
           position = 'dodge') +
  ggtitle('Barplot of Re-Election Winners by \n Previous Election Vote Share Decile') +
  theme_fivethirtyeight() +
  coord_flip()
```



QUESTION: Groups 1 - 4 look ok - by definition someone cannot win reelection if they lost the last election. But what about the difference in distributions between deciles like 5 and 6, versus the distributions in deciles 7-10? Why would these different distributions pose a problem for trusting our previous point estimate?

ANSWER: Our main worry is that people who won huge proportions of the vote in the last election are systematically different from those who barely won in the last election. For instance, it should not be surprising when a Democratic incumbent carries CA-13 (Berkeley's congressional district) not because they enjoy an incumbency advantage but because the district is heavily Democratic for other reasons. In other words, there might be other reasons they win re-election beyond simple incumbency advantage that we are trying to estimate.

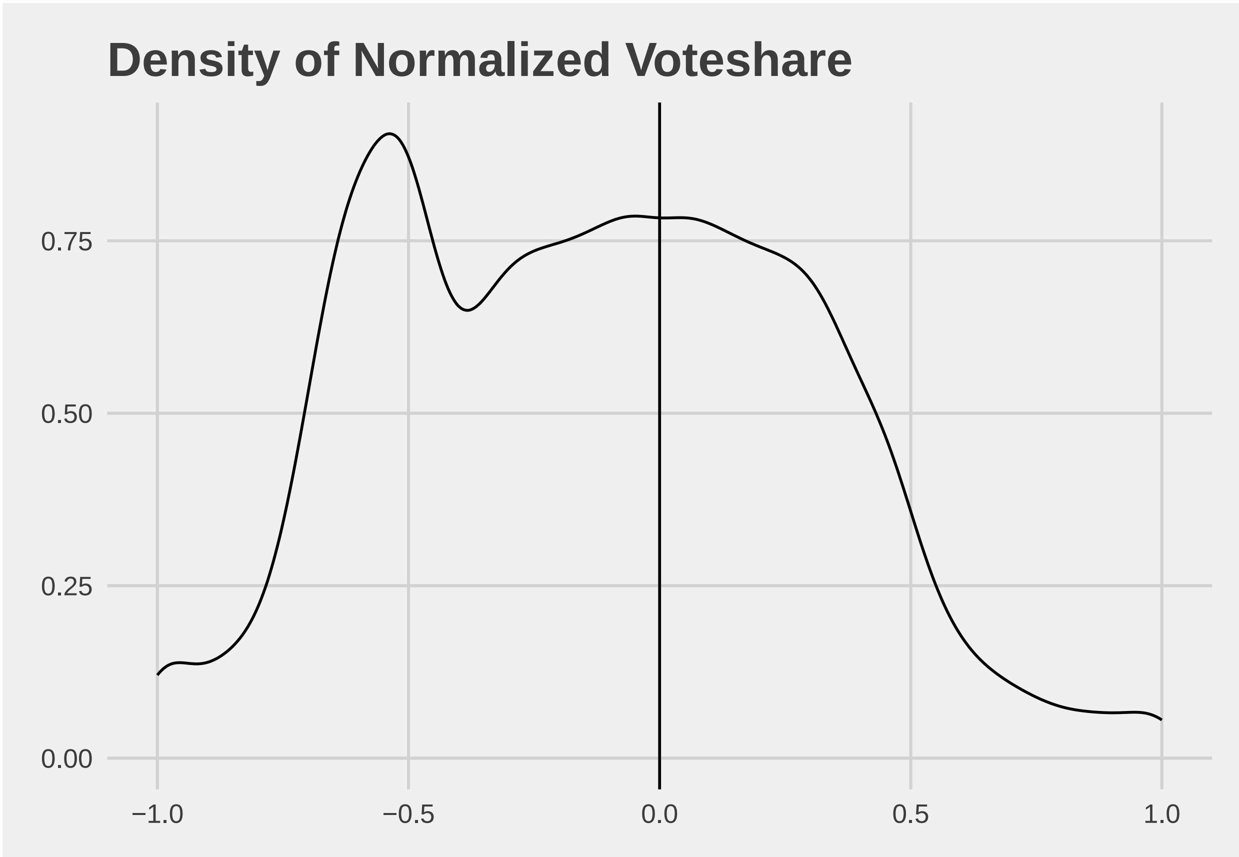
Running Variable

We might assume that the selection into treatment/control conditions is driven entirely by observable characteristics (selection on the observables). If this was the case, then we could add these characteristics as controls to our regression, and we would be in the clear! In practice, this is rarely a realistic assumption, and we most likely will worry about selection on unobservable characteristics - essentially confounders that we do not see but affect both treatment and the outcome.

The basic intuition behind regression discontinuity designs is that we use a **running variable** that determines whether a unit was assigned to either treatment or control. Let's take a look:

```
#
# visualize distribution of winners and losers based on previous election by the running variable
# ----------------------------------------------------------------
elections %>%
  ggplot() +
  geom_density(aes(x = difshare)) + # smoothed kernal density estimator
```

```r
  ggtitle('Density of Normalized Voteshare') +
  geom_vline(xintercept = 0) +
  theme_fivethirtyeight()
```



"0" here is the cutpoint we use to assign someone to either incumbent or non-incumbent treatment conditions. The basic logic behind the RD is that those individuals on either side of the cutpoints will be very similar to each other in terms of baseline covariates (on both observed and unobserved characteristics).

McCrary Density Test

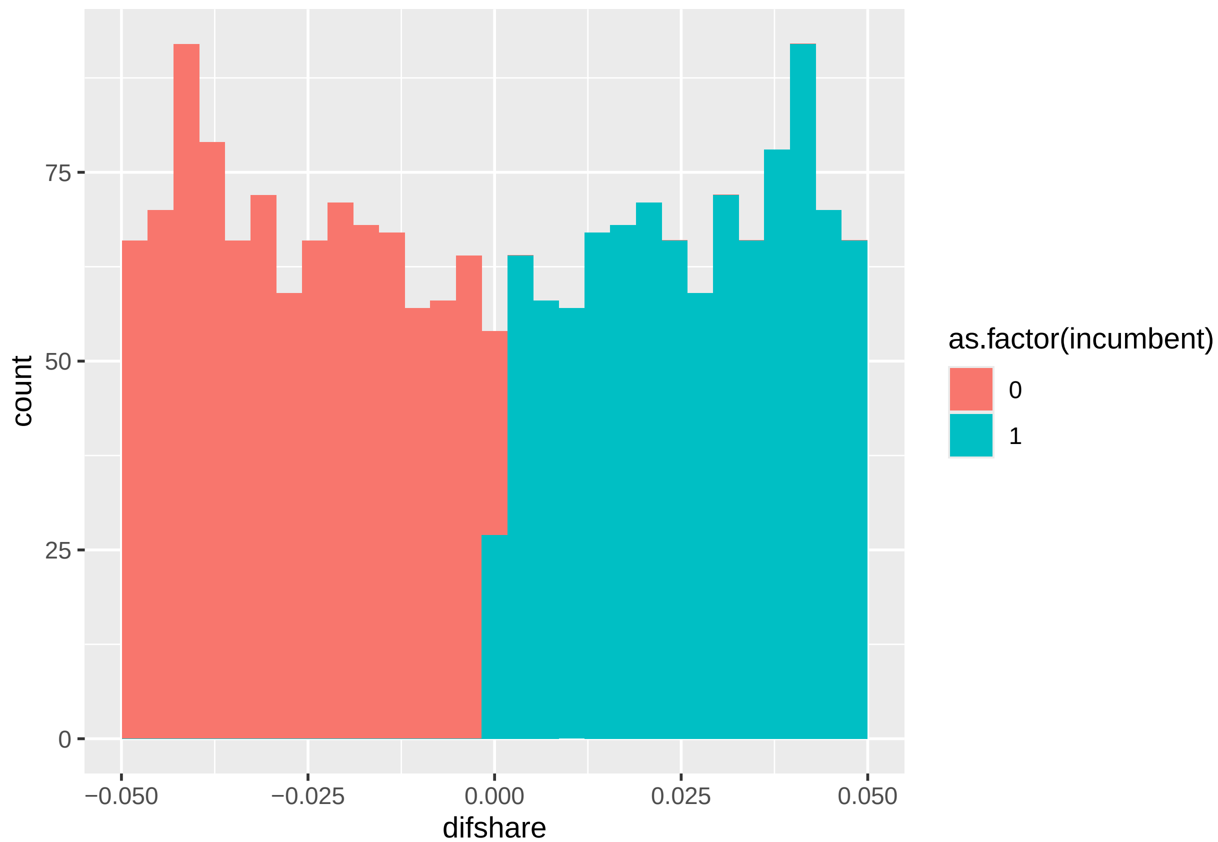
Before we estimate model for individuals on either side of the cutpoint though, we might be concerned about their manipulation into treatment and control. For example, if the running variable was a passing test score to move onto to the next grade, you might imagine that a teacher bumps up a student from a 59 to a 60. Similarly, if the cutoff to be recruited into an honors program is a 90, you might worry that a student with an 89 who knows that they could appeal to be admitted anyway differs from the student who does not think such a thing is negotiable.

McCrary proposes a test for this kind of problem. Specifically, he motivates the test by giving an example of patients who are assigned to either waiting room A or B, but only waiting room A receives the experimental treatment. Patients learn about this fact, so as those who are assigned to waiting room B are are walking there, they instead decide to go to waiting room A. Given enough patients doing this, we should expect waiting room A to become crowded and waiting room B to be relatively empty. Let's see what that looks like graphically in our dataset:

```r
#
# density test using histograms
# ----------------------------------------------------------------
```

```r
elections %>%
  # filter to just those who won or lost (basically zoom in)
  filter(difshare > -.05 & difshare < .05) %>%
  ggplot() +
  geom_histogram(aes(x = difshare,
                     group = as.factor(incumbent),
                     fill = as.factor(incumbent)))
```

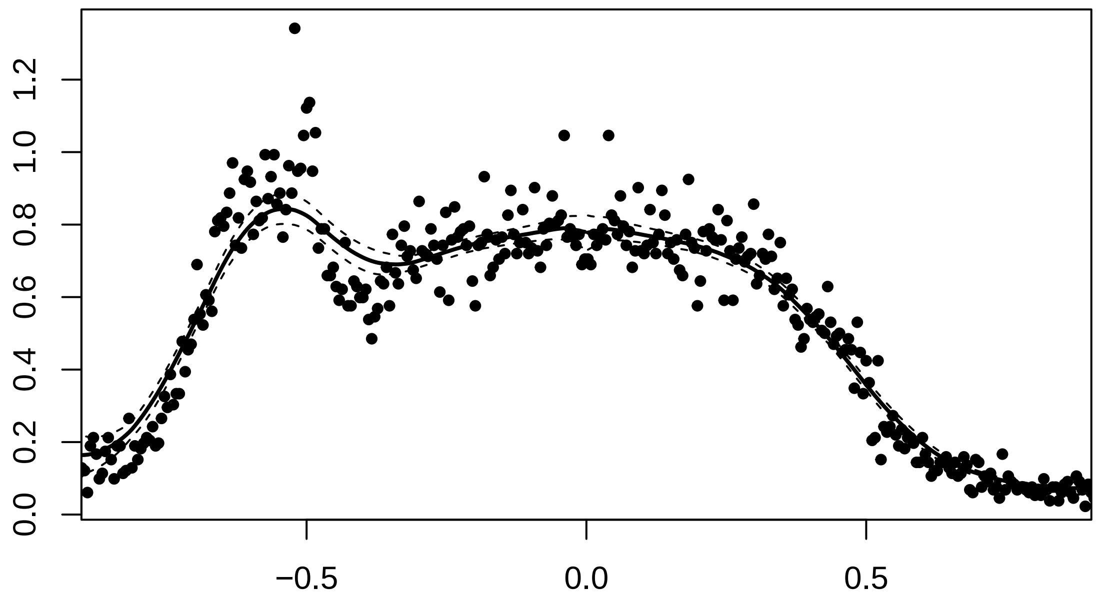
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It looks like there isn't evidence of sorting at the cutpoint - in fact the distributions look identical. McCrary points out that these types of histograms can be biased at the cutpoint. He instead advocates for using local linear regressions to smooth the histograms at the cutpoint. Luckily, this procedure is implemented for us in the rdd package:

```r
#
# density test using histograms
# ------------------------------------------------------------------
DCdensity(elections$difshare, # specify variable
          cutpoint = 0,       # specify the cutpoint
          verbose = TRUE,     # gives the output
          ext.out = TRUE,     # save the output
          htest = TRUE)       # pass through to other testing functions in base R
```

```
## Using calculated bin size:  0.005
## Using calculated bandwidth:  0.183
```




```
## Log difference in heights is  -0.002  with SE  0.052
##   this gives a z-stat of  -0.048
##   and a p value of  0.962

## $theta
## [1] -0.002470001
##
## $se
## [1] 0.05193608
##
## $z
## [1] -0.04755847
##
## $p
## [1] 0.9620681
##
## $binsize
## [1] 0.005290885
##
## $bw
## [1] 0.1833466
##
## $cutpoint
## [1] 0
##
## $data
##          cellmp    cellval
```

```
## 1   -1.002622783 0.01515854
## 2   -0.997331898 1.60680547
## 3   -0.992041013 0.21979886
## 4   -0.986750127 0.07579271
## 5   -0.981459242 0.03789636
## 6   -0.976168356 0.12884761
## 7   -0.970877471 0.12126834
## 8   -0.965586586 0.07579271
## 9   -0.960295700 0.11368907
## 10  -0.955004815 0.08337198
## 11  -0.949713929 0.06821344
## 12  -0.944423044 0.09853052
## 13  -0.939132159 0.20464032
## 14  -0.933841273 0.08337198
## 15  -0.928550388 0.16674396
## 16  -0.923259502 0.15916469
## 17  -0.917968617 0.07579271
## 18  -0.912677732 0.06063417
## 19  -0.907386846 0.11368907
## 20  -0.902095961 0.12884761
## 21  -0.896805075 0.12126834
## 22  -0.891514190 0.06063417
## 23  -0.886223305 0.18948178
## 24  -0.880932419 0.21221959
## 25  -0.875641534 0.16674396
## 26  -0.870350648 0.09853052
## 27  -0.865059763 0.11368907
## 28  -0.859768878 0.17432323
## 29  -0.854477992 0.21221959
## 30  -0.849187107 0.15158542
## 31  -0.843896221 0.09853052
## 32  -0.838605336 0.18948178
## 33  -0.833314451 0.18948178
## 34  -0.828023565 0.11368907
## 35  -0.822732680 0.12126834
## 36  -0.817441794 0.26527449
## 37  -0.812150909 0.12884761
## 38  -0.806860024 0.18948178
## 39  -0.801569138 0.15158542
## 40  -0.796278253 0.18190251
## 41  -0.790987367 0.19706105
## 42  -0.785696482 0.21221959
## 43  -0.780405597 0.20464032
## 44  -0.775114711 0.24253667
## 45  -0.769823826 0.18948178
## 46  -0.764532940 0.19706105
## 47  -0.759242055 0.26527449
## 48  -0.753951170 0.32590866
## 49  -0.748660284 0.29559157
## 50  -0.743369399 0.38654282
## 51  -0.738078513 0.30317084
## 52  -0.732787628 0.33348793
## 53  -0.727496743 0.33348793
## 54  -0.722205857 0.47749408
```

```
## 55  -0.716914972 0.39412210
## 56  -0.711624086 0.45475626
## 57  -0.706333201 0.46991481
## 58  -0.701042316 0.53812825
## 59  -0.695751430 0.68971367
## 60  -0.690460545 0.55328679
## 61  -0.685169659 0.52296970
## 62  -0.679878774 0.60634169
## 63  -0.674587889 0.59118314
## 64  -0.669297003 0.56086606
## 65  -0.664006118 0.78066492
## 66  -0.658715232 0.81098200
## 67  -0.653424347 0.81856128
## 68  -0.648133462 0.79582346
## 69  -0.642842576 0.83371982
## 70  -0.637551691 0.88677472
## 71  -0.632260805 0.97014670
## 72  -0.626969920 0.74276857
## 73  -0.621679035 0.81856128
## 74  -0.616388149 0.73518929
## 75  -0.611097264 0.92467107
## 76  -0.605806378 0.94740888
## 77  -0.600515493 0.91709180
## 78  -0.595224608 0.77308565
## 79  -0.589933722 0.86403690
## 80  -0.584642837 0.81098200
## 81  -0.579351951 0.81856128
## 82  -0.574061066 0.99288451
## 83  -0.568770181 0.87161617
## 84  -0.563479295 0.93225034
## 85  -0.558188410 0.99288451
## 86  -0.552897524 0.85645763
## 87  -0.547606639 0.88677472
## 88  -0.542315754 0.76550638
## 89  -0.537024868 0.84129909
## 90  -0.531733983 0.96256743
## 91  -0.526443097 0.88677472
## 92  -0.521152212 1.34153098
## 93  -0.515861327 0.94740888
## 94  -0.510570441 0.95498816
## 95  -0.505279556 1.04593941
## 96  -0.499988670 1.12173212
## 97  -0.494697785 1.13689066
## 98  -0.489406900 0.94740888
## 99  -0.484116014 1.05351868
## 100 -0.478825129 0.73518929
## 101 -0.473534243 0.78824419
## 102 -0.468243358 0.78824419
## 103 -0.462952473 0.65939658
## 104 -0.457661587 0.65939658
## 105 -0.452370702 0.68213440
## 106 -0.447079816 0.62907950
## 107 -0.441788931 0.59118314
## 108 -0.436498046 0.62150023
```

```
## 109 -0.431207160 0.75034784
## 110 -0.425916275 0.57602460
## 111 -0.420625389 0.57602460
## 112 -0.415334504 0.64423804
## 113 -0.410043619 0.62907950
## 114 -0.404752733 0.59876241
## 115 -0.399461848 0.59876241
## 116 -0.394170962 0.62150023
## 117 -0.388880077 0.53812825
## 118 -0.383589192 0.48507335
## 119 -0.378298306 0.54570752
## 120 -0.373007421 0.56844533
## 121 -0.367716535 0.64423804
## 122 -0.362425650 0.63665877
## 123 -0.357134765 0.68213440
## 124 -0.351843879 0.57602460
## 125 -0.346552994 0.77308565
## 126 -0.341262108 0.66697585
## 127 -0.335971223 0.63665877
## 128 -0.330680338 0.74276857
## 129 -0.325389452 0.79582346
## 130 -0.320098567 0.71245148
## 131 -0.314807681 0.72761002
## 132 -0.309516796 0.67455513
## 133 -0.304225911 0.65181731
## 134 -0.298935025 0.86403690
## 135 -0.293644140 0.72761002
## 136 -0.288353254 0.72003075
## 137 -0.283062369 0.71245148
## 138 -0.277771484 0.78824419
## 139 -0.272480598 0.74276857
## 140 -0.267189713 0.70487221
## 141 -0.261898827 0.61392096
## 142 -0.256607942 0.74276857
## 143 -0.251317057 0.83371982
## 144 -0.246026171 0.59118314
## 145 -0.240735286 0.75792711
## 146 -0.235444400 0.84887836
## 147 -0.230153515 0.76550638
## 148 -0.224862630 0.78066492
## 149 -0.219571744 0.78824419
## 150 -0.214280859 0.74276857
## 151 -0.208989973 0.79582346
## 152 -0.203699088 0.64423804
## 153 -0.198408203 0.57602460
## 154 -0.193117317 0.74276857
## 155 -0.187826432 0.75034784
## 156 -0.182535546 0.93225034
## 157 -0.177244661 0.77308565
## 158 -0.171953776 0.65939658
## 159 -0.166662890 0.68213440
## 160 -0.161372005 0.75792711
## 161 -0.156081119 0.70487221
## 162 -0.150790234 0.76550638
```

```
## 163 -0.145499349 0.72003075
## 164 -0.140208463 0.82614055
## 165 -0.134917578 0.89435399
## 166 -0.129626692 0.77308565
## 167 -0.124335807 0.72003075
## 168 -0.119044922 0.75034784
## 169 -0.113754036 0.84129909
## 170 -0.108463151 0.75034784
## 171 -0.103172265 0.72003075
## 172 -0.097881380 0.73518929
## 173 -0.092590495 0.90193326
## 174 -0.087299609 0.72761002
## 175 -0.082008724 0.68213440
## 176 -0.076717838 0.78824419
## 177 -0.071426953 0.74276857
## 178 -0.066136068 0.80340273
## 179 -0.060845182 0.87919544
## 180 -0.055554297 0.80340273
## 181 -0.050263411 0.81098200
## 182 -0.044972526 0.82614055
## 183 -0.039681641 1.04593941
## 184 -0.034390755 0.76550638
## 185 -0.029099870 0.78824419
## 186 -0.023808984 0.76550638
## 187 -0.018518099 0.74276857
## 188 -0.013227214 0.77308565
## 189 -0.007936328 0.68971367
## 190 -0.002645443 0.70487221
## 191  0.002645443 0.70487221
## 192  0.007936328 0.68971367
## 193  0.013227214 0.77308565
## 194  0.018518099 0.74276857
## 195  0.023808984 0.76550638
## 196  0.029099870 0.78824419
## 197  0.034390755 0.75792711
## 198  0.039681641 1.04593941
## 199  0.044972526 0.82614055
## 200  0.050263411 0.81098200
## 201  0.055554297 0.77308565
## 202  0.060845182 0.87919544
## 203  0.066136068 0.79582346
## 204  0.071426953 0.74276857
## 205  0.076717838 0.78066492
## 206  0.082008724 0.68213440
## 207  0.087299609 0.72761002
## 208  0.092590495 0.90193326
## 209  0.097881380 0.72761002
## 210  0.103172265 0.72003075
## 211  0.108463151 0.74276857
## 212  0.113754036 0.84129909
## 213  0.119044922 0.75034784
## 214  0.124335807 0.72003075
## 215  0.129626692 0.77308565
## 216  0.134917578 0.89435399
```

```
## 217  0.140208463 0.82614055
## 218  0.145499349 0.72003075
## 219  0.150790234 0.75034784
## 220  0.156081119 0.70487221
## 221  0.161372005 0.75792711
## 222  0.166662890 0.67455513
## 223  0.171953776 0.65939658
## 224  0.177244661 0.77308565
## 225  0.182535546 0.92467107
## 226  0.187826432 0.75034784
## 227  0.193117317 0.73518929
## 228  0.198408203 0.57602460
## 229  0.203699088 0.64423804
## 230  0.208989973 0.78066492
## 231  0.214280859 0.72761002
## 232  0.219571744 0.78824419
## 233  0.224862630 0.76550638
## 234  0.230153515 0.75792711
## 235  0.235444400 0.84129909
## 236  0.240735286 0.75792711
## 237  0.246026171 0.59118314
## 238  0.251317057 0.81098200
## 239  0.256607942 0.72761002
## 240  0.261898827 0.59118314
## 241  0.267189713 0.70487221
## 242  0.272480598 0.73518929
## 243  0.277771484 0.76550638
## 244  0.283062369 0.69729294
## 245  0.288353254 0.71245148
## 246  0.293644140 0.72003075
## 247  0.298935025 0.85645763
## 248  0.304225911 0.63665877
## 249  0.309516796 0.65939658
## 250  0.314807681 0.72003075
## 251  0.320098567 0.70487221
## 252  0.325389452 0.77308565
## 253  0.330680338 0.71245148
## 254  0.335971223 0.62150023
## 255  0.341262108 0.65181731
## 256  0.346552994 0.75034784
## 257  0.351843879 0.57602460
## 258  0.357134765 0.65181731
## 259  0.362425650 0.60634169
## 260  0.367716535 0.62150023
## 261  0.373007421 0.53812825
## 262  0.378298306 0.52296970
## 263  0.383589192 0.46233554
## 264  0.388880077 0.48507335
## 265  0.394170962 0.56844533
## 266  0.399461848 0.53812825
## 267  0.404752733 0.53054898
## 268  0.410043619 0.54570752
## 269  0.415334504 0.55328679
## 270  0.420625389 0.50781116
```

```
## 271  0.425916275 0.50023189
## 272  0.431207160 0.62907950
## 273  0.436498046 0.53054898
## 274  0.441788931 0.46991481
## 275  0.447079816 0.49265262
## 276  0.452370702 0.50023189
## 277  0.457661587 0.44717699
## 278  0.462952473 0.45475626
## 279  0.468243358 0.48507335
## 280  0.473534243 0.45475626
## 281  0.478825129 0.34864647
## 282  0.484116014 0.53054898
## 283  0.489406900 0.44717699
## 284  0.494697785 0.33348793
## 285  0.499988670 0.42443918
## 286  0.505279556 0.36380501
## 287  0.510570441 0.20464032
## 288  0.515861327 0.21221959
## 289  0.521152212 0.42443918
## 290  0.526443097 0.15158542
## 291  0.531733983 0.24253667
## 292  0.537024868 0.22737813
## 293  0.542315754 0.24253667
## 294  0.547606639 0.27285376
## 295  0.552897524 0.21979886
## 296  0.558188410 0.18948178
## 297  0.563479295 0.23495740
## 298  0.568770181 0.18190251
## 299  0.574061066 0.21221959
## 300  0.579351951 0.21221959
## 301  0.584642837 0.19706105
## 302  0.589933722 0.14400615
## 303  0.595224608 0.14400615
## 304  0.600515493 0.21221959
## 305  0.605806378 0.16674396
## 306  0.611097264 0.14400615
## 307  0.616388149 0.10610980
## 308  0.621679035 0.12126834
## 309  0.626969920 0.12126834
## 310  0.632260805 0.14400615
## 311  0.637551691 0.14400615
## 312  0.642842576 0.15916469
## 313  0.648133462 0.12884761
## 314  0.653424347 0.11368907
## 315  0.658715232 0.14400615
## 316  0.664006118 0.10610980
## 317  0.669297003 0.11368907
## 318  0.674587889 0.15916469
## 319  0.679878774 0.13642688
## 320  0.685169659 0.06821344
## 321  0.690460545 0.06063417
## 322  0.695751430 0.15158542
## 323  0.701042316 0.14400615
## 324  0.706333201 0.07579271
```

```
## 325  0.711624086 0.10610980
## 326  0.716914972 0.09095125
## 327  0.722205857 0.11368907
## 328  0.727496743 0.06821344
## 329  0.732787628 0.07579271
## 330  0.738078513 0.04547563
## 331  0.743369399 0.16674396
## 332  0.748660284 0.06821344
## 333  0.753951170 0.10610980
## 334  0.759242055 0.09095125
## 335  0.764532940 0.08337198
## 336  0.769823826 0.06821344
## 337  0.775114711 0.07579271
## 338  0.780405597 0.07579271
## 339  0.785696482 0.06821344
## 340  0.790987367 0.06063417
## 341  0.796278253 0.07579271
## 342  0.801569138 0.05305490
## 343  0.806860024 0.06821344
## 344  0.812150909 0.05305490
## 345  0.817441794 0.09853052
## 346  0.822732680 0.06821344
## 347  0.828023565 0.03789636
## 348  0.833314451 0.07579271
## 349  0.838605336 0.07579271
## 350  0.843896221 0.03789636
## 351  0.849187107 0.06063417
## 352  0.854477992 0.08337198
## 353  0.859768878 0.09095125
## 354  0.865059763 0.06063417
## 355  0.870350648 0.04547563
## 356  0.875641534 0.10610980
## 357  0.880932419 0.09095125
## 358  0.886223305 0.06821344
## 359  0.891514190 0.02273781
## 360  0.896805075 0.08337198
## 361  0.902095961 0.06063417
## 362  0.907386846 0.04547563
## 363  0.912677732 0.03789636
## 364  0.917968617 0.04547563
## 365  0.923259502 0.08337198
## 366  0.928550388 0.07579271
## 367  0.933841273 0.07579271
## 368  0.939132159 0.04547563
## 369  0.944423044 0.09095125
## 370  0.949713929 0.04547563
## 371  0.955004815 0.04547563
## 372  0.960295700 0.05305490
## 373  0.965586586 0.03789636
## 374  0.970877471 0.09095125
## 375  0.976168356 0.03031708
## 376  0.981459242 0.02273781
## 377  0.986750127 0.04547563
## 378  0.992041013 0.11368907
```

```
## 379  0.997331898 0.65939658
## 380  1.002622783 0.01515854
```

The hypothesis test is looking to see whether the density of the points is statistically different at the cutpoint. The p-value does is rather larger, indicating that there is no evidence of this (it would be hard to manipulate winning an election!).

Sharp Discontinuity

We can go ahead and estimate our model now! Once again the `rdd` package provides a nice function to let us do this:

```
#
# Sharp discontinuity
# ----------------------------------------------------------------
sharp_rdd_model <-
  RDestimate(myoutcomenext ~ difshare,  # y ~ x | c1 + c2... (add additional covariates using the "|")
             data = elections,          # data
             cutpoint = 0,              # specify cut point
             bw = .25,                  # specify bandwidth from Lee paper
             model = TRUE)              # return a model object

# view output
sharp_rdd_model
```

```
##
## Call:
## RDestimate(formula = myoutcomenext ~ difshare, data = elections,
##     cutpoint = 0, bw = 0.25, model = TRUE)
##
## Coefficients:
##      LATE    Half-BW  Double-BW
##    0.5158     0.4725     0.5855
```

QUESTION: How does our LATE compare to the .77 estimate we saw before?

ANSWER: Much lower, a difference of about 20% in predicted probability of being re-elected.

Fuzzy Discontinuity

We can also easily implement a fuzzy RD design. As we discussed in lecture, a fuzzy RD does not use one cutoff to assign to treatment and control, but rather uses the running variable as an instrument. To do a fuzzy RD, you simply need to add a `z` to your formula to indicate the treatment variable.

```
#
# Fuzzy discontinuity
# ----------------------------------------------------------------
fuzzy_rdd_model <-
  RDestimate(# A fuzzy rdd requires you add "z" to the formula to indicate the treatment variable
             # formula = y ~ x + z + | c1 + c2 (add additional covariates using the "|")
             formula = myoutcomenext ~ difshare + incumbent, # incumbent is our "z" in this case
             data = elections,          # data
             cutpoint = 0,              # specify cut point
             bw = .25,                  # specify bandwidth from Lee paper
             model = TRUE)              # return a model object

fuzzy_rdd_model
```

```
##
## Call:
## RDestimate(formula = myoutcomenext ~ difshare + incumbent, data = elections,
##     cutpoint = 0, bw = 0.25, model = TRUE)
##
## Coefficients:
##      LATE    Half-BW  Double-BW
##    0.5158     0.4725     0.5855
```

QUESTION: Our LATE for both the Sharp and Fuzzy models was the same here. Does that make sense?

ANSWER: Yes! Because we defined our treatment indicator in sharp terms relative to our running variable, the sharp and fuzzy designs should be identical in this case.

Bandwidth Selection

One of the main drawbacks of the regression discontinuity design is determining the optimal choice of bandwidth around the cutpoint. The intuition is that we want to pick a bandwidth such that the units on either side are very similar on both observed and unobserved characteristics - but if we knew how to do that then we could just use all of the data and matching! One way to select the bandwidth might be theory-driven in that the analyst picks the bandwidth that they think should yield unbiased estimates.

The `rdd` package implements the Imbens-Kalyanaraman method to approach this problem. Imbens and Kalyanaraman advocate for optimizing the mean squared error using an algorithm that basically:

- Chooses an initial bandwidth and calculates the conditional expectation function and variance of y at the cutpoint
- Chooses a second initial bandwidth and do the same thing but calculate a second derivative of the CEF
- Add a regularization penalty

By iterating on these steps, we can eventually find the optimal bandwidth. Luckily, this is also implemented for us and we can just leave the `bw` argument blank by default to do this calculation:

```r
#
# let the model choose the bandwidth iteratively
# ----------------------------------------------------------------
rdd_model <-
  RDestimate(myoutcomenext ~ difshare, # formula
           data = elections,           # data
                                       # omit bw argument and it will search automatically
           cutpoint = 0)               # specify cutpoint

# view model output
rdd_model
```

```
##
## Call:
## RDestimate(formula = myoutcomenext ~ difshare, data = elections,
##     cutpoint = 0)
##
## Coefficients:
##      LATE    Half-BW  Double-BW
##    0.4803     0.4608     0.5340
```

Under the hood, `RDestimate()` function will use `IKbandwidth()` to identify the optimal bandwidth.

```r
# use the Imbens-Kalyanaraman to identify the optimal bandwidth  - should get the same result
# ---------
```

```r
IKbandwidth(elections$difshare,
            elections$myoutcomenext,
            cutpoint = NULL,
            verbose = TRUE,
            kernel = "triangular")
```

```
## Using default cutpoint of zero.
## Imbens-Kalyanamaran Optimal Bandwidth:  0.151

## [1] 0.1514853
```

QUESTION: How did the Imbens-Kalyanaraman bandwidth estimate compare to our choice of .25?

ANSWER: This bandwidth is slightly different and therefore the estimate is slightly different, indicating the previous estimates might have been slightly biased based on our selection of bandwidth. Note that you can access the bandwidth from the summary object.

Challenge

Another option is to use cross-validation. The basic procedure here is:

- Choose several values of bandwidths to search through then for each bandwidth value:
  - Split the data into v-folds
  - Estimate a RDD model using that bandwidth and calculate the MSE in each fold
  - Average the MSE across folds
- Select the bandwidth with the lowest MSE

See if you can implement these steps for cross-validation on your own! In the solutions, we make use of a few of the more advanced/latest tools in R like `predict()`, vfold from tidymodels, and purrr, a functional programming library that is part of the tidyverse. You may use these tools, or whatever else you like to attempt this challenge! Find an optimal bandwidth using this procedure, and report your average treatment effect.

Also note that the `RDEstimate()` function returns an object with class "RD". See if you can extract the model object from it for calculating the MSE.

Solution

1. The first step here is to create a function that we will use to estimate a regression discontinuity and calculate the Mean Squared Error.

```r
#
# 1: create a function
# ----------------------------------------------------------------
calculate_rdd_and_bandwidth <- function(df_split, bw){ # two arguments the function takes

  # specify model
  # ---------
  rdd_model <-
    RDestimate(
           myoutcomenext ~ difshare,  # formula
           data = df_split,  # specify data as df_split
           cutpoint = 0,     # specify cutpoint
           bw = bw,          # specify bandwidth that will take various values
           model = TRUE)     # return a model object

  # calculations
  # ---------
```

```r
  # create a dataframe
  mse_data <- data.frame(
    # get predictions from the model
    pred = predict(rdd_model$model[[1]]),  # pull models from the object because rrd does not automatic
    # compare them to the actual values of myoutcomenext
    actual = (df_split %>% filter(difshare >= -bw & difshare <= bw))$difshare)
    # return the mean of MSE
    return(mean((mse_data$actual - mse_data$pred)^2))
}
```

2. Now we need to split our data. The `vfold()` function from `tidymodels` is similar to `train_test_split()` or `Kfold()` in Python's sklearn
3. Then we use the `map()` function from `purrr` to grab the data associated with each split using the "assessment" feature. We can then use `compose()` to prepare our rdd's for each split. For now we'll set a constant bw (bw = 0.25), just to test our function.
4. Then we use `map2()` to map our tidy functions and our custom rdd function to each split. **Note**: This operation returns v tibbles instead of a numeric vector, so we bind rows and then grab the mean

```r
#
# 2 - 4: test run: split, map, and map2
# ----------------------------------------------------------------

# set constant bw just to test our function
bw = .25


# 2: split
# ---------
elections_vfold <-
  # create train-test splits
  vfold_cv(elections,    # data
           v = 10,       # folds
           repeats = 1) # repeat


# 3: map
# ---------
elections_vfold <-
  elections_vfold %>%
  # grab the data associated with each split
  mutate(df_split = map(splits, assessment))

# prepare each split
tidy_rdd_model <-
  purrr::compose( # compose multiple functions
  broom::tidy,    # convert lm objects into tidy tibbles
  calculate_rdd_and_bandwidth
)

# 4: map2
# ---------
tidied_models <-
  elections_vfold %>%
```

```r
  # create new rdd variable
  mutate(rdd = map2(df_split, bw, tidy_rdd_model))

# calculate mean and add to dataframe
mean(bind_rows(tidied_models$rdd)$x)
```

```
## [1] 0.2068613
```

5. Now that we know this works for one value of bw, let's loop through 10 values of bw by incrementing from .01 to 1 in a for loop and performing the same operations. We'll return our average mean squared errors to a list and see which one was the lowest.

```r
#
# 5: final run: loop
# ----------------------------------------------------------------

# set seed for replication
set.seed(123456789)

# set bandwidth sequence
# ---------
bw_seq <- seq(.01, 1, .05)

# create empty list and counter
# ---------
mses <- c()
counter = 1

# loop
# ---------
for (bw in bw_seq){

  # 2: split
  # ---------
  elections_vfold <-
    vfold_cv(elections, v = 10, repeats = 1)

  # 3: map
  # ---------
  elections_vfold <-
    elections_vfold %>%
    mutate(df_split = map(splits, assessment))

  tidy_rdd_model <- purrr::compose( # compose multiple functions
    broom::tidy, # convert lm objects into tidy tibbles
    calculate_rdd_and_bandwidth
  )

  # 4: map2
  # ---------
  tidied_models <-
    elections_vfold %>%
    mutate(rdd = map2(df_split, bw, tidy_rdd_model))

  # set counter
```

```r
  mses[counter] <-
    mean(bind_rows(tidied_models$rdd)$x)
  counter = counter + 1
}
```

Now identify the best model with the lowest MSE and identify the bw

```r
# identify the lowest mses and corresponding bandwidth
# ---------
bw_seq %>%
  # bind bw_seq with mses
  bind_cols(mses) %>%
  # rename columns
  rename(bw_seq= `...1`) %>%
  rename(mses = `...2`) %>%
  # sort with min at top
  arrange(mses) %>%
  # take minimum
  slice_head(n = 1) %>%
  print()
```

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
```

```
## # A tibble: 1 x 2
##   bw_seq  mses
##    <dbl> <dbl>
## 1   0.51 0.191
```

QUESTION: What was our lowest bandwidth?

ANSWER: Looks like .1913 was our lowest in this case, corresponding to a bw of .51. Let's check our ATT for this run:

```r
#
# final model with CV bandwidth
# ----------------------------------------------------------------
RDestimate(myoutcomenext ~ difshare, # formula
           data = elections,         # data
           cutpoint = 0,             # specify cutpoint
           bw = .51,                 # specify corresponding bandwidth from CV
           model = TRUE)             # return a model object
```

```
##
## Call:
## RDestimate(formula = myoutcomenext ~ difshare, data = elections,
##     cutpoint = 0, bw = 0.51, model = TRUE)
##
## Coefficients:
##      LATE    Half-BW  Double-BW
##    0.5876     0.5177     0.6474
```