Quant II

Lab 2

Yinxuan Wang

2026-02-12

Homework 1

- Homework 1 due February 16 before class
- Email to **both** Cyrus & me
- Send one PDF, including math, code, and output
- Math part can be scanned and attached (or from tablet)
  LaTeX recommended

Today's Agenda

- Robust standard errors
- Cluster robust standard errors
- Bootstrap

Robust Variance Estimators

- From lecture, asymptotic variance of OLS under unit-level sampling

$$V = \mathbb{E}[X_i X_i']^{-1} \, \mathbb{E}[X_i X_i' \varepsilon_i^2] \, \mathbb{E}[X_i X_i']^{-1}$$

Robust Variance Estimators

- From lecture, asymptotic variance of OLS under unit-level sampling

$$V = \mathbb{E}[X_i X_i']^{-1} \, \mathbb{E}[X_i X_i' \varepsilon_i^2] \, \mathbb{E}[X_i X_i']^{-1}$$

- A general form of estimators for the asymptotic variance:

$$\hat{V}() = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\hat{\Psi}\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

- Where $\hat{\Psi}$ is an estimator of $plim[\varepsilon\varepsilon']$

Robust Variance Estimators

- From lecture, asymptotic variance of OLS under unit-level sampling

$$V = \mathbb{E}[X_i X_i']^{-1} \, \mathbb{E}[X_i X_i' \varepsilon_i^2] \, \mathbb{E}[X_i X_i']^{-1}$$

- A general form of estimators for the asymptotic variance:

$$\hat{V}() = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\hat{\Psi}\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

- Where $\hat{\Psi}$ is an estimator of $plim[\varepsilon\varepsilon']$
- These estimators are "robust" because they don't require specific distributional assumptions

Robust Variance Estimators

From *Mostly Harmless Econometrics*, Ch.8.

- HC0: $\hat{\psi}_i = \hat{e}_i^{\,2}$

Robust Variance Estimators

From *Mostly Harmless Econometrics*, Ch.8.

- HC0: $\widehat{\psi}_i = \widehat{e}_i^{\,2}$
- HC1: $\widehat{\psi}_i = \frac{n}{n-k}\widehat{e}_i^{\,2}$

Robust Variance Estimators

From *Mostly Harmless Econometrics*, Ch.8.

- HC0: $\widehat{\psi}_i = \widehat{e}_i^{\,2}$
- HC1: $\widehat{\psi}_i = \frac{n}{n-k}\widehat{e}_i^{\,2}$
- HC2: $\widehat{\psi} = \frac{1}{1-h_{ii}}\widehat{e}_i^{\,2}$, where h_{ii} is the *leverage* of unit i (the influence of i on the regression line)

Robust Variance Estimators

From *Mostly Harmless Econometrics*, Ch.8.

- HC0: $\widehat{\psi}_i = \widehat{e}_i^{\,2}$
- HC1: $\widehat{\psi}_i = \frac{n}{n-k}\widehat{e}_i^{\,2}$
- HC2: $\widehat{\psi} = \frac{1}{1-h_{ii}}\widehat{e}_i^{\,2}$, where h_{ii} is the *leverage* of unit i (the influence of i on the regression line)
- HC3: $\widehat{\psi} = \frac{1}{(1-h_{ii})^2}\widehat{e}_i^{\,2}$

Linear Regression in R

- In base R, `lm()` implements the OLS estimation
  - Fit the model
  - Then use functions in the package `sandwich` for robust SEs

Linear Regression in R

- In base R, `lm()` implements the OLS estimation
  - Fit the model
  - Then use functions in the package `sandwich` for robust SEs
- With packages `estimatr`, `lfe`, or `fixest`
  - Specify the SE estimator from inside the function

Linear Regression in R

- In base R, `lm()` implements the OLS estimation
  - Fit the model
  - Then use functions in the package `sandwich` for robust SEs
- With packages `estimatr`, `lfe`, or `fixest`
  - Specify the SE estimator from inside the function
- Check the function documentation to see what SEs are computed

Robust Standard Errors in R, i

```r
set.seed(123)
pacman::p_load(tidyverse, estimatr, sandwich, lmtest)

# simulate population
pop <- data.frame(Y1 = rnorm(1000, 4, 2), Y0 = rnorm(1000, 0.5, 3))

# random sample
sample <- pop[sample(nrow(pop), 100),]
sample$D <- 0
sample$D[sample(100, 30)] <- 1

# observed potential outcomes
sample <- sample %>% mutate(Y = D*Y1 + (1-D)*Y0)
```

Robust Standard Errors in R, ii

```r
### Example 1: lm_robust
# Default: HC2 estimator
lm_robust(Y ~ D, data = sample)
```

```
             Estimate Std. Error   t value     Pr(>|t|)   CI Lower CI Upper
(Intercept) 0.3137636  0.3844703 0.8160932 4.164261e-01 -0.4492052 1.076732
D           4.1641379  0.5135960 8.1078083 1.493054e-12  3.1449234 5.183352
```

```r
# HC1
lm_robust(Y ~ D, data = sample, se_type = "HC1")
```

```
             Estimate Std. Error   t value     Pr(>|t|)   CI Lower CI Upper
(Intercept) 0.3137636  0.3855896 0.8137243 4.177757e-01 -0.4514264 1.078954
D           4.1641379  0.5128987 8.1188318 1.414176e-12  3.1463072 5.181969
```

```r
# or
lm_robust(Y ~ D, data = sample, se_type = "stata")
```

```
             Estimate Std. Error   t value     Pr(>|t|)   CI Lower CI Upper
(Intercept) 0.3137636  0.3855896 0.8137243 4.177757e-01 -0.4514264 1.078954
D           4.1641379  0.5128987 8.1188318 1.414176e-12  3.1463072 5.181969
```

Robust Standard Errors in R, iii

```r
### Example 2: lm + lmtest + sandwich
fit <- lm(Y ~ D, data = sample)
summary(fit) # Default is homoskedastic
```

```
Call:
lm(formula = Y ~ D, data = sample)

Residuals:
    Min      1Q  Median      3Q     Max
-5.7960 -2.1130 -0.0758  2.4649  6.0472

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.3138     0.3446   0.910    0.365
D             4.1641     0.6292   6.618 1.95e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.884 on 98 degrees of freedom
Multiple R-squared:  0.3089,    Adjusted R-squared:  0.3018
F-statistic: 43.79 on 1 and 98 DF,  p-value: 1.949e-09
```

```r
N <- nrow(sample)
D <- cbind(rep(1, N), sample$D)
K <- dim(D)[2]
vcov <- solve((t(D) %*% D)) %*% t(D) %*% diag((sum(residuals(fit)^2)/(N-K)), N, N) %*% D %*% solve((t(D) %*% D)
cbind(coef(fit), sqrt(diag(vcov)))
```

```
                [,1]      [,2]
(Intercept) 0.3137636 0.3446480
D           4.1641379 0.6292383
```

Robust Standard Errors in R, iv

```
coeftest(fit, vcov = vcovHC(fit, type = "HC1"))
```

```
t test of coefficients:

            Estimate Std. Error t value  Pr(>|t|)
(Intercept)  0.31376    0.38559  0.8137    0.4178
D            4.16414    0.51290  8.1188 1.414e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coeftest(fit, vcov = vcovHC(fit, type = "HC2"))
```

```
t test of coefficients:

            Estimate Std. Error t value  Pr(>|t|)
(Intercept)  0.31376    0.38447  0.8161    0.4164
D            4.16414    0.51360  8.1078 1.493e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Export Regression Output in Tables - stargazer

```
pacman::p_load(stargazer)
stargazer(fit, type = "text")
```

```
================================================
                        Dependent variable:
                    ---------------------------
                               Y
------------------------------------------------
D                           4.164***
                            (0.629)

Constant                     0.314
                            (0.345)

------------------------------------------------
Observations                  100
R2                           0.309
Adjusted R2                  0.302
Residual Std. Error     2.884 (df = 98)
```

Export Regression Output in Tables - modelsummary

```
pacman::p_load(modelsummary)
modelsummary(list(fit, fit),
             vcov=list("HC1", "HC2"),
             output = "markdown", gof_omit = "^(?!R2|Num)")
```

	①	②
(Intercept)	0.314	0.314
	(0.386)	(0.384)
D	4.164	4.164
	(0.513)	(0.514)
Num.Obs.	100	100
R2	0.309	0.309
R2 Adj.	0.302	0.302

Export Regression Output in Tables - `fixest::etable`

```
pacman::p_load(fixest)
fix <- list(feols(Y ~ D, sample, vcov="iid"),
            feols(Y ~ D, sample, vcov="HC1"))
fix %>% etable()
```

```
                           model 1           model 2
Dependent Var.:                  Y                 Y

Constant             0.3138 (0.3446)   0.3138 (0.3856)
D                  4.164*** (0.6292) 4.164*** (0.5129)
------------------ ----------------- -----------------
S.E. type                        IID Heteroskeda.-rob.
Observations                     100               100
R2                           0.30886           0.30886
Adj. R2                      0.30181           0.30181
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Clustering

- Clustered **sampling**
  - For instance, we sample villages or electoral precincts: individual behavior (e.g. voting) is correlated within cluster

Clustering

- Clustered **sampling**
  - For instance, we sample villages or electoral precincts: individual behavior (e.g. voting) is correlated within cluster
- Clustered **treatment assignment**
  - For instance, we randomize a treatment at the housing block level, assignment is correlated within cluster

Clustering

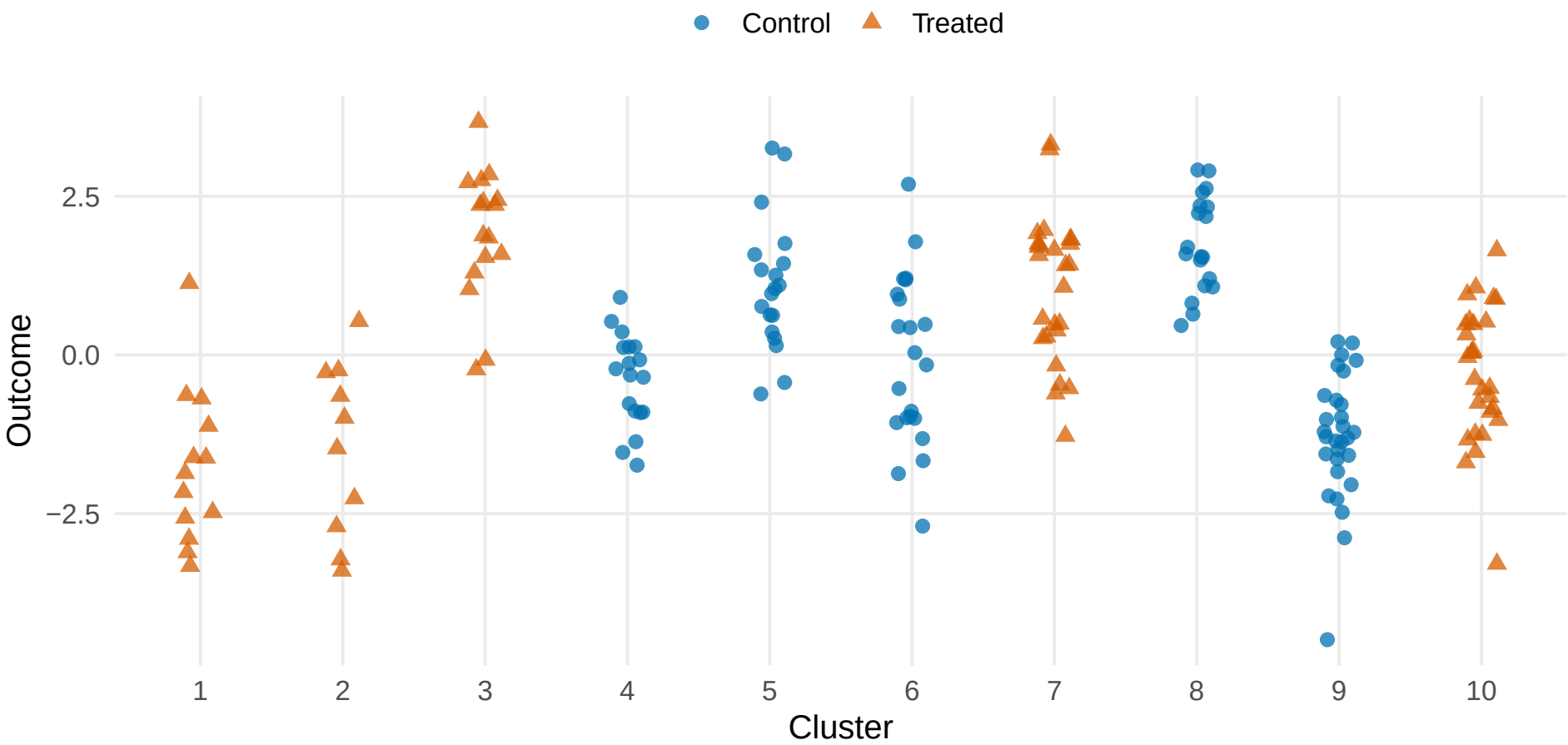
- Clustered **sampling**
  - For instance, we sample villages or electoral precincts: individual behavior (e.g. voting) is correlated within cluster
- Clustered **treatment assignment**
  - For instance, we randomize a treatment at the housing block level, assignment is correlated within cluster
- In both cases, relative to simple random process, units are contributing less information
- If we treat units as independent
  - we overstate the information in the data
  - and underestimate the variance of the estimator

Level of Clustering

A guide to empirical practice - MacKinnon, Nielsen, and Webb (*JE*, 2023)

- In observational work we do not perfectly observe the cluster structure
- "How do you cluster SEs?' '
- Traditional rules of thumb:
  - When in doubt, cluster at the highest possible level
  - Choose the level which gives the highest SEs
- DiD setting: cluster at the unit level
  - Can also cluster in two dimensions if suspects of correlation along other dimensions
- If treatment is assigned at a higher level: cluster at that level
  - E.g. individual-level data, geographic distances at place level

Cluster Simulation

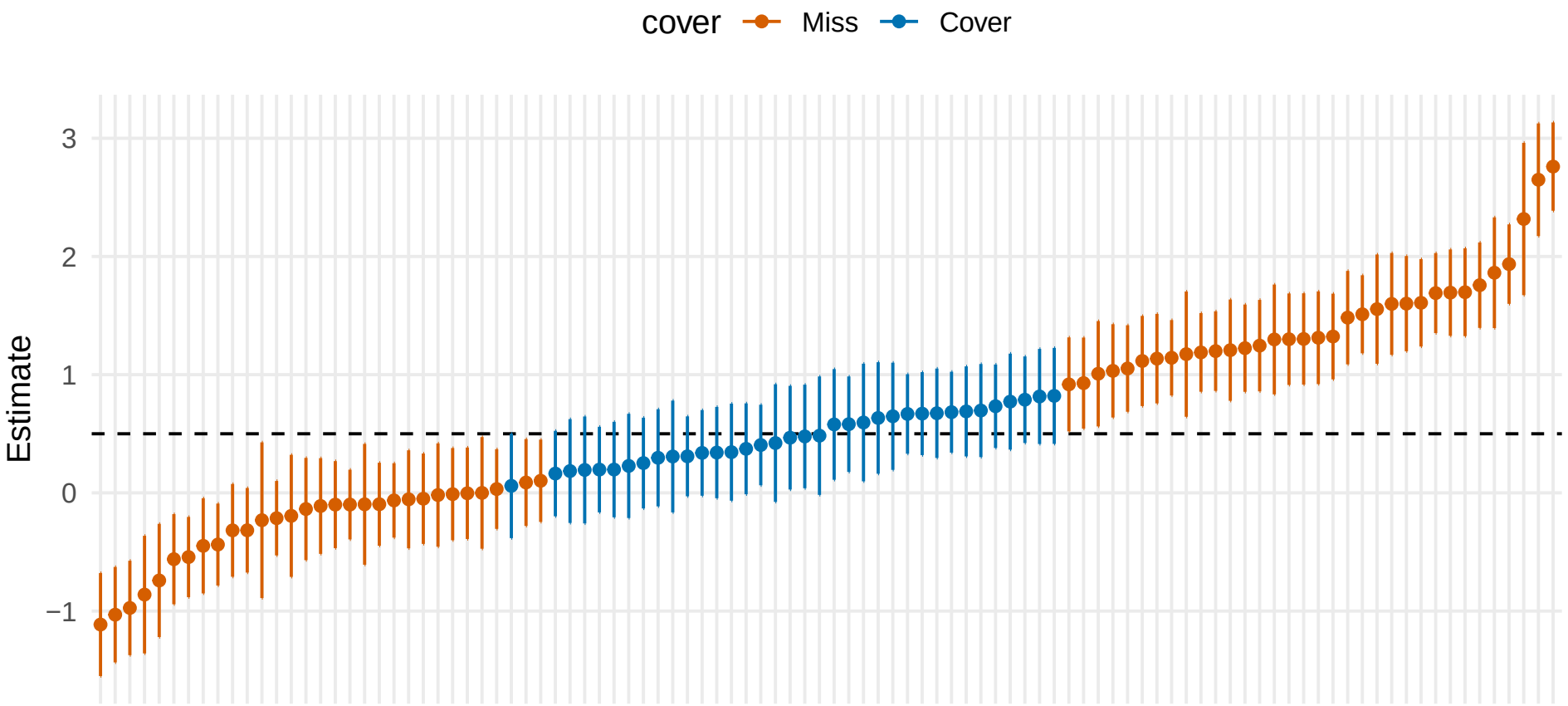


Cluster Simulation, ii

```
set.seed(123)
make_sim <- function(x) {
    df <- make_clustered_data(G = 10, N = 200)
    model <- summary(lm(y ~ treat, data = df))$coefficients["treat", ]
    model
}
res <- map(1:100, make_sim) %>%
    bind_rows() %>%
    mutate(
        lo = Estimate - 1.96 * `Std. Error`,
        hi = Estimate + 1.96 * `Std. Error`,
        cover = lo < 0.5 & hi > 0.5,
        i = 1:100
    )
```

Cluster Simulation, iii

- Only 36% cover the true value



Cluster Simulation - Clustered SEs

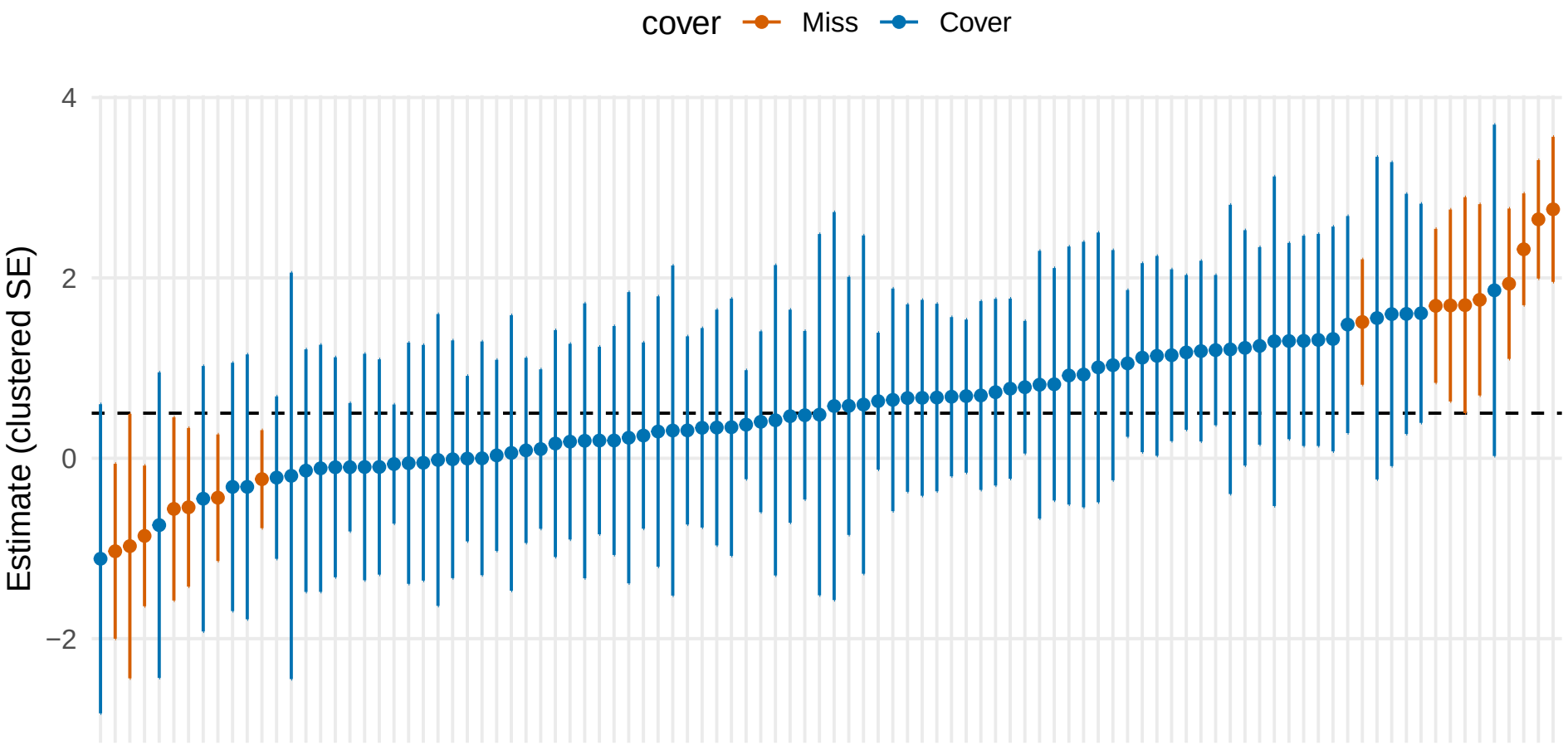
- `lm` + `sandwich`
- `lfe::felm`
- `fixest::feols`

```
set.seed(123)
make_sim_clustered <- function(x) {
  df <- make_clustered_data(G = 10, N = 200)
  m  <- feols(y ~ treat, data = df, cluster = ~ g)

  ct <- coeftable(m)
  out <- ct["treat", c("Estimate", "Std. Error")]
  tibble(Estimate = out[1], `Std. Error` = out[2])
}
```

Cluster Simulation - Clustered SEs, ii

- 84% cover the true value



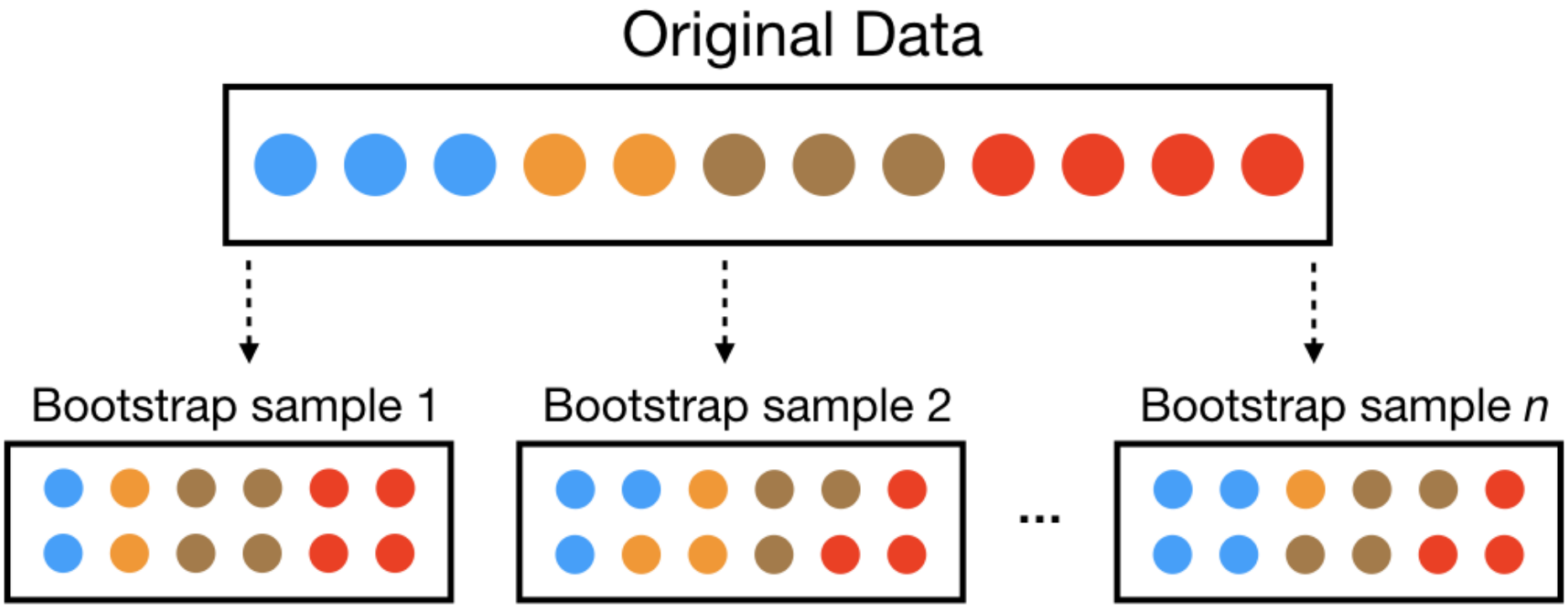
Bootstrapping

- Simulate sampling distributions (from repeated random sampling) using the available sample
- Parametric, Semi-parametric, Non-parametric
- **Non-parametric**: based on re-sampling
- Random re-sampling from the sample approximates random sampling from the population
- The sampling distribution of statistics from re-samples approximates the true sampling distribution

Vanilla Non-parametric Bootstrapping

1. From our sample of size N, draw a random sample of size N with replacement
2. On this sample, compute the estimate
3. Repeat many times (e.g. 1000)
4. Obtain a distribution of estimates from the resamples (a bootstrapped sampling distribution)
5. For inference, compute moments of the bootstrapped distribution

Vanilla Non-parametric Bootstrapping



Cluster Simulation, Bootstrapping

```r
set.seed(123)
df <- make_clustered_data(G = 30, N = 500)
n_clusters <- length(unique(df$g))
```

Cluster Simulation, Vanilla Unit-level Bootstrap

Let's try to bootstrap SEs for β_{treat}

Cluster Simulation, Vanilla Unit-level Bootstrap

Let's try to bootstrap SEs for β_{treat}

```
vanilla_bootstrap <- function(a) {
    resampled.data <- df %>% sample_frac(1, replace = T)
    broom::tidy(lm(y ~ treat, data = resampled.data)) %>%
      dplyr::filter(term == "treat")
}
vb.results <- map(1:1000, vanilla_bootstrap) %>%
    bind_rows()
sd(vb.results$estimate)
```

```
[1] 0.1325565
```

Cluster Simulation, Vanilla Unit-level Bootstrap

Let's try to bootstrap SEs for β_{treat}

```
vanilla_bootstrap <- function(a) {
    resampled.data <- df %>% sample_frac(1, replace = T)
    broom::tidy(lm(y ~ treat, data = resampled.data)) %>%
      dplyr::filter(term == "treat")
}
vb.results <- map(1:1000, vanilla_bootstrap) %>%
    bind_rows()
sd(vb.results$estimate)
```

```
[1] 0.1325565
```

```
tidy(summary(lm(y ~ treat, data = df)))
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic   p.value
  <chr>          <dbl>     <dbl>     <dbl>     <dbl>
1 (Intercept)  -0.0102     0.114   -0.0900 0.928
2 treat         0.560      0.141    3.98   0.0000799
```

Cluster Simulation, Vanilla Cluster Bootstrap

- Bootstrap procedure with clustered data needs to reflect the clustered structure
- Let's resample at the **cluster** level

Cluster Simulation, Vanilla Cluster Bootstrap

- Bootstrap procedure with clustered data needs to reflect the clustered structure
- Let's resample at the **cluster** level

```
get_cluster_bs_sample <- function(df) {
  clusters <- unique(df$g)
  sampled_clusters <- sample(clusters, size = n_clusters, replace = TRUE)
  dplyr::bind_rows(lapply(sampled_clusters, function(cl) df[df$g == cl, ]))
}

estimates <- purrr::map_dbl(1:1000, function(b) {
  boot_df <- get_cluster_bs_sample(df)
  coef(lm(y ~ treat, data = boot_df))["treat"]
})

sd(estimates)
```

```
[1] 0.4648703
```

```
tidy(summary(feols(y ~ treat,data =df,cluster =~ g)))
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)  -0.0102     0.353   -0.0290   0.977
2 treat         0.560      0.459    1.22     0.232
```

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\hat{\beta}$ and t

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\hat{\beta}$ and t
- Fit the restricted model, obtain residuals

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\hat{\beta}$ and t
- Fit the restricted model, obtain residuals
- **Randomize the sign of the residual** at the **cluster level** (Rademacher weights)

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\hat{\beta}$ and t
- Fit the restricted model, obtain residuals
- **Randomize the sign of the residual** at the **cluster level** (Rademacher weights)
- Construct $Y_b^\star$ as restricted fitted values + randomized residuals

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\widehat{\beta}$ and t
- Fit the restricted model, obtain residuals
- **Randomize the sign of the residual** at the **cluster level** (Rademacher weights)
- Construct $Y_b^\star$ as restricted fitted values + randomized residuals
- Estimate $\widehat{\beta}_b$

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\widehat{\beta}$ and t
- Fit the restricted model, obtain residuals
- **Randomize the sign of the residual** at the **cluster level** (Rademacher weights)
- Construct $Y_b^\star$ as restricted fitted values + randomized residuals
- Estimate $\widehat{\beta}_b$
- Compute t-statistic, t_b, for the new estimate

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\widehat{\beta}$ and t
- Fit the restricted model, obtain residuals
- **Randomize the sign of the residual** at the **cluster level** (Rademacher weights)
- Construct $Y_b^\star$ as restricted fitted values + randomized residuals
- Estimate $\widehat{\beta}_b$
- Compute t-statistic, t_b, for the new estimate
- Do this `N_boot` times

Wild Cluster Bootstrap-t

Let's not draw conventional bootstrap samples

Instead,

- Fit the unrestricted model, estimate $\widehat{\beta}$ and t
- Fit the restricted model, obtain residuals
- **Randomize the sign of the residual** at the **cluster level** (Rademacher weights)
- Construct $Y_b^\star$ as restricted fitted values + randomized residuals
- Estimate $\widehat{\beta}_b$
- Compute t-statistic, t_b, for the new estimate
- Do this `N_boot` times
- Compute bootstrap p-values by counting the share of simulated t_b to the left/right of the observed t

Cluster Simulation, Wild Cluster Bootstrap-t

```r
pacman::p_load(fwildclusterboot)
# install.packages(c("dqrng", "collapse", "JuliaConnectoR", "gtools"))

feols_fit <- feols(y ~ treat, data = df)
boot <- fwildclusterboot::boottest(
  feols_fit,
  clustid = "g",
  param   = "treat",
  B       = 9999,
  type    = "rademacher",
  conf_int = TRUE,
  sign_level = 0.05
)
summary(boot)
```

```
boottest.fixest(object = feols_fit, param = "treat", B = 9999,
    clustid = "g", sign_level = 0.05, conf_int = TRUE, type = "rademacher")

 Hypothesis: 1*treat = 0
 Observations: 500
  Bootstr. Type: rademacher
 Clustering: 1-way
 Confidence Sets: 95%
 Number of Clusters: 30


         term estimate statistic p.value conf.low conf.high
1 1*treat = 0     0.56      1.22   0.253   -0.416     1.588
```

R Packages for Bootstrapping

- `boot`
- `ClusterBootstrap` - bootstrapping on hierarchically structured data
- `fwildclusterboot` - wild cluster bootstrap
- `multiwaycov::cluster.boot` for post-estimation SE calculation