Package ‘DeclareDesign’

January 10, 2023

Title Declare and Diagnose Research Designs

Version 1.0.2

Description Researchers can characterize and learn about the properties of research designs before implementation using ‘DeclareDesign’. Ex ante declaration and diagnosis of designs can help researchers clarify the strengths and limitations of their designs and to improve their properties, and can help readers evaluate a research strategy prior to implementation and without access to results. It can also make it easier for designs to be shared, replicated, and critiqued.

Depends R (>= 3.5.0), randomizr (>= 0.20.0), fabricatr (>= 0.10.0), estimatr (>= 0.20.0)

Imports rlang, generics, methods

License MIT + file LICENSE


Encoding UTF-8

RoxygenNote 7.2.3

Suggests testthat, knitr, markdown, AER, dffobj, dplyr, data.table, tibble, ggplot2, future.apply, broom, MASS, Matching, betareg, biglm, gam, sf, reshape2, DesignLibrary, coin, margins, psych

NeedsCompilation no

Author Graeme Blair [aut, cre] (<https://orcid.org/0000-0001-9164-2102>), Jasper Cooper [aut] (<https://orcid.org/0000-0002-8639-3188>), Alexander Coppock [aut] (<https://orcid.org/0000-0002-5733-2386>), Macartan Humphreys [aut] (<https://orcid.org/0000-0001-7029-2326>), Neal Fultz [aut]

Maintainer Graeme Blair <graeme.blair@gmail.com>

Repository CRAN

Date/Publication 2023-01-10 07:13:10 UTC
### R topics documented:

- `cite_design` ..... 2
- `compare_diagnoses` ..... 3
- `compare_functions` ..... 4
- `DeclareDesign` ..... 6
- `declare_assignment` ..... 7
- `declare_design` ..... 9
- `declare_estimator` ..... 10
- `declare_inquiry` ..... 15
- `declare_measurement` ..... 17
- `declare_model` ..... 19
- `declare_sampling` ..... 22
- `declare_step` ..... 24
- `declare_test` ..... 25
- `diagnosand_handler` ..... 27
- `diagnose_design` ..... 29
- `diagnosis_helpers` ..... 33
- `draw_functions` ..... 34
- `expand_design` ..... 35
- `get_functions` ..... 36
- `modify_design` ..... 37
- `pop.var` ..... 39
- `post_design` ..... 39
- `redesign` ..... 41
- `reshape_diagnosis` ..... 43
- `run_design` ..... 44
- `set_citation` ..... 44
- `set_diagnosands` ..... 45
- `simulate_design` ..... 46
- `tidy.diagnosis` ..... 48
- `tidy_try` ..... 49

### Description

Obtain the preferred citation for a design

### Usage

```
cite_design(design, ...)
```
The function `compare_diagnoses` runs a many-to-many merge matching by inquiry and term (if present). If `merge_by_estimator` equals TRUE, estimator is also included in the merging condition. Any diagnosand that is not included in both designs will be dropped from the merge.

**Value**

A list with a data.frame of compared diagnoses and both diagnoses.
Examples

```r
design_a <- declare_model(N = 100, 
  U = rnorm(N), 
  Y_Z_0 = U, 
  Y_Z_1 = U + rnorm(N, mean = 2, sd = 2)) + 
  declare_assignment(Z = complete_ra(N, prob = 0.5)) + 
  declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) + 
  declare_measurement(Y = reveal_outcomes(Y ~ Z)) + 
  declare_estimator(Y ~ Z, inquiry = "ATE")

design_b <- replace_step(design_a, step = "assignment", 
  declare_assignment(Z = complete_ra(N, prob = 0.3)) )

comparison <- compare_diagnoses(design_a, design_b, sims = 40)
```

---

**compare_functions**  
**Compare two designs**

**Description**

Compare two designs

**Usage**

```r
compare_designs(
  design1, 
  design2, 
  format = "ansi8", 
  pager = "off", 
  context = -1L, 
  rmd = FALSE 
)
```

```r
compare_design_code(
  design1, 
  design2, 
  format = "ansi256", 
  mode = "sidebyside", 
  pager = "off", 
  context = -1L, 
  rmd = FALSE 
)
```

```r
compare_design_summaries(
  design1, 
  design2, 
)```
compare_functions

```r
classifiers <- compare_design_data(
  design1, design2,
  format = "ansi256",
  mode = "sidebyside",
  pager = "off",
  context = -1L,
  rmd = FALSE
)
```

```r
classifiers <- compare_design_estimates(
  design1, design2,
  format = "ansi256",
  mode = "auto",
  pager = "off",
  context = -1L,
  rmd = FALSE
)
```

```r
classifiers <- compare_design_inquiries(
  design1, design2,
  format = "ansi256",
  mode = "sidebyside",
  pager = "off",
  context = -1L,
  rmd = FALSE
)
```

**Arguments**

- **design1**
  A design object, typically created using the `+` operator

- **design2**
  A design object, typically created using the `+` operator

- **format**
  Format (in console or HTML) options from `diffobj::diffChr`

- **pager**
  Pager option from `diffobj::diffChr`

- **context**
  Context option from `diffobj::diffChr` which sets the number of lines around differences that are printed. By default, all lines of the two objects are shown. To show only the lines that are different, set `context = 0`; to get one line around differences for context, set to 1.
rmd
Set to TRUE use in Rmarkdown HTML output. NB: will not work with LaTeX, Word, or other .Rmd outputs.

mode
Mode options from diffobj::diffChr

Examples

design1 <- declare_model(N = 100, u = rnorm(N), potential_outcomes(Y ~ Z + u)) +
declare_inquiry(ATE = mean(Y[Z_1] - Y[Z_0])) +
declare_sampling(S = complete_rs(N, n = 75)) +
declare_assignment(Z = complete_ra(N, m = 50)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")

design2 <- declare_model(N = 200, U = rnorm(N),
potential_outcomes(Y ~ 0.5*Z + U)) +
declare_inquiry(ATE = mean(Y[Z_1] - Y[Z_0])) +
declare_sampling(S = complete_rs(N, n = 100)) +
declare_assignment(Z = complete_ra(N, m = 25)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, .method = lm_robust, inquiry = "ATE")

compare_designs(design1, design2)
compare_design_code(design1, design2)
compare_design_summaries(design1, design2)
compare_design_data(design1, design2)
compare_design_estimates(design1, design2)
compare_design_inquiries(design1, design2)

==

DeclareDesign  DeclareDesign package

Description

The four main types of functions are to declare a step, to combine steps into designs, and to manipulate designs and designers (functions that return designs).

Design Steps

declare_model  Model step
declare_inquiry Inquiry step
declare_sampling Data strategy step (sampling)
declare_assignment Data strategy step (assignment)
declare_measurement Data strategy step (measurement)
declare_estimator  Answer strategy step (Estimator)
declare_test Answer strategy step (Testing function)
**Design Objects**

+ Add steps to create a design
  
  **redesign** Change design parameters
  
  **draw_data** Draw a simulated dataset
  
  **run_design** Draw one set of inquiry values and estimates
  
  **diagnose_design** Diagnose a design
  
  **cite_design** Cite a design

**Design Editing**

  **modify_design** Add, delete or replace a step
  
  **redesign** Modify local variables within a design (advanced)

**Designers**

  **expand_design** Generate designs from a designer
  
  **designs** See also the DesignLibrary package for designers to use

---

**declare_assignment** **Declare Data Strategy: Assignment**

**Description**

Declare Data Strategy: Assignment

**Usage**

```r
declare_assignment(..., handler = assignment_handler, label = NULL)
```

```r
assignment_handler(data, ..., legacy = FALSE)
```

**Arguments**

- `...` arguments to be captured, and later passed to the handler
- `handler` a tidy-in, tidy-out function
- `label` a string describing the step
- `data` A data.frame.
- `legacy` Use the legacy randomizr functionality. This will be disabled in future; please use legacy = FALSE.

**Value**

A function that takes a data.frame as an argument and returns a data.frame with assignment columns appended.
Examples

# declare_assignment in use
## Two-arm randomized experiment
design <-
    declare_model(
        N = 500,
        X = rep(c(0, 1), each = N / 2),
        U = rnorm(N, sd = 0.25),
        potential_outcomes(Y ~ 0.2 * Z + X + U)
    ) +
    declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
    declare_sampling(S = complete_rs(N = N, n = 200)) +
    declare_assignment(Z = complete_ra(N = N, m = 100)) +
    declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
    declare_estimator(Y ~ Z, inquiry = "ATE")

# Set up population to assign
model <- declare_model(
    villages = add_level(
        N = 30,
        N_households = sample(c(50:100), N, replace = TRUE)
    ),
    households = add_level(
        N = N_households,
        N_members = sample(c(1, 2, 3, 4), N, replace = TRUE)
    ),
    individuals = add_level(
        N = N_members,
        age = sample(18:90, N, replace = TRUE),
        gender = rbinom(n = N, size = 1, prob = .5)
    )
)

# Assignment procedures
## Complete random assignment
design <-
    model +
    declare_assignment(Z = complete_ra(N = N, m = 1000))

## Cluster random assignment
design <-
    model +
    declare_assignment(Z = cluster_ra(clusters = villages, n = 15))

## Block and cluster random assignment
design <-
    model +
    declare_assignment(Z = block_and_cluster_ra(
        blocks = villages,
        clusters = households,
    )
)
## Block random assignment

```r
define_design(block_m = rep(20, 30))

## Block random assignment
```r
design <- model +
  declare_assignment(Z = block_ra(blocks = gender, m = 100))

## Block random assignment using probabilities
```r
design <-
  model +
  declare_assignment(Z = block_ra(blocks = gender,
                                 block_prob = c(1 / 3, 2 / 3)))

## Factorial assignment
```r
design <-
  model +
  declare_assignment(Z1 = complete_ra(N = N, m = 100),
                    Z2 = block_ra(blocks = Z1))

## Assignment using functions outside of randomizr
```r
design <-
  model +
  declare_assignment(Z = rbinom(n = N, size = 1, prob = 0.35))
```

---

**declare_design**

*Declare a design*

**Description**

Declare a design

**Usage**

```r
## S3 method for class 'dd'
lhs + rhs
```

**Arguments**

- **lhs**
  - A step in a research design, beginning with a function that defines the model. Steps are evaluated sequentially. With the exception of the first step, all steps must be functions that take a `data.frame` as an argument and return a `data.frame`. Steps are declared using the `declare FUNCTIONS` functions, i.e., `declare_model`, `declare_inquiry`, `declare_sampling`, `declare_assignment`, `declare_measurement`, `declare_estimator`, and `declare_test`.  

- **rhs**
  - A second step in a research design
**Value**

a design

**Examples**

design <-
  declare_model(
    N = 500,
    U = rnorm(N),
    potential_outcomes(Y ~ Z + U)
  ) +
  declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
  declare_sampling(S = complete_rs(N, n = 250)) +
  declare_assignment(Z = complete_ra(N, m = 25)) +
  declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
  declare_estimator(Y ~ Z, inquiry = "ATE")

dat <- draw_data(design)
head(dat)
run_design(design)

# You may wish to have a design with only one step:

design <- declare_model(N = 500, noise = rnorm(N)) + NULL

## Not run:
diagnosis <- diagnose_design(design)
summary(diagnosis)
## End(Not run)

design

---

**declare_estimater**

*Declare estimator*

**Description**

Declares an estimator which generates estimates and associated statistics.

Use of `declare_test` is identical to use of `declare_estimater`. Use `declare_test` for hypothesis testing with no specific inquiry in mind; use `declare_estimater` for hypothesis testing when you can link each estimate to an inquiry. For example, `declare_test` could be used for a K-S test of distributional equality and `declare_estimater` for a difference-in-means estimate of an average treatment effect.
Usage

```
decallar_estimator(
    ..., 
    handler = label_estimator(method_handler), 
    label = "estimator"
)
```

```
decallar_estimators(
    ..., 
    handler = label_estimator(method_handler), 
    label = "estimator"
)
```

```
label_estimator(fn)
```

```
method_handler(
    data,
    ..., 
    .method = estimatr::lm_robust,
    .summary = tidy_try,
    model,
    model_summary,
    term = FALSE
)
```

Arguments

... arguments to be captured, and later passed to the handler

handler a tidy-in, tidy-out function

label a string describing the step

fn A function that takes a data.frame as an argument and returns a data.frame with the estimates, summary statistics (i.e., standard error, p-value, and confidence interval), and a term column for labeling coefficient estimates.

data a data.frame

.method A method function, e.g. lm or glm. By default, the method is the `lm_robust` function from the `estimatr` package, which fits OLS regression and calculates robust and cluster-robust standard errors.

.summary A method-in data-out function to extract coefficient estimates or method summary statistics, such as `tidy` or `glance`. By default, the DeclareDesign method summary function `tidy_try` is used, which first attempts to use the available tidy method for the method object sent to `method`, then if not attempts to summarize coefficients using the `coef(summary())` and `confint` methods. If these do not exist for the method object, it fails.

model Deprecated argument. Use .method instead.

model_summary Deprecated argument. Use .summary instead.
declare_estimator
term

Symbols or literal character vector of term that represent quantities of interest, i.e. Z. If FALSE, return the first non-intercept term; if TRUE return all term. To escape non-standard-evaluation use !!.

Details

declare_estimator is designed to handle two main ways of generating parameter estimates from data.

In declare_estimator, you can optionally provide the name of an inquiry or an objected created by declare_inquiry to connect your estimate(s) to inquiry(s).

The first is through label_estimator(method_handler), which is the default value of the handler argument. Users can use standard method functions like lm, glm, or iv_robust. The methods are summarized using the function passed to the summary argument. This will usually be a "tidier" like broom::tidy. The default summary function is tidy_try, which applies a tidy method if available, and if not, tries to make one on the fly.

An example of this approach is:

\[
declare_estimator(Y \sim Z + X, \text{.method = lm\_robust, .summary = tidy, term = "Z", inquiry = "ATE"})
\]

The second approach is using a custom data-in, data-out function, usually first passed to label_estimator. The reason to pass the custom function to label_estimator first is to enable clean labeling and linking to inquiries.

An example of this approach is:

\[
my\_fun <- function(data){ with(data, median(Y[Z == 1]) - median(Y[Z == 0])) }
declare_estimator(handler = label_estimator(my\_fun), inquiry = "ATE")
\]

label_estimator takes a data-in-data out function to fn, and returns a data-in-data-out function that first runs the provided estimation function fn and then appends a label for the estimator and, if an inquiry is provided, a label for the inquiry.

Value

A function that accepts a data.frame as an argument and returns a data.frame containing the value of the estimator and associated statistics.

Examples

# Setup for examples
design <-
declare_model(
  N = 500,
  gender = rbinom(N, 1, 0.5),
  U = rnorm(N, sd = 0.25),
  potential_outcomes(Y ~ rbinom(
    N, 1, prob = pnorm(0.2 * Z + 0.2 * gender + 0.1 * Z * gender + U)
  )))
) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_estimator

```r
declare_sampling(S = complete_rs(N = N, n = 200)) +
declare_assignment(Z = complete_ra(N = N, m = 100)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z))

# default estimator is lm_robust with tidy summary
design_0 <-
design +
declare_estimator(Y ~ Z, inquiry = "ATE")
run_design(design_0)

# Linear regression using lm_robust and tidy summary
design_1 <-
design +
declare_estimator(formula = Y ~ Z,
  .method = lm_robust,
  .summary = tidy,
  term = "Z",
  inquiry = "ATE",
  label = "lm_no_controls"
)
run_design(design_1)

# Use glance summary function to view model fit statistics
design_2 <-
design +
declare_estimator(.method = lm_robust,
  formula = Y ~ Z,
  .summary = glance)
run_design(design_2)

# Use declare_estimator to implement custom answer strategies
my_estimator <- function(data) {
  data.frame(estimate = mean(data$Y))
}

design_3 <-
design +
declare_inquiry(Y_bar = mean(Y)) +
declare_estimator(handler = label_estimator(my_estimator),
  label = "mean",
  inquiry = "Y_bar")
run_design(design_3)

# Use `term` to select particular coefficients
design_4 <-
design +
declare_inquiry(difference_in_cates = mean(Y_Z_1[gender == 1]) - Y_Z_0[gender == 1]) -
  mean(Y_Z_1[gender == 0] - Y_Z_0[gender == 0])) +
```

"grave/var" term to select particular coefficients
declare_estimator(Y ~ Z * gender,
    term = "Z:gender",
    inquiry = "difference_in_cates",
    .method = lm_robust)

run_design(design_4)

# Use glm from base R
design_5 <-
    design +
    declare_estimator(Y ~ Z + gender,
        family = "gaussian",
        inquiry = "ATE",
        .method = glm)

run_design(design_5)

# If we use logit, we'll need to estimate the average marginal effect with
# margins::margins. We wrap this up in function we'll pass to model_summary
library(margins) # for margins
library(broom) # for tidy
tidy_margins <- function(x) {
    tidy(margins(x, data = x$data), conf.int = TRUE)
}

design_6 <-
    design +
    declare_estimator(
        Y ~ Z + gender,
        .method = glm,
        family = binomial("logit"),
        .summary = tidy_margins,
        term = "Z"
    )

run_design(design_6)

# Multiple estimators for one inquiry
design_7 <-
    design +
    declare_estimator(Y ~ Z,
        .method = lm_robust,
        inquiry = "ATE",
        label = "OLS") +
    declare_estimator(
        Y ~ Z + gender,
        .method = glm,
        family = binomial("logit"),
        .summary = tidy_margins,
        inquiry = "ATE",
        .method = glm,
        family = binomial("logit"),
        .summary = tidy_margins,
        inquiry = "ATE",
```
declare_inquiry

    term = "Z",
    label = "logit"

) run_design(design_7)
```

---

**Declare inquiry**

**Description**

Declares inquiries, or the inferential target of interest. Conceptually very close to "estimand" or "quantity of interest".

**Usage**

```
declare_inquiry(..., handler = inquiry_handler, label = "inquiry")
declare_inquiries(..., handler = inquiry_handler, label = "inquiry")
declare_estimand(...)
declare_estimands(...)
inquiry_handler(data, ..., subset = NULL, term = FALSE, label)
```

**Arguments**

- `...`: arguments to be captured, and later passed to the handler
- `handler`: a tidy-in, tidy-out function
- `label`: a string describing the step
- `data`: a data.frame
- `subset`: a subset expression
- `term`: TRUE/FALSE

**Details**

For the default diagnosands, the return value of the handler should have `inquiry` and `estimand` columns.

If `term` is TRUE, the names of `...` will be returned in a `term` column, and `inquiry` will contain the step label. This can be used as an additional dimension for use in diagnosis.
Value

a function, I(), that accepts a data.frame as an argument and returns a data.frame containing the value of the inquiry, a^m.

Examples

# Set up a design for use in examples:
## Two-arm randomized experiment
design <-
  declare_model(
    N = 500,
    X = rep(c(0, 1), each = N / 2),
    U = rnorm(N, sd = 0.25),
    potential_outcomes(Y ~ 0.2 * Z + X + U)
  ) +
  declare_assignment(Z = complete_ra(N = N, m = 250)) +
  declare_measurement(Y = reveal_outcomes(Y ~ Z))

# Some common inquiries
design +
  declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0))

design +
  declare_inquiry(difference_in_var = var(Y_Z_1) - var(Y_Z_0))

design +
  declare_inquiry(mean_Y = mean(Y))

# Inquiries among a subset
design +
  declare_inquiry(ATT = mean(Y_Z_1 - Y_Z_0),
                  subset = (Z == 1))

design +
  declare_inquiry(CATE = mean(Y_Z_1 - Y_Z_0),
                  subset = X == 1)

# equivalently
design +
  declare_inquiry(CATE = mean(Y_Z_1[X == 1] - Y_Z_0[X == 1]))

# Add inquiries to a design along with estimators that reference them
diff_in_variances <-
  function(data) {
    data.frame(estimate = with(data, var(Y[Z == 1]) - var(Y[Z == 0])))
  }

design_1 <-
  design +
  declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0),
                  diff_in_variances)
difference_in_var = \text{var}(Y_{Z_1}) - \text{var}(Y_{Z_0}) + \\
declare_measurement(Y = \text{reveal_outcomes}(Y \sim Z)) + \\
declare_estimator(Y \sim Z, \\
\text{inquiry} = \text{"ATE"}, \\
\text{label} = \text{"DIM"}) + \\
declare_estimator(handler = \\
\text{label_estimator}(\text{diff_in_variances}), \\
\text{inquiry} = \text{"difference_in_var"}, \\
\text{label} = \text{"DIV"})

run_design(design_1)

# Two inquiries using one estimator

design_2 <- \\
design + \\
declare_inquiry(ATE = mean(Y_{Z_1} - Y_{Z_0})) + \\
declare_inquiry(ATT = mean(Y_{Z_1} - Y_{Z_0}), \text{subset} = (Z == 1)) + \\
declare_estimator(Y \sim Z, \text{inquiry} = \text{c("ATE", "ATT")})

run_design(design_2)

# Two inquiries using different coefficients from one estimator

design_3 <- \\
design + \\
declare_inquiry(intercept = mean(Y_{Z_0}), \\
slope = mean(Y_{Z_1} - Y_{Z_0})) + \\
declare_estimator( \\
Y \sim Z, \\
.method = \text{lm_robust}, \\
term = \text{TRUE}, \\
inquiry = \text{c("intercept", "slope")})

run_design(design_3)

# declare_inquiries usage

design_4 <- design + \\
declare_inquiries( \\
ATE = mean(Y_{Z_1}[X == 1] - Y_{Z_0}[X == 1]), \\
CATE_{X0} = mean(Y_{Z_1}[X == 0] - Y_{Z_0}[X == 0]), \\
CATE_{X1} = mean(Y_{Z_1}[X == 1] - Y_{Z_0}[X == 1]), \\
Difference_in_CATEs = CATE_{X1} - CATE_{X0}, \\
mean_Y = mean(Y))
Description
This function adds measured data columns that can be functions of unmeasured data columns.

Usage

```r
declare_measurement(..., handler = measurement_handler, label = NULL)
```

```r
measurement_handler(data, ...)
```

Arguments

- `...` arguments to be captured, and later passed to the handler
- `handler` a tidy-in, tidy-out function
- `label` a string describing the step
- `data` A data.frame.

Details

It is also possible to include measured variables in your declare_model call or to add variables using declare_step. However, putting latent variables in declare_model and variables-as-measured in declare_measurement helps communicate which parts of your research design are in M and which parts are in D.

Value

A function that returns a data.frame.

Examples

```r
# declare_measurement in use
## Two-arm randomized experiment
design <-
declare_model(
  N = 500,
  X = rep(c(0, 1), each = N / 2),
  U = rnorm(N, sd = 0.25),
  potential_outcomes(Y ~ 0.2 * Z + X + U)
) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_sampling(S = complete_rs(N = N, n = 200)) +
declare_assignment(Z = complete_ra(N = N, m = 100)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")

# Reveal potential outcomes according to treatment assignment
design <-
declare_model(N = 100,
  potential_outcomes(Y ~ rbinom(
    N, size = 1, prob = 0.1 * Z + 0.5
  )
) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_sampling(S = complete_rs(N = N, n = 200)) +
declare_assignment(Z = complete_ra(N = N, m = 100)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")
```
## declare_model

Declare the size and features of the population

### Description

Declare the size and features of the population

### Usage

```r
declare_model(..., handler = fabricate, label = NULL)
```

### Arguments

- `...`: arguments to be captured, and later passed to the handler
- `handler`: a tidy-in, tidy-out function
- `label`: a string describing the step
Value

A function that returns a data.frame.

Examples

```r
# declare_model is usually used when concatenating
# design elements with `+`

## Example: Two-arm randomized experiment
design <-
  declare_model(
    N = 500,
    X = rep(c(0, 1), each = N / 2),
    U = rnorm(N, sd = 0.25),
    potential_outcomes(Y ~ 0.2 * Z + X + U)
  ) +
  declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
  declare_assignment(Z = complete_ra(N = N, m = 250)) +
  declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
  declare_estimator(Y ~ Z, inquiry = "ATE")

# declare_model returns a function:
M <- declare_model(N = 100)
M()

# Declare a population from existing data
M <- declare_model(data = mtcars)

# Resample from existing data
M <- declare_model(N = 100, data = mtcars, handler = resample_data)

# Declare a model with covariates:
# observed covariates X1 and X2 and
# unobserved heterogeneity U that each affect
# outcome Y
M <- declare_model(
  N = 100,
  U = rnorm(N),
  X1 = rbinom(N, size = 1, prob = 0.5),
  X2 = X1 + rnorm(N),
  Y = 0.1 * X1 + 0.2 * X2 + 0.1 * X1 * X2 + U
)

# We can draw correlated variables using draw_multivariate
M <-
  declare_model(
    draw_multivariate(c(X1, X2) ~ MASS::mvrnorm(
      N = 1000,
      mu = c(0, 0),
      Sigma = matrix(c(1, 0.3, 0.3, 1), nrow = 2)
    )))
```
# Declare potential outcomes model dependent on assignment Z

## Manually

```r
M <-
  declare_model(N = 100,
    Y_Z_0 = rbinom(N, size = 1, prob = 0.5),
    Y_Z_1 = rbinom(N, size = 1, prob = 0.6)
  )
```

## Using potential_outcomes

```r
M <-
  declare_model(N = 100,
    potential_outcomes(Y ~ rbinom(N, size = 1, prob = 0.1 * Z + 0.5))
  )
```

## we can draw from a distribution of effect sizes

```r
M <-
  declare_model(
    N = 100,
    tau = runif(1, min = 0, max = 1),
    U = rnorm(N),
    potential_outcomes(Y ~ tau * Z + U)
  )
```

## we can simulate treatment-by-covariate effect heterogeneity:

```r
M <-
  declare_model(
    N = 100,
    U = rnorm(N),
    X = rbinom(N, 1, prob = 0.5),
    potential_outcomes(Y ~ 0.3 * Z + 0.2 * X + 0.1 * Z * X + U)
  )
```

## potential outcomes can respond to two treatments:

```r
M <-
  declare_model(
    N = 6,
    U = rnorm(N),
    potential_outcomes(Y ~ Z1 + Z2 + U,
      conditions = list(Z1 = c(0, 1), Z2 = c(0, 1)))
  )
```

# Declare a two-level hierarchical population

# containing varying numbers of individuals within
# households and an age variable defined at the individual
# level

```r
M <-
  declare_model(
    households = add_level(
      N = 100,
      N_members = sample(c(1, 2, 3, 4), N,
        prob = c(0.2, 0.3, 0.25, 0.25),
        replace = TRUE)
    ),
    individuals = add_level(
      N = 100,
      age = runif(N, min = 0, max = 100)
    )
  )
```
\[ N = N_{\text{members}}, \]
\[ \text{age} = \text{sample}(18:90, N, \text{replace} = \text{TRUE}) \]

\[ \]

## Panel data have a more complex structure:
\[
M \leftarrow \text{declare_model}(
    \text{countries} = \text{add_level}(
        N = 196,
        \text{country\_shock} = \text{rnorm}(N)
    ),
    \text{years} = \text{add_level}(
        N = 100,
        \text{time\_trend} = 1:N,
        \text{year\_shock} = \text{runif}(N, 1, 10),
        \text{nest} = \text{FALSE}
    ),
    \text{observation} = \text{cross\_levels}(
        \text{by} = \text{join\_using(countries, years)},
        \text{observation\_shock} = \text{rnorm}(N),
        \text{Y} = 0.01 \times \text{time\_trend} + \text{country\_shock} + \text{year\_shock} + \text{observation\_shock}
    )
)
\]

# Declare a population using a custom function
# the default handler is fabricatr::fabricate,
# but you can supply any function that returns a data.frame
my_model_function <- function(N) {
    data.frame(u = rnorm(N))
}

M <- declare_model(N = 10, handler = my_model_function)

---

### `declare_sampling`  
#### Declare sampling procedure

**Description**

Declare sampling procedure

**Usage**

```
declare_sampling(..., handler = sampling_handler, label = NULL)
```

```
sampling_handler(data, ..., legacy = FALSE)
```
`declare_sampling`

Arguments

- `...`: arguments to be captured, and later passed to the handler
- `handler`: a tidy-in, tidy-out function
- `label`: a string describing the step
- `data`: A data.frame.
- `legacy`: Use the legacy randomizr functionality. This will be disabled in future; please use `legacy = FALSE`.

Value

A sampling declaration, which is a function that takes a data.frame as an argument and returns a data.frame subsetted to sampled observations and (optionally) augmented with inclusion probabilities and other quantities.

Examples

```r
# declare_sampling in use
## Two-arm randomized experiment
design <-
declare_model(
  N = 500,
  X = rep(c(0, 1), each = N / 2),
  U = rnorm(N, sd = 0.25),
  potential_outcomes(Y ~ 0.2 * Z + X + U)
) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_sampling(S = complete_rs(N = N, n = 200)) +
declare_assignment(Z = complete_ra(N = N, m = 100)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")

# Set up population to sample from
model <- declare_model(
  villages = add_level(
    N = 30,
    N_households = sample(c(50:100), N, replace = TRUE)
  ),
  households = add_level(
    N = N_households,
    N_members = sample(c(1, 2, 3, 4), N,
      prob = c(0.2, 0.3, 0.25, 0.25), replace = TRUE)
  ),
  individuals = add_level(
    N = N_members,
    age = sample(18:90, N, replace = TRUE),
    gender = rbinom(n = N, size = 1, prob = .5)
  )
)
```
declare_step

# Sampling procedures
## Complete random sampling
design <- model +
  declare_sampling(S = complete_rs(N = N, n = 1000))

## Cluster random sampling
design <- model +
  declare_sampling(S = cluster_rs(clusters = villages, n = 15))

## Strata and cluster random sampling
design <- model +
  declare_sampling(S = strata_and_cluster_rs(
    strata = villages,
    clusters = households,
    strata_n = rep(20, 30)))

## Stratified random sampling
design <- model +
  declare_sampling(S = strata_rs(strata = gender, n = 100))

- **declare_step**
  - Declare a custom step

**Description**

With `declare_step`, you can include any function that takes data as one of its arguments and returns data in a design declaration. The first argument is always a "handler", which is the name of the data-in, data-out function. For handy data manipulations use `declare_step(fabricate, ...)`.

**Usage**

```r
declare_step(
  ...
  handler = function(data, ...f, ...) ...f(data, ...),
  label = NULL
)
```

**Arguments**

- `...` arguments to be captured, and later passed to the handler
- `handler` a tidy-in, tidy-out function
- `label` a string describing the step
Value

A function that returns a data.frame.

Examples

    population <- declare_model(N = 5, noise = rnorm(N))
    manipulate <- declare_step(fabricate, noise_squared = noise^2, zero = 0)

    design <- population + manipulate
    draw_data(design)

Description

Declares an test which generates a test statistic and associated inferential statistics.

Use of declare_test is identical to use of declare_estimator. Use declare_test for hypothesis testing with no specific inquiry in mind; use declare_estimator for hypothesis testing when you can link each estimate to an inquiry. For example, declare_test could be used for a K-S test of distributional equality and declare_estimator for a difference-in-means estimate of an average treatment effect.

See declare_estimator help for an explanation of how to use method_handler, which is used identically in both declare_estimator and declare_test. The main difference between declare_estimator and declare_test is that declare_test does not link with an explicit inquiry.

Usage

    declare_test(..., handler = label_test(method_handler), label = "test")

    label_test(fn)

Arguments

... arguments to be captured, and later passed to the handler
handler a tidy-in, tidy-out function
label a string describing the step
fn A function that takes a data.frame as an argument and returns a data.frame with test statistics as columns.

Details

label_test takes a data-in-data out function to fn, and returns a data-in-data-out function that first runs the provided test function fn and then appends a label for the test.
Value

A function that accepts a data.frame as an argument and returns a data.frame containing the value of the test statistic and other inferential statistics.

See Also

See `declare_estimator` for documentation of the `method_handler` function.

Examples

# Balance test F test

balance_test_design <-
declare_model(
  N = 100,        
  cov1 = rnorm(N),
  cov2 = rnorm(N),
  cov3 = rnorm(N)
) + 
declare_assignment(Z = complete_ra(N, prob = 0.2)) + 
declare_test(Z ~ cov1 + cov2 + cov3, .method = lm_robust, .summary = glance)

## Not run:
diagnosis <- diagnose_design(
  design = balance_test_design, 
  diagnosands = declare_diagnosands(  
    false_positive_rate = mean(p.value <= 0.05)
  )
)

## End(Not run)

# K-S test of distributional equality

ks_test <- function(data) {
  test <- with(data, ks.test(x = Y[Z == 1], y = Y[Z == 0]))
  data.frame(statistic = test$statistic, p.value = test$p.value)
}

distributional_equality_design <-
declare_model(
  N = 100,        
  Y_Z_1 = rnorm(N),
  Y_Z_0 = rnorm(N, sd = 1.5)
) + 
declare_assignment(Z = complete_ra(N, prob = 0.5)) + 
declare_measurement(Y = reveal_outcomes(Y ~ Z)) + 
declare_test(handler = label_test(ks_test), label = "ks-test")

## Not run:
diagnosis <- diagnose_design(
  design = distributional_equality_design,
diagnosand_handler

    diagnosands = declare_diagnosands(power = mean(p.value <= 0.05))

## End(Not run)

# Thanks to Jake Bowers for this example

library(coin)

our_ttest <- function(data) {
    res <- coin::oneway_test(
        outcome ~ factor(Xclus),
        data = data,
        distribution = "asymptotic"
    )
    data.frame(p.value = pvalue(res)[[1]])
}

ttest_design <-
    declare_model(
        N = 100,
        Xclus = rbinom(n = N, size = 1, prob = 0.2),
        outcome = 3 + rnorm(N)) +
    declare_test(handler = label_test(our_ttest), label = "t-test")

## Not run:
diagnosis <- diagnose_design(
    design = ttest_design,
    diagnosands = declare_diagnosands(
        false_positive_rate = mean(p.value <= 0.05))
)

## End(Not run)

---

**diagnosand_handler**  
*Declare diagnosands*

**Description**

Declare diagnosands

**Usage**

```r
diagnosand_handler(data, ..., subset = NULL, alpha = 0.05, label)

declare_diagnosands(..., handler = diagnosand_handler, label = NULL)
```
Arguments

data
A data.frame.

... A set of new diagnosands.

subset A subset of the simulations data frame within which to calculate diagnosands
e.g. subset = p.value < .05.

alpha Alpha significance level. Defaults to .05.

label Label for the set of diagnosands.

handler a tidy-in, tidy-out function

Details

If term is TRUE, the names of ... will be returned in a term column, and inquiry will contain the
step label. This can be used as an additional dimension for use in diagnosis.

Diagnosands summarize the simulations generated by diagnose_design or simulate_design. Typically, the columns of the resulting simulations data.frame include the following variables: estimate, std.error, p.value, conf.low, conf.high, and inquiry. Many diagnosands will be a function of these variables.

Value

a function that returns a data.frame

Examples

design <-
declare_model(
  N = 500,
  U = rnorm(N),
  Y_Z_0 = U,
  Y_Z_1 = U + rnorm(N, mean = 2, sd = 2)
) +
declare_assignment(Z = complete_ra(N)) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_estimator(Y ~ Z, inquiry = my_inquiry) +
declare_measurement(Y = reveal_outcomes(Y ~ Z))

## Not run:
# using built-in defaults:
diagnosis <- diagnose_design(design)
diagnosis

## End(Not run)

# You can choose your own diagnosands instead of the defaults e.g.,

my_diagnosands <-
declare_diagnosands(median_bias = median(estimate - inquiry))
## Not run:
diagnose_design <- diagnose_design(design, diagnosands = my_diagnosands)

diagnosis

## End(Not run)
## Not run:
design <- set_diagnosands(design, diagnosands = my_diagnosands)
diagnosis <- diagnose_design(design)

diagnosis

## End(Not run)

# If you do not specify diagnosands in diagnose_design,
# the function default_diagnosands() is used,
# which is reproduced below.

alpha <- 0.05
default_diagnosands <-
declare_diagnosands(
  mean_estimand = mean(estimand),
  mean_estimate = mean(estimate),
  bias = mean(estimate - estimand),
  sd_estimate = sqrt(pop.var(estimate)),
  rmse = sqrt(mean((estimate - estimand) ^ 2)),
  power = mean(p.value <= alpha),
  coverage = mean(estimand <= conf.high & estimand >= conf.low)
)

# A longer list of potentially useful diagnosands might include:

extended_diagnosands <-
declare_diagnosands(
  mean_estimand = mean(estimand),
  mean_estimate = mean(estimate),
  bias = mean(estimate - estimand),
  sd_estimate = sd(estimate),
  rmse = sqrt(mean((estimate - estimand) ^ 2)),
  power = mean(p.value <= alpha),
  coverage = mean(estimand <= conf.high & estimand >= conf.low),
  mean_se = mean(std.error),
  type_s_rate = mean((sign(estimate) != sign(estimand))[p.value <= alpha]),
  exaggeration_ratio = mean((estimate/estimand)[p.value <= alpha]),
  var_estimate = pop.var(estimate),
  mean_var_hat = mean(std.error^2),
  prop_pos_sig = estimate > 0 & p.value <= alpha,
  mean_ci_length = mean(conf.high - conf.low)
)
**Description**

Generates diagnosands from a design or simulations of a design.

**Usage**

```r
diagnose_design(
  ..., 
  diagnosands = NULL,
  sims = 500,
  bootstrap_sims = 100,
  make_groups = NULL,
  add_grouping_variables = NULL
)

diagnose_designs(
  ..., 
  diagnosands = NULL,
  sims = 500,
  bootstrap_sims = 100,
  make_groups = NULL,
  add_grouping_variables = NULL
)

vars(...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>...</code></td>
<td>A design or set of designs typically created using the + operator, or a data.frame of simulations, typically created by <code>simulate_design</code>.</td>
</tr>
<tr>
<td><code>diagnosands</code></td>
<td>A set of diagnosands created by <code>declare_diagnosands</code>. By default, these include bias, root mean-squared error, power, frequentist coverage, the mean and standard deviation of the estimate(s), the &quot;type S&quot; error rate (Gelman and Carlin 2014), and the mean of the inquiry(s).</td>
</tr>
<tr>
<td><code>sims</code></td>
<td>The number of simulations, defaulting to 500. <code>sims</code> may also be a vector indicating the number of simulations for each step in a design, as described for <code>simulate_design</code>.</td>
</tr>
<tr>
<td><code>bootstrap_sims</code></td>
<td>Number of bootstrap replicates for the diagnosands to obtain the standard errors of the diagnosands, defaulting to 100. Set to FALSE to turn off bootstrapping.</td>
</tr>
<tr>
<td><code>make_groups</code></td>
<td>Add group variables within which diagnosand values will be calculated. New variables can be created or variables already in the simulations data frame selected. Type name-value pairs within the function <code>vars</code>, i.e. <code>vars(significant = p.value &lt;= 0.05)</code>.</td>
</tr>
<tr>
<td><code>add_grouping_variables</code></td>
<td>Deprecated. Please use <code>make_groups</code> instead. Variables used to generate groups of simulations for diagnosis. Added to default list: c(&quot;design&quot;, &quot;estimand_label&quot;, &quot;estimator&quot;, &quot;outcome&quot;, &quot;term&quot;)</td>
</tr>
</tbody>
</table>
Details

If the diagnosand function contains a group_by attribute, it will be used to split-apply-combine diagnosands rather than the intersecting column names.

If `sims` is named, or longer than one element, a fan-out strategy is created and used instead.

If the packages `future` and `future.apply` are installed, you can set `plan` to run multiple simulations in parallel.

Value

a list with a data frame of simulations, a data frame of diagnosands, a vector of diagnosand names, and if calculated, a data frame of bootstrap replicates.

Examples

design <-
declare_model(
  N = 500,
  U = rnorm(N),
  Y_Z_0 = U,
  Y_Z_1 = U + rnorm(N, mean = 2, sd = 2)
) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_assignment(Z = complete_ra(N)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")

## Not run:
# using built-in defaults:
diagnosis <- diagnose_design(design)
reshape_diagnosis(diagnosis, select = "Power")

## End(Not run)

## Not run:
# Adding a group for within group diagnosis:
diagnosis <- diagnose_design(design,
  make_groups = vars(significant = p.value <= 0.05),
)
diagnosis
diagnosis <- diagnose_design(design,
  make_groups =
  vars(effect_size =
    cut(estimand, quantile(estimand, (0:4)/4),
      include.lowest = TRUE)),
)
diagnosis

## End(Not run)

# using a user-defined diagnosand
my_diagnosand <- declare_diagnosands(absolute_error = mean(abs(estimate - estimand)))

## Not run:
diagnosis <- diagnose_design(design, diagnosands = my_diagnosand)
diagnosis

get_diagnosands(diagnosis)

get_simulations(diagnosis)

## End(Not run)

# Using an existing data frame of simulations
## Not run:
simulations <- simulate_design(designs, sims = 2)
diagnosis <- diagnose_design(simulations_df = simulations_df)

## End(Not run)

# If you do not specify diagnosands, the function default_diagnosands() is used,
# which is reproduced below.

alpha <- 0.05

default_diagnosands <-
declare_diagnosands(
  mean_estimand = mean(estimand),
  mean_estimate = mean(estimate),
  bias = mean(estimate - estimand),
  sd_estimate = sqrt(pop.var(estimate)),
  rmse = sqrt(mean((estimate - estimand) ^ 2)),
  power = mean(p.value <= alpha),
  coverage = mean(estimand <= conf.high & estimand >= conf.low)
)

# A longer list of useful diagnosands might include:

extended_diagnosands <-
declare_diagnosands(
  mean_estimand = mean(estimand),
  mean_estimate = mean(estimate),
  bias = mean(estimate - estimand),
  sd_estimate = sd(estimate),
  rmse = sqrt(mean((estimate - estimand) ^ 2)),
  power = mean(p.value <= alpha),
  coverage = mean(estimand <= conf.high & estimand >= conf.low),
  mean_se = mean(std.error),
  type_s_rate = mean((sign(estimate) != sign(estimand))[p.value <= alpha]),
  exaggeration_ratio = mean((estimate/estimand)[p.value <= alpha]),
  var_estimate = pop.var(estimate),
  mean_var_hat = mean(std.error^2),
  prop_pos_sig = estimate > 0 & p.value <= alpha,
mean_ci_length = mean(conf.high - conf.low)
)

diagnosis_helpers   Explore your design diagnosis

Description

Explore your design diagnosis

Usage

get_diagnosands(diagnosis)

get_simulations(diagnosis)

Arguments

diagnosis   A design diagnosis created by `diagnose_design`.

Examples

design <-
declare_model(
  N = 500,
  U = rnorm(N),
  Y_Z_0 = U,
  Y_Z_1 = U + rnorm(N, mean = 2, sd = 2)
) +
declare_assignment(Z = complete_ra(N)) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")

## Not run:
# using built-in defaults:
diagnosis <- diagnose_design(design)
diagnosis

## End(Not run)

# using a user-defined diagnosand
my_diagnosand <- declare_diagnosands(
  absolute_error = mean(abs(estimate - estimand)))

## Not run:
diagnosis <- diagnose_design(design, diagnosands = my_diagnosand)
diagnosis
get_diagnosands(diagnosis)
get_simulations(diagnosis)
reshape_diagnosis(diagnosis)

## End(Not run)

draw_functions

**Draw data, estimates, and inquiries from a design**

**Description**
Draw data, estimates, and inquiries from a design

**Usage**

draw_data(design, data = NULL, start = 1, end = length(design))
draw_estimand(...)  
draw_estimands(...)  
draw_estimates(...)

**Arguments**

design A design object, typically created using the + operator
data A data.frame object with sufficient information to get the data, estimates, inquiries, an assignment vector, or a sample.
start (Defaults to 1) a scalar indicating which step in the design to begin with. By default all data steps are drawn, from step 1 to the last step of the design.
end (Defaults to length(design)) a scalar indicating which step in the design to finish drawing data by.
... A design or set of designs typically created using the + operator

**Examples**

design <-
  declare_model(
    N = 100,
    U = rnorm(N),
    potential_outcomes(Y ~ Z + U)
  ) +
**expand_design**

`declare_inquiry(ATE = mean(Y_{Z=1} - Y_{Z=0})) +
declare_sampling(S = complete_rs(N, n = 75)) +
declare_assignment(Z = complete_ra(N, m = 50)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")`

dat <- draw_data(design)

dat_no_sampling <- draw_data(design, end = 3)
draw_estimands(design)
draw_estimates(design)

__Description__

`expand_design` easily generates a set of design from a designer function.

__Usage__

```r
expand_design(designer, ..., expand = TRUE, prefix = "design")
```

__Arguments__

- `designer`: a function which yields a design
- `...`: Options sent to the designer
- `expand`: boolean - if true, form the crossproduct of the ..., otherwise recycle them
- `prefix`: prefix for the names of the designs, i.e. if you create two designs they would be named prefix_1, prefix_2

__Value__

if set of designs is size one, the design, otherwise a ‘by’-list of designs. Designs are given a parameters attribute with the values of parameters assigned by `expand_design`.

__Examples__

```r
## Not run:
# in conjunction with DesignLibrary
library(DesignLibrary)
```
get_estimates <- function(design, data = NULL, start = 1, end = length(design))

Arguments

- **design**: A design object, typically created using the `+` operator.
- **data**: A data.frame object with sufficient information to get the data, estimates, inquiries, an assignment vector, or a sample.
modify_design

(Defaults to 1) a scalar indicating which step in the design to begin with. By
default all data steps are drawn, from step 1 to the last step of the design.

(Defaults to length(design)) a scalar indicating which step in the design to
finish with.

Examples

design <-
  declare_model(
    N = 100,
    U = rnorm(N),
    potential_outcomes(Y ~ Z + U)
  ) +
  declare_inquiry(ATE = mean(Y|Z_1 - Y|Z_0)) +
  declare_sampling(S = complete_rs(N, n = 75)) +
  declare_assignment(Z = complete_ra(N, m = 50)) +
  declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
  declare_estimator(Y ~ Z, inquiry = "ATE")

dat <- draw_data(design)

draw_data(design, data = dat, start = 2)

get_estimates(design, data = dat)

modify_design

Modify a design after the fact

Description

Insert, delete and replace steps in an (already declared) design object.

Usage

insert_step(design, new_step, before, after)

delete_step(design, step)

replace_step(design, step, new_step)

Arguments

design A design object, usually created using the + operator, expand_design, or the
design library.

new_step The new step: Either a function or a partial call.

before The step before which to add steps.

after The step after which to add steps.

step The quoted label of the step to be deleted or replaced.
modify_design

Details

See modify_design for details.

Value

A new design object.

Examples

my_model <-
  declare_model(
    N = 100,
    U = rnorm(N),
    Y_Z_0 = U,
    Y_Z_1 = U + rnorm(N, mean = 2, sd = 2)
  )

my_assignment <- declare_assignment(Z = complete_ra(N, m = 50))
my_assignment_2 <- declare_assignment(Z = complete_ra(N, m = 25))
design <- my_model + my_assignment
design

## Not run:
insert_step(design, declare_step(dplyr::mutate, income = noise^2),
  after = my_assignment)
insert_step(design, declare_step(dplyr::mutate, income = noise^2),
  before = my_assignment)

# If you are using a design created by a designer, for example from
# the DesignLibrary package, you will not have access to the step
# objects. Instead, you can always use the label of the step.

# get the labels for the steps
names(design)

insert_step(design,
  declare_sampling(S = complete_rs(N, n = 50),
  legacy = FALSE),
  after = "my_pop")

## End(Not run)

delete_step(design, my_assignment)
replace_step(design, my_assignment, declare_step(dplyr::mutate, words = "income"))
**pop.var**

### Description

Population variance function

### Usage

```r
pop.var(x, na.rm = FALSE)
```

### Arguments

- `x`: a numeric vector, matrix or data frame.
- `na.rm`: logical. Should missing values be removed?

### Value

numeric scalar of the population variance

### Examples

```r
x <- 1:4
d # divides by (n-1)
pop.var(x) # divides by n
```

---

**post_design**

### Description

Explore your design

Print code to recreate a design

### Usage

```r
print_code(design)
```

```r
## S3 method for class 'design'
print(x, verbose = FALSE, ...)
```

```r
## S3 method for class 'design'
summary(object, verbose = TRUE, ...)
```
Arguments

- **design**: A design object, typically created using the + operator
- **x**: a design object, typically created using the + operator
- **verbose**: an indicator for printing a long summary of the design, defaults to `TRUE`
- **...**: optional arguments to be sent to summary function
- **object**: a design object created using the + operator

Examples

```r
design <-
  declare_model(
    N = 500,
    U = rnorm(N),
    potential_outcomes(Y ~ U + Z * rnorm(N, 2, 2))
  ) +
  declare_sampling(S = complete_rs(N, n = 250)) +
  declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
  declare_assignment(Z = complete_ra(N, m = 25)) +
  declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
  declare_estimator(Y ~ Z, inquiry = "my_inquiry")

design
df <- draw_data(design)
estimates <- draw_estimates(design)
inquiries <- draw_estimands(design)
print_code(design)

my_population <- declare_model(N = 100)
my_assignment <- declare_assignment(Z = complete_ra(N, m = 50))
my_design <- my_population + my_assignment
print_code(my_design)

my_model <-
  declare_model(
    N = 500,
    noise = rnorm(N),
    Y_Z_0 = noise,
    Y_Z_1 = noise + rnorm(N, mean = 2, sd = 2)
  )

my_sampling <- declare_sampling(S = complete_rs(N, n = 250))
```
my_assignment <- declare_assignment(Z = complete_ra(N, m = 25))
my_inquiry <- declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0))
my_estimator <- declare_estimator(Y ~ Z, inquiry = my_inquiry)
my_reveal <- declare_measurement(Y = reveal_outcomes(Y ~ Z))

design <- my_model +
  my_sampling +
  my_inquiry +
  my_assignment +
  my_reveal +
  my_estimator

summary(design)

---

**redesign**

**Redesigned**

**Description**

`redesign` quickly generates a design from an existing one by resetting symbols used in design handler parameters in a step’s environment (Advanced).

**Usage**

```r
redesign(design, ..., expand = TRUE)
```

**Arguments**

- `design`: An object of class design.
- `...`: Arguments to redesign e.g., `n = 100`. If redesigning multiple arguments, they must be specified as a named list.
- `expand`: If TRUE, redesign using the crossproduct of . . . , otherwise recycle them.

**Details**

Warning: `redesign` will edit any symbol in your design, but if the symbol you attempt to change does not exist in a step’s environment no changes will be made and no error or warning will be issued.

Please note that `redesign` functionality is experimental and may be changed in future versions.

**Value**

A design, or, in the case of multiple values being passed onto . . . , a ‘by’-list of designs.
Examples

```r
n <- 500
population <- declare_model(N = 1000)
sampling <- declare_sampling(S = complete_rs(N, n = n),
                            legacy = FALSE)
design <- population + sampling

# returns a single, modified design
modified_design <- redesign(design, n = 200)

# returns a list of six modified designs
design_vary_N <- redesign(design, n = seq(400, 900, 100))

# When redesigning with arguments that are vectors,
# use list() in redesign, with each list item
# representing a design you wish to create
prob_each <- c(.1, .5, .4)
assignment <- declare_assignment(
    Z = complete_ra(prob_each = prob_each),
    legacy = FALSE)
design <- population + assignment

# returns two designs

designs_vary_prob_each <- redesign(
    design,
    prob_each = list(c(.2, .5, .3), c(0, .5, .5)))

# To illustrate what does and does not get edited by redesign,
# consider the following three designs. In the first two, argument
# X is called from the step's environment; in the third it is not.
# Using redesign will alter the role of X in the first two designs
# but not the third one.
X <- 3
f <- function(b, X) b*X
g <- function(b) b*X

design1 <- declare_model(N = 1, A = X) + NULL
design2 <- declare_model(N = 1, A = f(2, X)) + NULL
design3 <- declare_model(N = 1, A = g(2)) + NULL

draw_data(design1)
draw_data(design2)
draw_data(design3)
draw_data(redesign(design1, X=0))
```
reshape_diagnosis

```r
draw_data(redesign(design2, X=0))
draw_data(redesign(design3, X=0))
```

---

**reshape_diagnosis**  
*Clean up a diagnosis object for printing*

**Description**

Take a diagnosis object and returns a pretty output table. If diagnosands are bootstrapped, se’s are put in parentheses on a second line and rounded to digits.

**Usage**

```r
reshape_diagnosis(diagnosis, digits = 2, select = NULL, exclude = NULL)
```

**Arguments**

- `diagnosis`: A diagnosis object generated by `diagnose_design`.
- `digits`: Number of digits.
- `select`: List of columns to include in output. Defaults to all.
- `exclude`: Set of columns to exclude from output. Defaults to none.

**Value**

A formatted text table with bootstrapped standard errors in parentheses.

**Examples**

```r
effect_size <- 0.1
design <-
declare_model(
  N = 100,
  U = rnorm(N),
  X = rnorm(N),
  potential_outcomes(Y ~ effect_size * Z + X + U)
)
+ declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_assignment(Z = complete_ra(N)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE", label = "unadjusted") +
declare_estimator(Y ~ Z + X, inquiry = "ATE", label = "adjusted")

diagnosis <- diagnose_design(design, sims = 100)
reshape_diagnosis(diagnosis)

reshape_diagnosis(diagnosis, select = c("Bias", "Power"))
```
run_design

Run a design one time

Description
Run a design one time

Usage
run_design(design)

Arguments
design a DeclareDesign object

Examples
design <-
declare_model(
  N = 100, X = rnorm(N),
  potential_outcomes(Y ~ (.25 + X) * Z + rnorm(N))
) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_assignment(Z = complete_ra(N, m = 50)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE")

run_design(design)

set_citation

Set the citation of a design

Description
Set the citation of a design

Usage
set_citation(
design,
title = NULL,
author = NULL,
year = NULL,
description = "Unpublished research design declaration",
citation = NULL
)
Arguments

- **design**: A design typically created using the `+` operator.
- **title**: The title of the design, as a character string.
- **author**: The author(s) of the design, as a character string.
- **year**: The year of the design, as a character string.
- **description**: A description of the design in words, as a character string.
- **citation**: (optional) The preferred citation for the design, as a character string, in which case title, author, year, and description may be left unspecified.

Value

A design object with a citation attribute.

Examples

```r
design <-
declare_model(data = sleep) +
    declare_sampling(S = complete_rs(N, n = 10))

design <-
    set_citation(design,
        author = "Lovelace, Ada",
        title = "Notes",
        year = 1953,
        description = "This is a text description of a design")

cite_design(design)
```

Description

A researcher often has a set of diagnosands in mind to appropriately assess the quality of a design. `set_diagnosands` sets the default diagnosands for a design, so that later readers can assess the design on the same terms as the original author. Readers can also use `diagnose_design` to diagnose the design using any other set of diagnosands.

Usage

```
set_diagnosands(x, diagnosands = default_diagnosands)
```
simulate_design

Arguments

- `x` A design typically created using the `+` operator, or a simulations data.frame created by `simulate_design`.
- `diagnosands` A set of diagnosands created by `declare_diagnosands`.

Value

A design object with a diagnosand attribute.

Examples

```r
design <-
declare_model(data = sleep) +
declare_inquiry(mean_outcome = mean(extra)) +
declare_sampling(S = complete_rs(N, n = 10)) +
declare_estimator(extra ~ 1, inquiry = "mean_outcome",
    term = '(Intercept)', .method = lm_robust)

diagnosands <- declare_diagnosands(
    median_bias = median(estimate - inquiry))

design <- set_diagnosands(design, diagnosands)

## Not run:
diagnose_design(design)
simulations_df <- simulate_design(design)
simulations_df <- set_diagnosands(simulations_df, design)
diagnose_design(simulations_df)

## End(Not run)
```

Description

Runs many simulations of a design and returns a simulations data.frame.

Usage

- `simulate_design(..., sims = 500)`
- `simulate_designs(..., sims = 500)`
Arguments

... A design created using the + operator, or a set of designs. You can also provide a single list of designs, for example one created by \texttt{expand_design}.

\texttt{sims} The number of simulations, defaulting to 500. If \texttt{sims} is a vector of the form \texttt{c(10, 1, 2, 1)} then different steps of a design will be simulated different numbers of times.

Details

Different steps of a design may each be simulated different a number of times, as specified by \texttt{sims}. In this case simulations are grouped into "fans". The nested structure of simulations is recorded in the dataset using a set of variables named "step\_x\_draw." For example if \texttt{sims = c(2,1,1,3)} is passed to \texttt{simulate\_design}, then there will be two distinct draws of step 1, indicated in variable "step\_1\_draw" (with values 1 and 2) and there will be three draws for step 4 within each of the step 1 draws, recorded in "step\_4\_draw" (with values 1 to 6).

Examples

\begin{verbatim}
my_model <-
declare_model(
    N = 500,
    U = rnorm(N),
    Y\_Z\_0 = U,
    Y\_Z\_1 = U + rnorm(N, mean = 2, sd = 2)
)

my_assignment <- declare_assignment(Z = complete_ra(N))

my_inquiry <- declare_inquiry(ATE = mean(Y\_Z\_1 - Y\_Z\_0))

my_estimator <- declare_estimator(Y ~ Z, inquiry = my_inquiry)

my_reveal <- declare_measurement(Y = reveal_outcomes(Y ~ Z))

design <- my_model +
    my_inquiry +
    my_assignment +
    my_reveal +
    my_estimator

## Not run:
simulations <- simulate_design(designs, sims = 2)
diagnosis <- diagnose_design(simulations_df = simulations)
## End(Not run)

## Not run:
# A fixed population with simulations over assignment only
head(simulate_design(design, sims = c(1, 1, 100, 1)))
## End(Not run)
\end{verbatim}
tidy.diagnosis

Tidy diagnosis

Description
Tidy diagnosis

Usage
```r
## S3 method for class 'diagnosis'
tidy(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

- `x`: A diagnosis object generated by `diagnose_design`.
- `conf.int`: Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to `TRUE`.
- `conf.level`: The confidence level to use for the confidence interval if `conf.int = TRUE`. Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
- `...`: extra arguments (not used)

Value
A data.frame with columns for diagnosand names, estimated diagnosand values, bootstrapped standard errors and confidence intervals

Examples
```r
effect_size <- 0.1
design <-
declare_model(
  N = 100,
  U = rnorm(N),
  X = rnorm(N),
  potential_outcomes(Y ~ effect_size * Z + X + U)
) +
declare_inquiry(ATE = mean(Y_Z_1 - Y_Z_0)) +
declare_assignment(Z = complete_ra(N)) +
declare_measurement(Y = reveal_outcomes(Y ~ Z)) +
declare_estimator(Y ~ Z, inquiry = "ATE", label = "unadjusted") +
declare_estimator(Y ~ Z + X, inquiry = "ATE", label = "adjusted")

diagnosis <- diagnose_design(design, sims = 100)
tidy(diagnosis)
```
Tidy Model Results and Filter to Relevant Coefficients

Description

Tidy function that returns a tidy data.frame of model results and allows filtering to relevant coefficients. The function will attempt to tidy model objects even when they do not have a tidy method available. For best results, first load the broom package via `library(broom)`.

Usage

```
tidy_try(fit, term = FALSE)
```

Arguments

- `fit`: A model fit, as returned by a modeling function like `lm`, `glm`, or estimatr::lm_robust.
- `term`: A character vector of the terms that represent quantities of interest, i.e., "Z". If FALSE, return the first non-intercept term; if TRUE return all terms.

Value

A data.frame with coefficient estimates and associated statistics.

Examples

```
fit <- lm(mpg ~ hp + disp + cyl, data = mtcars)

 tidy_try(fit)
```
Index

+.dd(declare_design), 9
assignment_handler
  (declare_assignment), 7
cite_design, 2, 7
compare_design_code
  (compare_functions), 4
compare_design_data
  (compare_functions), 4
compare_design_estimates
  (compare_functions), 4
compare_design_inquiries
  (compare_functions), 4
compare_design_summaries
  (compare_functions), 4
compare_designs (compare_functions), 4
compare_diagnoses, 3
compare_functions, 4
declare_assignment, 6, 7, 9
declare_design, 9
declare_diagnosands, 30, 46
declare_diagnosands
  (diagnosand_handler), 27
decrease_estimand (declare_inquiry), 15
decrease_estimands (declare_inquiry), 15
decrease_estimator, 6, 9, 10, 10, 25, 26
decrease_estimators (declare_estimator), 10
decrease_inquiries (declare_inquiry), 15
decrease_inquiry, 6, 9, 12, 15
decrease_measurement, 6, 9, 17
decrease_model, 6, 9, 19
decrease_sampling, 6, 9, 22
decrease_step, 24
decrease_test, 6, 9, 25
DeclarerDesign, 6
delete_step (modify_design), 37
diagnosand_handler, 27
diagnose_design, 7, 28, 29, 33
diagnose_designs (diagnose_design), 29
diagnosis_helpers, 33
draw_data, 7
draw_data (draw_functions), 34
draw_estimand (draw_functions), 34
draw_estimands (draw_functions), 34
draw_estimates (draw_functions), 34

draw_functions, 34
estimatr, 11
expand_design, 7, 35, 37, 47
get_diagnosands (diagnosis_helpers), 33
generate_estimates (get_functions), 36
get_functions, 36
generate_simulations (diagnosis_helpers), 33
glance, 11
inquiry_handler (declare_inquiry), 15
insert_step (modify_design), 37
label_estimator (declare_estimator), 10
label_test (declare_test), 25
lm_robust, 11
measurement_handler
  (declare_measurement), 17
method_handler (declare_estimator), 10
modify_design, 7, 37, 38
plan, 31
pop.var, 39
post_design, 39
print.design (post_design), 39
print_code (post_design), 39
redesign, 7, 41
replace_step (modify_design), 37
reshape_diagnosis, 43
run_design, 7, 44
sampling_handler (declare_sampling), 22
set_citation, 44
set_diagnosands, 45
simulate_design, 3, 28, 30, 46
simulate_designs (simulate_design), 46
summary.design (post_design), 39

 tidy, 11
 tidy.diagnosis, 48
 tidy_try, 11, 49

 vars (diagnose_design), 29