[Graeme Blair](https://graemeblair.com/" \t "_blank) is an Assistant Professor of Political Science at UCLA.

[Jasper Cooper](http://jasper-cooper.com/) is a Postdoctoral Research Associate at the Kahneman-Treisman Center for Behavioral Science and Public Policy at Princeton University.

[Alexander Coppock](https://alexandercoppock.com/) is an Assistant Professor of Political Science at Yale University.

[Macartan Humphreys](http://www.macartan.nyc/) is a Professor of Political Science at Columbia University and a Director of the research group “Institutions and Political Inequality” at the WZB Berlin Social Science Center.

Research design consists of a set of choices about what research *procedures* to use. For example, how many subjects to interview, which questions to ask them, and what to do in the analysis phase with the data that results from these choices. We do not have good tools for assessing whether the chosen procedures are good ones. [DeclareDesign](https://declaredesign.org/r/declaredesign) is an R package for learning about, implementing, and communicating research procedures, from data collection to data analysis.

Today, most data scientists take raw data as the starting point for analysis, and consider how best to import, tidy, transform, visualize, model, and communicate their results. Yet often during this process, we learn that our data can only provide limited answers to our questions. Perhaps the sample size is too small, we do not have enough observations from men and women to estimate gender differences, or we did not collect data on units’ geographic location, so we cannot merge to administrative data. We rely on rules of thumb or expert opinion to navigate the myriad research design choices we face. We select analysis strategies tailored to the data that we have before us rather than the data that could have arisen. In doing so, we risk introducing biases. Our practices can often lead to bad inferences and bad science.

To learn whether a research procedure is a good one, we need to know what the results *would have been* had the data turned out differently. With that knowledge, we can assess our data analysis procedures in terms of whether they will yield answers that are unbiased and precise. DeclareDesign helps us simulate different data sets that could result from our data collection procedures. With these simulated data, we can ask the question: is our set of procedures likely to yield unbiased, low-variance answers to our research questions? We can assess not only whether we should select difference-in-means or regression as our estimator, but also whether it would have been better to assign half of subjects to treatment in our A/B test, or more than half.

We’ve built a set of linked tools that help with each step of the research design process. In this post, we’ll talk about using DeclareDesign to declare and diagnose the properties of a full set of procedures. We’ll also introduce [DesignLibrary](https://declaredesign.org/library/), which helps you quickly get started learning about your research design for a set of common designs. In the coming weeks, we’ll talk about tools for simulating data ([fabricatr](https://declaredesign.org/r/fabricatr" \t "_blank)), sampling cases and assigning A/B tests ([randomizr](https://declaredesign.org/r/randomizr" \t "_blank)), and estimating effects ([estimatr](https://declaredesign.org/r/estimatr/" \t "_blank)).

library(DeclareDesign)

library(tidyverse)

**A grammar of research designs**

DeclareDesign implements a set of ideas laid out in [our paper](https://declaredesign.org/declare.pdf), forthcoming at the *American Political Science Review*. We think of any research design as having four parts:

1. A **M**odel of the world.
2. An **I**nquiry about that model.
3. A **D**ata strategy according to which data will be collected or brought into existence.
4. An **A**nswer strategy according to which the data will be summarized to generate a guess about the inquiry.

Writing designs down is hard, and DeclareDesign can make that process simpler, more explicit, and reproducible.

**Declaration Steps**

In DeclareDesign, you declare each component of a research design using one of the declare\_\* functions.

| **declare\_\* function** | **MIDA** | **Description** |
| --- | --- | --- |
| declare\_population() | M | Pre-treatment covariates |
| declare\_potential\_outcomes() | M | Relationships between treatments, covariates, and outcomes |
| declare\_estimand() | I | Quantity of interest |
| declare\_sampling() | D | (Possibly random) sampling procedure |
| declare\_assignment() | D | (Possibly random) assignment procedure |
| declare\_reveal() | D | Function that maps potential outcomes to realized outcomes |
| declare\_estimator() | A | Estimation (and hypothesis testing, if applicable) procedure |

Each declare\_\* function is a “function factory”, which means that each of these functions themselves returns a function. Each function takes data as an argument, and returns either data or statistics. Many R users are not used to having functions return functions, but this makes simulation very easy, and is not so odd once you get used to it. In the example below, we define the pop function using declare\_population(). Each time we call the resulting pop function, we get a slightly different data set.

pop <- declare\_population(N = 3, X = rnorm(N))

pop()

ID X

1 1 -2.120337

2 2 -1.156013

3 3 1.703992

pop()

ID X

1 1 -0.6172369

2 2 0.3183334

3 3 -1.3762445

**Chaining steps together**

A research design can be thought of as a series of steps. In DeclareDesign, we chain steps together using the + operator, which may be familiar to ggplot2 users. Each step takes the data generated in the step before it, and either returns statistics or passes that data on to the next step, having added new variables or changed the shape of the data. In the next example, we declare a workhorse research strategy used in many scientific fields, the two-arm randomized trial:

design <-

# M -- Model

declare\_population(N = 100, X = rpois(N, 4), noise = rnorm(N)) +

declare\_potential\_outcomes(Y ~ 0.2 \* Z + 0.5 \* X + noise) +

# I -- Inquiry

declare\_estimand(ATE = mean(Y\_Z\_1 - Y\_Z\_0)) +

# D -- Data Strategy

declare\_sampling(n = 50) +

declare\_assignment(m = 25) +

declare\_reveal(Y, Z) +

# A -- Answer Strategy

declare\_estimator(Y ~ Z, model = difference\_in\_means, estimand = "ATE", label = "DIM")

Some points to note about this declaration:

1. design is a “design object,” which can then be used to learn all kinds of things about the design. We’ll turn to some common post-declaration functions in a moment.
2. “Potential outcomes” is a concept from the causal inference literature. Potential outcomes are the outcomes that each unit would express depending on the counterfactual level of some treatment. The revealed outcome is the **observed** outcome: treated subjects reveal their treated outcome and untreated subjects reveal their untreated outcome. The potential outcomes model of causality is deep. For users who are new to thinking in terms of potential outcomes, we promise it will get easier with time. [The Wikipedia article](https://en.wikipedia.org/wiki/Rubin_causal_model) is OK, but we would suggest [Chapter 2 of this book](https://www.amazon.com/Field-Experiments-Design-Analysis-Interpretation/dp/0393979954) if it’s available to you.
3. In the code above, the estimand and estimator steps are explicitly linked together – we define the average treatment effect (ATE) in terms of potential outcomes, and then in the estimation step, we tell the estimator that it is shooting at the ATE. This means that a design “knows” what question each estimator is trying to answer.
4. These declaration functions depend on the other packages in the DeclareDesign set of packages, which includes fabricatr, randomizr, and estimatr. We’ll introduce each of these packages in upcoming posts. The population and potential outcomes steps use fabricatr, the sampling and assignment use randomizr, and the estimator steps use estimatr.

**What can you do with a design object?**

The first thing you might want to do with a design object is draw\_data. Take a look:

dat <- draw\_data(design)

head(dat)

ID X noise Y\_Z\_0 Y\_Z\_1 S\_inclusion\_prob Z Z\_cond\_prob

1 001 3 -0.09946134 1.400539 1.600539 0.5 1 0.5

2 002 3 0.42156668 1.921567 2.121567 0.5 1 0.5

3 003 3 0.24064949 1.740649 1.940649 0.5 0 0.5

4 005 5 -1.29990991 1.200090 1.400090 0.5 1 0.5

5 007 4 1.38503254 3.385033 3.585033 0.5 1 0.5

6 008 3 0.16181031 1.661810 1.861810 0.5 1 0.5

Y

1 1.600539

2 2.121567

3 1.740649

4 1.400090

5 3.585033

6 1.861810

You can quickly generate simulated data that (if you did your declaration right!) is the same size and shape as the data your study **will** generate, once it’s run. Imagining your data ex ante gives you the chance to think about logistical, practical, theoretical, or analytic concerns that might otherwise be put off until far later in the design process.

Next up, you can take a look a the estimands and estimates produced by one run of your design:

draw\_estimands(design)

estimand\_label estimand

1 ATE 0.2

draw\_estimates(design)

estimator\_label term estimate std.error statistic p.value conf.low

1 DIM Z 0.1302799 0.3997006 0.3259437 0.7459163 -0.6738506

conf.high df outcome estimand\_label

1 0.9344104 46.9193 Y ATE

These functions give you a chance to ensure that your estimands and estimators make sense and are returning what you expect them to return. If a 0.2 average treatment effect is implausible in your research setting, change it!

You can also add steps to a design. Here, we’re adding another estimator that **also** shoots at the ATE, but this time adjusts for a pre-treatment covariate, X:

design <- design + declare\_estimator(Y ~ Z + X, model = lm\_robust, estimand = "ATE", label = "OLS")

**Simulating designs**

DeclareDesign has two functions for conducting simulations: simulate\_design and diagnose\_design. If your answer strategy returns estimates, \(p\)-values, and standard errors, and it is linked to an estimand, diagnose\_design can tell you a lot about your design: its power, its coverage probability, its “true” standard error (the standard deviation of the estimates across repeat samples), the average standard error it returns, its bias, its root mean square error, the probability you get the sign of the estimand wrong, its false positive rate, and more. If you just want to simulate the design, simulate\_design will return a data set in which each row is an estimator-estimand pair. Since we have two estimators, each run of the design produces two rows.

simulations <- simulate\_design(design, sims = 500)

head(simulations)

design\_label sim\_ID estimand\_label estimand estimator\_label term

1 design 1 ATE 0.2 DIM Z

2 design 1 ATE 0.2 OLS Z

3 design 2 ATE 0.2 DIM Z

4 design 2 ATE 0.2 OLS Z

5 design 3 ATE 0.2 DIM Z

6 design 3 ATE 0.2 OLS Z

estimate std.error statistic p.value conf.low conf.high

1 -0.73149295 0.4594131 -1.5922335 0.11789917 -1.655212062 0.1922262

2 -0.04367092 0.3435135 -0.1271301 0.89937977 -0.734730962 0.6473891

3 0.23917794 0.4454914 0.5368856 0.59384801 -0.656742583 1.1350985

4 0.62821514 0.3105686 2.0227904 0.04880466 0.003431732 1.2529986

5 0.27936681 0.3567281 0.7831365 0.43758754 -0.438821047 0.9975547

6 0.04149547 0.2894608 0.1433544 0.88662317 -0.540824504 0.6238154

df outcome

1 47.98565 Y

2 47.00000 Y

3 47.58893 Y

4 47.00000 Y

5 45.69013 Y

6 47.00000 Y

The simulations data can then be used to summarize the design using ggplot2 and dplyr. For example, simulation is useful to understand the **distribution** of effect estimates; are they centered on the estimand? Is the distribution wide or narrow? Are there outliers that might indicate estimation issues?

summary\_df <-

simulations %>%

group\_by(estimator\_label) %>%

summarize(`Mean Estimate` = mean(estimate),

`Mean Estimand` = mean(estimand)) %>%

gather(key, value, `Mean Estimand`, `Mean Estimate`)

ggplot(simulations, aes(estimate)) +

geom\_histogram(bins = 30) +

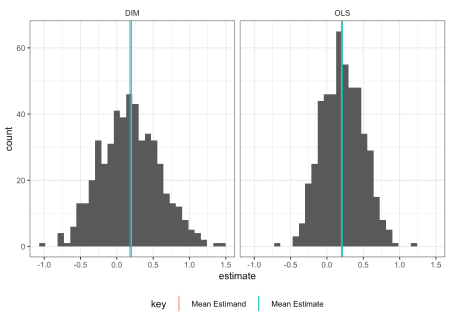
geom\_vline(data = summary\_df, aes(xintercept = value, color = key)) +

facet\_wrap( ~ estimator\_label) +

theme\_bw() +

theme(strip.background = element\_blank(),

legend.position = "bottom")



**Getting started quickly with DesignLibrary**

You can get started by declaring a design in a single line of code with DesignLibrary.

library(DesignLibrary)

two\_arm\_design <- two\_arm\_designer(N = 1000)

DesignLibrary contains 15 common research designs that can be declared in this simple  
way. In each case, a set of function arguments are set up to define the core inputs to the design, such as sample size. For example, we can declare a pretest-posttest design in which data is collected before (pre) and after (post) a treatment is assigned. We define the sample size (N), the average treatment effect (ate), and the correlation in outcomes between the pre and post periods (rho):

prepost\_design <- pretest\_posttest\_designer(N = 1000, ate = 5, rho = 0.25)

**Using DeclareDesign in your data science workflow**

Since the four helper packages can be used to simulate data, sample units, randomize treatments, and analyze data, we see at least seven ways you can get value out of using DeclareDesign in your everyday data-science workflow.

1. Simulate mock data (draw\_data()).
2. Assess the properties of your design before you run it (diagnose\_design()).
3. Front-load design decisions before data is collected (+ to declare it).
4. Compare the properties of several possible designs (compare\_designs()).
5. Write a preanalysis plan or registered report (summary(design) to get started).
6. Describe your design in code in a paper or report (print\_code(design)).
7. Learn about properties of someone else’s design (diagnose\_design()).

Finally, you may find the [DeclareDesign cheatsheet](https://github.com/rstudio/cheatsheets/blob/master/declaredesign.pdf) hosted by RStudio helpful.