Seminar 7

Natalia Lamberova

11/10/2016

Final Project Info

Full draft including DeclareDesign is due on ***November 23***
Presentations: November 30 and December 2
Final Draft of Registered Report: December 6
Final Draft of the analysis: December 16

Important Dates

Task	Date	Time
Research Question	November 18	
Design in Words	November 18	6 pm
Measurement Strategy	November 23	12 pm
DeclareDesign & Diagnosis	November 23	12 pm
Ethical Issues	November 23	12 pm
Registered report	December 6	6 pm
Fake data delivered	December 7	
Results	December 16	6 pm
Discussion	December 16	6 pm

Setting up the design in Declare Design

```r
library(pacman)
p_load(DeclareDesign,knitr)
#Here we've loaded the needed packages and setting the seed
set.seed(1234)
```

Data Simulations:

Data Simulations: Multivariate:

```r
multivariate_pop <- declare_population(
  income = declare_variable(type = "normal",
                            location_scale = c(mean = 1000, sd = 10)),
  gender = declare_variable(type = "gender"),
  party = declare_variable(type = "us_party"),
  employed = declare_variable(type = "binary"),
  size = 500
)
```

```
multivariate_draw <- draw_population(population = multivariate_pop)
head(multivariate_draw)
```

```
##   level_A_ID    income gender party employed
## 1          1  987.9293      M   ind        0
## 2          2 1002.7743      F   ind        0
## 3          3 1010.8444      F   ind        1
## 4          4  976.5430      F   dem        1
## 5          5 1004.2912      M   dem        0
## 6          6 1005.0606      F   dem        1
```

Data Simulations: Linear Otcomes

If you wish to simply create linear potential outcomes, you must specify the different conditions, and include a function that specifies how they are generated in a linear model.

```
# Make a very simple sample frame
simple_frame <- declare_population(one_level = list(),size = 10)
# Make a very simple assignment scheme
condition_names <- c(0,1)
assignment <- declare_assignment(condition_names = condition_names)
# Make the draw
pop_draw <- draw_population(population = simple_frame)
pop_draw <- assign_treatment(data = pop_draw,
                                assignment = assignment)

outcomes_1 <- declare_potential_outcomes(
  # Specify the conditions of the experiment
  condition_names = condition_names,
  # Here the average treatment effect is .5
  formula = outcome ~ .5*Z
    )


linear_draw <- draw_outcome(data = pop_draw,
                            potential_outcomes = outcomes_1)
head(linear_draw)
```

```
##   one_level_ID Z Z_assignment_probabilities Z_assignment_weights outcome
## 1            1 1                        0.5                    2     0.5
## 2            2 1                        0.5                    2     0.5
## 3            3 0                        0.5                    2     0.0
## 4            4 1                        0.5                    2     0.5
## 5            5 1                        0.5                    2     0.5
## 6            6 0                        0.5                    2     0.0
```

Non-linear Outcomes:

```
# For a probit data-generating process, write a predictive probit function
probit_outcome <- function(Z){
  rbinom(n = length(Z), size = 1, prob = pnorm(q = Z*.2))
}
```

```r
condition_names <- c(0,1)
# Insert the function into the outcome formula
probit_PO <- declare_potential_outcomes(
  condition_names = condition_names,
  formula = Y ~ probit_outcome(Z)
)

# Make data
probit_draw <- draw_outcome(data = pop_draw,
                            potential_outcomes = probit_PO)
head(probit_draw)
```

```
##   one_level_ID Z Z_assignment_probabilities Z_assignment_weights Y
## 1            1 1                        0.5                    2 1
## 2            2 1                        0.5                    2 1
## 3            3 0                        0.5                    2 1
## 4            4 1                        0.5                    2 0
## 5            5 1                        0.5                    2 1
## 6            6 0                        0.5                    2 0
```

Poisson Process:

Think about witing for the bus. λ is the event rate (average number of events per interval). So the Poisson process look like this:

$$P(kEventsInInterval) = \frac{\lambda e^{-\lambda}}{k!}$$

```r
# For a Poisson data-generating process, write a predictive poisson function
poisson_outcome <- function(Z){
  rpois(n = length(Z), lambda = exp(Z*.2))
}

# Insert the function into the outcome formula
pois_PO <- declare_potential_outcomes(
  condition_names = condition_names,
  formula = Y ~ poisson_outcome(Z)
)

pois_draw <- draw_outcome(data = pop_draw,
                          potential_outcomes = pois_PO)
head(pois_draw)
```

```
##   one_level_ID Z Z_assignment_probabilities Z_assignment_weights Y
## 1            1 1                        0.5                    2 0
## 2            2 1                        0.5                    2 3
## 3            3 0                        0.5                    2 1
## 4            4 1                        0.5                    2 1
## 5            5 1                        0.5                    2 0
## 6            6 0                        0.5                    2 0
```

Changes in proportions:

Simple Changes in Proportions You may want to specify potential outcomes at the group-level, especially when the outcome is binary or categorical. For example, we might want to specify that the treatment increases uptake of a service by 5%, or we might want to specify that it increases the probability of voting for candidate A by 10%, while reducing that of voting for B and C by 6% and 4%, respectively. Binary case in presented below:

```
success_proportions <- c(control = .3, treatment = .8)

proportion_PO_1 <- declare_potential_outcomes(
  potential_outcomes_function = proportion_potential_outcomes_function,
  condition_names = condition_names,
  assignment_variable_name = "Z",
  outcome_variable_name = "Y",
  population_proportions = success_proportions)

prop_draw_1 <- draw_outcome(data = pop_draw,
                            potential_outcomes = proportion_PO_1)
head(prop_draw_1)
```

```
##   one_level_ID Z Z_assignment_probabilities Z_assignment_weights Y
## 1            1 1                        0.5                    2 1
## 2            2 1                        0.5                    2 1
## 3            3 0                        0.5                    2 0
## 4            4 1                        0.5                    2 1
## 5            5 1                        0.5                    2 1
## 6            6 0                        0.5                    2 0
```

Multilevel:

```
N_per_level <- c(students = 8,
                 classrooms = 4,
                 schools = 2)

# Note: level names are inhereted from the variable declarations, not
# from the the N_per_level vector

school_pop <- declare_population(
 student = list(
   math_score = declare_variable(type = "normal",
     location_scale = c(10,5)),
   lang_score = declare_variable(type = "normal",
     location_scale = c(10,5))
 ),
 classroom = list(
   N_students_raw = declare_variable(
     type = "normal",
     location_scale = c(20,1)),
   N_students = "round(N_students_raw,0)"
 ),
 school = list(
   poor_area = declare_variable(type = "binary")
 ),
 size = N_per_level
)

school_draw <- draw_population(population = school_pop)
head(school_draw)
```

```
##   student_ID classroom_ID school_ID poor_area N_students_raw N_students
## 1          1            1         1         0       20.56014         21
## 2          2            1         1         0       20.56014         21
## 3          3            2         1         0       20.15815         20
## 4          4            2         1         0       20.15815         20
## 5          5            3         2         1       18.83583         19
## 6          6            3         2         1       18.83583         19
##   math_score lang_score
## 1  14.295867   2.394438
## 2  10.252741  17.423469
## 3  21.544721  15.168517
## 4   5.702684  16.856332
## 5  17.695581   6.955393
## 6   1.927669  10.427985
```

A Six-Step Workflow for Experiments with Custom Functions

Custom population function just selects N random numbers.

A population function has to create a data frame whenever it is run. It can optionally have inputs for N, N_per_level, or group_sizes_per_level (but not more than one) to indicate the sample size. In this example, the input N is used and set in the declaration to 100 (note this allows you to reuse my_pop_function to test out designs with differing sample sizes).

```
N<-100
my_population <- function(size) { data.frame(income = rnorm(size)) }
population      <- declare_population(
  custom_population_function = my_population,
  size = N)
```

Declaring potential outcomes requires that you declare both the ***treatment levels*** and a ***potential outcomes*** formula,ie a statement that maps from possible treatment conditions to Ys.It must indicate what value Y takes for all values that a treatment might take.

This formula and the condition names defined in the declaration place restrictions on the remaining workflow, by setting the outcome name to Y and the names of the conditions. When potential outcomes are constructed in mock data, they will be named in this case Y_Z1 and Y_Z0. When realized outcomes based on a treatment assignment are created, they will be named in this case Y. "'

Potential Outcomes:

```
my_potential_outcomes <- function(data) {
  (data$Z == "Z1") * 0.25 + runif(nrow(data)) }
potential_outcomes <- declare_potential_outcomes(
  condition_names = c("Z0", "Z1"),
  potential_outcomes_function = my_potential_outcomes,
  outcome_variable_name = "Y")
```

A sampling function then takes a data frame and indicates which rows to select.

```
# Taking half the units.

my_sampling    <- function(data) {
  N <- nrow(data); n <- floor(N/2);
  sample(rep(c(0, 1), each = N-n), N) }
sampling       <- declare_sampling(
  custom_sampling_function = my_sampling)
```

An estimand function takes a data frame with potential outcome columns and returns a scalar.

```
#Estimand - Difference in means

my_estimand    <- function(data) {
  mean(data$Y_Z_Z1 - data$Y_Z_Z0) }
estimand       <- declare_estimand(
  estimand_function = my_estimand,
  potential_outcomes = potential_outcomes)
```

An assignment function takes a data frame and returns a vector of treatment assignments

```
my_assignment  <- function(data) {
  N <- nrow(data);
  sample(c("Z0", "Z1"),
         N, replace = T) }
assignment     <- declare_assignment(
  custom_assignment_function = my_assignment,
  potential_outcomes = potential_outcomes)
```

An estimates function

takes data and returns a matrix with columns representing estimates and rows representing statistics of the estimates, such as the estimate itself, the standard error, the p-value, etc.

```
my_estimates   <- function(data) {
  est       <- mean(data$Y[data$Z == "Z1"])
  - mean(data$Y[data$Z == "Z0"])
  se        <- sqrt(var(
  data$Y[data$Z == "Z1"])/sum(data$Z == "Z1") +
  var(data$Y[data$Z == "Z0"])/sum(data$Z == "Z0"))
  df        <- nrow(data) - 2
  p         <- 2 * pt(abs(est/se), df = df, lower.tail = FALSE)
  ci_lower <- est - 1.96*se
  ci_upper <- est + 1.96*se
  data.frame(estimate_label = "diff-in-means",
             est = est, se = se, p = p,
             ci_lower = ci_lower, ci_upper = ci_upper, df = df,
             stringsAsFactors = FALSE)
}
estimator      <- declare_estimator(
  estimates = my_estimates,
  estimand = estimand)
```

Full design in six steps.

```
my_design      <- declare_design(
  population = population,
  sampling = sampling,
  potential_outcomes = potential_outcomes,
  assignment = assignment,
  estimator = estimator,
  label = "my-design")
```

What next?

```
#Constructing mock data
mock_population <- draw_population(
  population = population,
  potential_outcomes = potential_outcomes)
mock_sample <- draw_sample(
  data = mock_population,
  sampling = sampling)
mock_sample <- assign_treatment(
  data = mock_sample,
  assignment = assignment)
mock_sample <- draw_outcome(
  data = mock_sample,
  potential_outcomes = potential_outcomes)
kable(head(mock_sample), digits = 3,
      row.names = FALSE)
```

income	Y_Z_Z0	Y_Z_Z1	Z	Y
-0.655	0.491	0.440	Z1	0.440
-0.701	0.231	1.152	Z1	1.152
0.516	0.090	0.816	Z0	0.090
1.051	0.703	1.230	Z1	1.230
0.367	0.085	0.523	Z0	0.085
0.445	0.441	0.905	Z0	0.441

Diagnosis

You can define a custom statistic function which takes as options estimates, estimators, or both and returns a scalar summary of them. Here, we calculate the proportion of estimates that are larger than zero.

```r
my_statistic <- function(estimates, ...) {
  mean(sapply(1:length(estimates), function(i)
  as.numeric(estimates[[i]]["p", , drop = FALSE])) > 0.5) }

diagnosis <- diagnose_design(design = my_design,
                             population_draws = 10,
                             sample_draws = 10)

kable(summary(diagnosis, statistics = list(
  calculate_PATE, calculate_sd_SATE, calculate_power,
  calculate_RMSE, calculate_bias, calculate_coverage,
  my_statistic), labels = c("PATE", "sd(SATE)", "power",
                            "RMSE", "bias", "coverage",
                           "custom_stat")), digits = 3)
```

estimand_label	estimand_level	estimator_label	estimate_label	diagnosand	diagnosand_label
estimand	population	estimator	diff-in-means	0.256	mean(estimand)
estimand	population	estimator	diff-in-means	0.760	mean(estimate)
estimand	population	estimator	diff-in-means	0.056	sd(estimate)
estimand	population	estimator	diff-in-means	0.504	bias
estimand	population	estimator	diff-in-means	0.508	RMSE
estimand	population	estimator	diff-in-means	0.000	coverage
estimand	population	estimator	diff-in-means	1.000	power
estimand	population	estimator	diff-in-means	0.000	type S rate

Interpreting Diagnosands

Bias
If we ran the experiment again and again, will the average of our estimates exactly equal our estimand?

$$\text{bias} \equiv E(\hat{\tau} - \tau)$$

Type S error
"the probability that the estimate has the incorrect sign, if it is statistically significantly different from zero"

Note this interprets an estimate of τ that is not statistically significant as not an exaggeration

Gelman and Carlin, "Beyond Power Calculations"

RMSE
How noisy is my estimate of the estimand?

$$\text{RMSE} \equiv \sqrt{E(\hat{\tau} - \tau)^2}$$

Coverage
Are the standard errors a good estimate of the uncertainty of my estimates, or are they too conservative or liberal?

$$E(\text{CI}_- \leq \tau \leq \text{CI}_+) \to^p 1 - \alpha$$

"Nominal coverage" for $\alpha = .05$ is 0.95: the confidence interval should contain the estimand 95 out of 100 times the experiment is run

When the coverage probability is below .95, the confidence intervals are too narrow (i.e. the standard errors are too small).

Power
How often will I reject the null hypothesis when it is false?

Reference value of power = 0.8

Simulating results

```
mock_population <- draw_population(population = population)
mock_population <- assign_treatment(
  data = mock_population,
  assignment = assignment)
mock_population <- draw_outcome(data =
                                  mock_population,
                                potential_outcomes =
                                  potential_outcomes)
kable(get_estimands(estimator = estimator,
              data = mock_population), digits = 3)
```

estimator_label	estimand_label	estimand_level	estimand
estimator	estimand	population	0.24

Communicating results

```
kable(get_estimates(estimator = estimator, data = mock_population), digits = 3)
```

estimate_label	est	se	p	ci_lower	ci_upper	df	estimator_label	estimand_label	estimand_level
diff-in-means	0.722	0.061	0	0.602	0.841	98	estimator	estimand	population