Package ‘edibble’

May 6, 2024

Title Encapsulating Elements of Experimental Design

Version 1.1.1

Description
A system to facilitate designing comparative (and non-comparative) experiments using the grammar of experimental designs <https://emitanaka.org/edibble-book/>.
An experimental design is treated as an intermediate, mutable object that is
built progressively by fundamental experimental components like units, treatments, and their relation.
The system aids in experimental planning, management and workflow.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Config/testthat/edition 3

URL https://edibble.emitanaka.org/,
https://github.com/emitanaka/edibble

BugReports https://github.com/emitanaka/edibble/issues

Imports magrittr, rlang, vctrs, tibble, cli, pillar, tidyselect (>= 1.0.0), nestr, stats, AlgDesign, dae, R6, lifecycle, dplyr

Suggests testthat (>= 3.0.0), rmarkdown, openxlsx2 (>= 1.0.0), visNetwork, blocksdesign, knitr, scales, tidyr

Depends R (>= 2.10)

VignetteBuilder knitr

NeedsCompilation no

Author Emi Tanaka [aut, cre, cph] (<https://orcid.org/0000-0002-1455-259X>)

Maintainer Emi Tanaka <dr.emi.tanaka@gmail.com>

Repository CRAN

Date/Publication 2024-05-06 02:30:02 UTC

R topics documented:

`edibble-package` *edibble: Encapsulating Elements of Experimental Design*

Description

A system to facilitate designing comparative (and non-comparative) experiments using the grammar of experimental designs `https://emitanaka.org/edibble-book/`. An experimental design is treated as an intermediate, mutable object that is built progressively by fundamental experimental components like units, treatments, and their relation. The system aids in experimental planning, management and workflow.

Details

[Experimental]

(WIP)

Website

- The website for the package is at `https://edibble.emitanaka.org`
- Discussion is at `https://github.com/emitanaka/edibble/discussions`

Package options

The following options are used for changing the default view for the print out of edibble design or edibble graph.

- `edibble.tree.decorate.trts`
- `edibble.tree.decorate.units`
- `edibble.tree.decorate.rcrd`
- `edibble.tree.decorate.levels`
- `edibble.tree.decorate.main`

TODO

Author(s)

Maintainer: Emi Tanaka <`dr.emi.tanaka@gmail.com`> (ORCID) [copyright holder]

See Also

Useful links:

- `https://edibble.emitanaka.org/`
- `https://github.com/emitanaka/edibble`
- Report bugs at `https://github.com/emitanaka/edibble/issues`

`activate_provenance` *Activate the provenance in the edibble design object*

Description

This is a developer function to create a new Kitchen class with the existing design.

Usage

```
activate_provenance(
  .edibble,
  overwrite = c("graph", "anatomy", "recipe", "validation", "simulate",
    "simulate_result")
)
```

Arguments

`.edibble`	An edibble object.
`overwrite`	What object to overwrite in the provenance object.

Value

A Provenance object.

Examples

```
activate_provenance(takeout())
```

`allot_table` *Allot treatments to units and serve table*

Description

This function is a short hand that combines `allot_trts()`, `assign_trts()` and `serve_table()`.

Usage

```
allot_table(
  .edibble = NULL,
  ...,
  order = "random",
  seed = NULL,
  constrain = nesting_structure(.edibble),
  label_nested = NULL,
  fail = "error",
  .record = TRUE
)
```

Arguments

`.edibble`	An edibble design which should have units, treatments and allotment defined.
`...`	One-sided or two-sided formula. If the input is a one-sided formula then the whole treatment is applied to the specified unit.
`order`	A character vector signifying the apportion of treatments to units. The value should be either "random", "systematic-fastest", "systematic-slowest", "systematic-random-fastest", "systematic-random-slowest" or a class name corresponding to the algorithm for order_trts(). "random" allocates the treatment randomly to units based on specified allotment with restrictions implied by unit structure. "systematic-slowest" allocates the treatment in a systematic order to units such that the treatment level is slow in varying. In contrast, "systematic-fastest" is fast in varying for treatment levels. "systematic-random-fastest" and "systematic-random-slowest" allocates the treatment in a systematic order to units but where it is not possible to divide treatments equally (as the number of units are not divisible by the number of levels of the treatment factor), then the extras are chosen randomly.
`seed`	A scalar value used to set the seed so that the result is reproducible.
`constrain`	The nesting structure for units.
`label_nested`	The columns to show nested labels (if available). Tidyselect compatible.
`fail`	What to do when failing to convert graph to table.
`.record`	Whether to record the step.

`allot_trts` *Define allotment of treatments to units*

Description

This function adds the edges between factor nodes to describe the high-level relationship between factors. This function does not actually assign edges between level nodes.

Usage

```
allot_trts(.edibble = NULL, ..., .record = TRUE)
```

Arguments

`.edibble`	An edibble design which should have units, treatments and allotment defined.
`...`	One-sided or two-sided formula. If the input is a one-sided formula then the whole treatment is applied to the specified unit.
`.record`	Whether to record the step.

Value

Return an edibble design.

See Also

assign_fcts

Other user-facing functions: `allot_units`(), `design`(), `expect_rcrds`(), `export_design`(), `serve_table`(), `set_rcrds`(), `set_trts`(), `set_units`()

Examples

```
design() %>%
  set_units(block = 10,
            plot = nested_in(block, 3)) %>%
  set_trts(treat = c("A", "B", "C"),
           pest = c("a", "b")) %>%
  allot_trts(treat ~ plot,
             pest ~ block)
```

`allot_units` *Define allotment of units to nested units*

Description

This function adds the edges between factor nodes to describe the high-level relationship between factors. This function does not actually assign edges between level nodes.

Usage

```
allot_units(.edibble, ..., .record = TRUE)
```

Arguments

`.edibble`	An edibble design which should have units, treatments and allotment defined.
`...`	A two-sided formula.
`.record`	Whether to record the step.

Value

Return an edibble design.

See Also

assign_fcts

Other user-facing functions: `allot_trts()`, `design()`, `expect_rcrds()`, `export_design()`, `serve_table()`, `set_rcrds()`, `set_trts()`, `set_units()`

Examples

```
design() %>%
  set_units(block = 10,
            plot = 20) %>%
  allot_units(block ~ plot)
```

`as.data.frame.edbl_table` *Convert edible table to normal data frame*

Description

Convert edibble table to normal data frame

Usage

```
## S3 method for class 'edbl_table'
as.data.frame(x, ..., levels_as = "factor", ignore_numeric = TRUE)
```

Arguments

`x`	An edibble table
`...`	Unused. i.e. don't coerce numeric factors.
`levels_as`	Coerce the edibble factors to either "factor" or "character".
`ignore_numeric`	Whether to coerce numeric factors or not. Default is TRUE,

---

`assign_fcts` *Assign treatments or units to units*

Description

This function assigns specific treatment or unit levels to actual units.

Usage

```
assign_trts(
  .edibble = NULL,
  order = "random",
  seed = NULL,
  constrain = nesting_structure(.edibble),
  ...,
  .record = TRUE
)

assign_units(
  .edibble = NULL,
  order = "random",
  seed = NULL,
  constrain = nesting_structure(.edibble),
  ...,
  .record = TRUE
)
```

Arguments

`.edibble`	An edibble design which should have units, treatments and allotment defined.
`order`	A character vector signifying the apportion of treatments to units. The value should be either "random", "systematic-fastest", "systematic-slowest", "systematic-random-fastest", "systematic-random-slowest" or a class name corresponding to the algorithm for order_trts(). "random" allocates the treatment randomly to units based on specified allotment with restrictions implied by unit structure. "systematic-slowest" allocates the treatment in a systematic order to units such that the treatment level is slow in varying. In contrast, "systematic-fastest" is fast in varying for treatment levels. "systematic-random-fastest" and "systematic-random-slowest" allocates the treatment in a systematic order to units but where it is not possible to divide treatments equally (as the number of units are not divisible by the number of levels of the treatment factor), then the extras are chosen randomly.
`seed`	A scalar value used to set the seed so that the result is reproducible.
`constrain`	The nesting structure for units.
`...`	Arguments parsed into `order_trts` functions.
`.record`	Whether to record the step.

Value

An edibble design.

Examples

```
# 10 subject, 2 vaccine treatments
design() %>%
  set_units(subject = 10) %>%
  set_trts(vaccine = 2) %>%
  allot_trts(vaccine ~ subject) %>%
  assign_trts() %>%
  serve_table()

# 20 subjects, 2 blocks, assign subjects to blocks
design() %>%
  set_units(subject = 20,
            block = 2) %>%
  allot_units(block ~ subject) %>%
  assign_units() %>%
  serve_table()
```

`as_tibble.edbl_table` *Convert an edibble data frame to normal data frame*

Description

A patch function where there is an issue with edbl factors

Usage

```
## S3 method for class 'edbl_table'
as_tibble(x, ...)
```

Arguments

`x`	can be a list or data frame
`...`	Not currently used.

Value

A data.frame.

`autofill_rcrds` *Autofill the records*

Description

This function fills the values of the record factors by automatically choosing a simulation process. It tries to be smart by ensuring to use values that is within expectation.

Usage

```
autofill_rcrds(.data, ..., .seed = NULL, .nsim = 1L)
```

Arguments

`.data`	An edibble data.
`...`	If supplied, it is a name-value pair where the name should correspond to the record factor name and value is the f
`.seed`	The seed number.
`.nsim`	The number of simulations to run.

`column` *Select a column.*

Description

This is a helper function to select a column when data is supplied for `lvls`.

Usage

```
column(x)
```

Arguments

`x`	The column to select. Can be unquoted name or the column index.

`crossed_by` *Specify the units to cross to index a new unit*

Description

`crossed_by(A, B)` is the same as `~A:B` but `crossed_by` offers more control over the names of the new units as well as adding new attributes.

Usage

```
crossed_by(..., attrs = NULL)
```

Arguments

`...`	a sequence of units
`attrs`	Currently not implemented.

Value

An object of class "cross_lvls".

Examples

```
design("Strip-Plot Design | Strip-Unit Design") %>%
  set_units(block = 3,
            row = nested_in(block, 7),
            col = nested_in(block, 6),
            unit = nested_in(block, crossed_by(row, col)))
```

`design` *Start the edibble design*

Description

This function doesn't really do much besides create a new edibble design object.

Usage

```
design(
  .title = NULL,
  ...,
  .name = "edibble",
  .record = TRUE,
  .seed = NULL,
  .provenance = Provenance$new()
)
```

```
redesign(
  .data,
  .title,
  ...,
  .name = NULL,
  .record = TRUE,
  .seed = NULL,
  .provenance = Provenance$new()
)
```

Arguments

`.title`	Optional title of the experiment.
`...`	A series of name-value pairs where the name corresponds to the name of the metadata nad the value corresponds to the actual metadata value. If the name is omitted, then no name to the metadata is assigned for the corresponding value.
`.name`	Optional name of the experiment.
`.record`	A logical value. This indicates whether to record this code step. The default is TRUE. It should remain TRUE unless this function is used as a wrapper in other code.
`.seed`	A seed number for reproducibility.
`.provenance`	An environment setup in a manner to store methods and information to trace the origin of the design
`.data`	An edibble table.

Value

An empty `edbl_design` object.

See Also

Add variables to this design with `set_units()`, `set_trts()`, and `set_rcrds()`.

Other user-facing functions: `allot_trts()`, `allot_units()`, `expect_rcrds()`, `export_design()`, `serve_table()`, `set_rcrds()`, `set_trts()`, `set_units()`

Examples

```
design("My design")
```

`design-helpers` *Test and get edibble objects*

Description

The `is` functions tests if an object (or an object in its attribute) inherits particular class and returns `TRUE` if it does, otherwise `FALSE`.

- `is_edibble_design` checks if it inherits `edbl_design`.
- `is_edibble_graph` checks if it inherits `edbl_graph`.
- `is_edibble_table` checks if it inherits `edbl_table`
- `is_edibble` checks if the object inherits `edbl`. The search is quite simple, it checks if the object is `edbl_design`, failing that it looks to see if the attribute "design" of the object is `edbl_design`.
- `is_named_design` check if it inherits `NamedDesign`.

The `get` functions extracts the requested edibble component (table, graph, or design) from the object if possible.

- `edbl_design` tries to get `edbl_design`.
- `edbl_table` tries to get `edbl_table` with no design attribute.
- `edbl_graph` tries to get `edbl_graph`.

Usage

```
is_edibble_design(x)

is_named_design(x)

is_edibble_table(x)

is_edibble_graph(x)

is_edibble(x)

is_edibble_levels(x)

is_nest_levels(x)

is_cross_levels(x)

edbl_design(x)

edbl_table(x)
```

Arguments

`x`	An object.

Value

A logical value.

Examples

```
is_edibble_design(takeout())
```

`design_anatomy` *Anatomy of the design*

Description

This is a convenient wrapper for `dae::designAnatomy` where the `formulae` structure is automatically determined by the unit and treatment structure specified in `edibble` system. Note: the computation may be long if the design is quite complicated or there are many units.

Usage

```
design_anatomy(.edibble, ...)
```

Arguments

`.edibble`	A complete edibble design object or edibble table.
`...`	Any other arguments parsed to `dae::designAnatomy`.

Value

An object of class "des_anatomy".

Examples

```
split <- takeout(menu_split(t1 = 3, t2 = 2, r = 2))
design_anatomy(split)
```

`design_data` *Get the node or edge data from an edibble design*

Description

Get the node or edge data from an edibble design

Usage

```
fct_nodes(x)

fct_edges(x)

lvl_nodes(x)

lvl_edges(x)
```

Arguments

`x`	An edibble object.

`design_model` *A baseline model for given experimental design*

Description

This

Usage

```
design_model(data, type = c("anova", "lmer"))
```

Arguments

`data`	An edibble data.
`type`	The type of model expression to return.

`examine_process` *Examine the simulation process*

Description

Examine the simulation process

Usage

```
examine_process(data, process = NULL)

examine_process_values(data, process = NULL, sim = 1L)
```

Arguments

`data`	An edibble data frame.
`process`	The process name. Typically the name of the process. If unknown, leave this empty.
`sim`	The simulation number. Default is 1.

`examine_recipe` *Check the recipe code*

Description

Check the recipe code

Usage

```
examine_recipe(x, ...)
```

Arguments

`x`	An edibble design, edibble, or takeout object.
`...`	Not used.

Value

The recipe code.

Examples

```
examine_recipe(takeout())
```

`expect-vars` *Expected type of data entry*

Description

These functions should be used within `expect_vars` where variables that are to be recorded are constraint to the expected values when exported as an xlsx file by `export_design()`. The functions to set a particular value type (numeric, integer, date, time and character) are preceded by "to_be_" where the corresponding restriction set by `with_value()`.

Usage

```
to_be_numeric(range)

to_be_integer(range)

to_be_date(range)

to_be_time(range)

to_be_character(length)

to_be_factor(levels)
```

Arguments

`range, length`	A named list with two elements: "operator" and "value" as provided by helper `with_value()` that gives the possible range of values that the expected type can take.
`levels`	A character vector with the factor levels.

Value

A record type.

`expect_rcrds` *Set the expected values for recording variables*

Description

Set the expected values for recording variables

Usage

```
expect_rcrds(.edibble = NULL, ..., .record = TRUE)
```

Arguments

`.edibble`	An edibble design (`edbl_design`), an edibble data frame (`edbl_table`) or an object that contains the edibble data frame in the attribute `design`.
`...`	Name-value pairs with the name belonging to the variable that are plan to be recorded from `set_rcrds()` and the values are the expected types and values set by helper functions, see `?expect-rcrds`.
`.record`	A logical value. This indicates whether to record this code step. The default is TRUE. It should remain TRUE unless this function is used as a wrapper in other code.

Value

An edibble design.

See Also

Other user-facing functions: `allot_trts()`, `allot_units()`, `design()`, `export_design()`, `serve_table()`, `set_rcrds()`, `set_trts()`, `set_units()`

Examples

```
takeout(menu_crd(t = 4, n = 10)) %>%
  set_rcrds(y = unit) %>%
  expect_rcrds(y > 0)
```

`export_design` *Export the design to xlsx*

Description

This function is designed to export the design made using edibble to an external xlsx file.

Usage

```
export_design(
  .data,
  file,
  author = NULL,
  date = Sys.Date(),
  overwrite = FALSE,
  hide_treatments = FALSE,
  theme = NULL,
  subject = NULL,
  category = NULL,
  table_style = "TableStyleMedium9"
)
```

Arguments

`.data`	An edibble table to export.
`file`	File, including the path, to export the data to.
`author`	(Optional) name of the author in character. A vector of character is supported for where there are multiple authors.
`date`	The date to be inserted in header (defaults to today).
`overwrite`	A logical value indicating whether to overwrite existing file or not.
`hide_treatments`	A logical value indicating whether treatments should be included in the data entry sheet. Default is true.
`theme`	The Excel theme to use (optional). One of "Atlas", "Badge", "Berlin", "Celestial", "Crop", "Depth", "Droplet", "Facet", "Feathered", "Gallery", "Headlines", "Integral", "Ion", "Ion Boardroom", "Madison", "Main Event", "Mesh", "Office Theme", "Old Office Theme", "Organic", "Parallax", "Parcel", "Retrospect", "Savon", "Slice", "Vapor Trail", "View", "Wisp", "Wood Type".
`subject`	The subject of the workbook (optional).
`category`	The category of the workbook (optional).
`table_style`	The table style to apply to the exported data (default: "TableStyleMedium9").

Value

The input data object.

See Also

Other user-facing functions: `allot_trts()`, `allot_units()`, `design()`, `expect_rcrds()`, `serve_table()`, `set_rcrds()`, `set_trts()`, `set_units()`

`fct` *Setting the traits of factors*

Description

This function is used to set characteristics of the factors.

Usage

```
fct(.levels = character(), ...)

fct_attrs(.levels = character(), ...)
```

Arguments

`.levels`	Either a short hand given as either as a single integer (number of levels), a vector or levels created from `lvls()`.
`...`	A name-value pair of attributes. The value must be a scalar and attributed to the whole factor (not individual levels). The values are added as attributes to the output object.

See Also

lvls

Examples

```
fct(c("A", "B"))
```

`fct_generator` *Factor name generator*

Description

Generate a factor with custom levels and repetitions.

Usage

```
fct_generator(labels, nlevels)
```

Arguments

`labels`	A character vector specifying the custom labels for the factor levels.
`nlevels`	An integer or a vector of integers indicating the number of repetitions for each label. If a single integer is provided, it is recycled to match the length of `labels`. If a vector is provided, it should have the same length as `labels`.

Details

This function creates a factor with custom labels and specified repetitions for each label.

Value

A factor with custom levels and repetitions.

Examples

```
# Example usage of the function
fct_generator(labels = c("A", "B", "C"), nlevels = 3)
```

`fct_graph` *Factor graph*

Description

Get the factor graph.

Usage

```
fct_graph(x)
```

Arguments

`x`	An edibble object.

`formatting` *Print intermediate experimental design to terminal*

Description

This function prints an `edbl_graph` object as a tree to terminal. The variables are color coded (or decorated) with the given options. Any ANSI coloring or styling are only visible in the console or terminal outputs that support it. The print output is best used interactively since any text styling are lost in text or R Markdown output. More details can be found in `vignette("edbl-output", package = "edibble")`.

Usage

```
## S3 method for class 'edbl_design'
print(
  x,
  decorate_units = edibble_decorate("units"),
  decorate_trts = edibble_decorate("trts"),
  decorate_rcrds = edibble_decorate("rcrds"),
  decorate_levels = edibble_decorate("levels"),
  decorate_title = edibble_decorate("title"),
  title = NULL,
  ...
)
```

Arguments

`x`	An edibble graph.
`decorate_trts, decorate_units, decorate_rcrds, decorate_levels, decorate_title`	A function applied to the name of treatment, unit, response factors or design title. The function should return a string. Most often this wraps the name with ANSI colored text.
`title`	The title of the design.
`...`	Unused.

`graph_input` *A function to process input as input for graph manipulation*

Description

A function to process input as input for graph manipulation

Usage

```
graph_input(input, prov, ...)
```

Arguments

`input`	An input.
`prov`	A provenance object.
`...`	Unused.

`is_provenance` *Check if an object is an instance of the "Provenance" class.*

Description

This function determines whether the given object is an instance of the "Provenance" class.

Usage

```
is_provenance(x)
```

Arguments

`x`	An object to be checked for its class membership.

Value

`TRUE` if the object is an instance of the "Provenance" class, `FALSE` otherwise.

`is_takeout` *A function to check if the output is a takeout design*

Description

The function returns `TRUE` if the input is a takeout design.

Usage

```
is_takeout(x)
```

Arguments

`x`	An object.

Value

A logical value.

Examples

```
is_takeout(takeout())
```

`label_nested` *Label with nested or distinct labels*

Description

Label with nested or distinct labels

Usage

```
label_nested(x)

label_distinct(x)

index_levels(x)
```

Arguments

`x`	A unit vector.

`label_seq` *Generate a sequence of labels with custom formatting options*

Description

These can be handy for generating pseudo labels for the levels or factor names using `fct_generator`

Usage

```
label_seq_from_to(
  from = 1L,
  to = 1L,
  by = 1L,
  prefix = "",
  suffix = "",
  sep_prefix = "",
  sep_suffix = "",
  leading_zero = edibble_labels_opt("leading_zero")
)

label_seq_from_length(
  from = 1L,
  length = 1L,
  by = 1L,
  prefix = "",
  suffix = "",
  sep_prefix = "",
  sep_suffix = "",
  leading_zero = edibble_labels_opt("leading_zero")
)

label_seq_to_length(
  to = 1L,
  length = 1L,
  by = 1L,
  prefix = "",
  suffix = "",
  sep_prefix = "",
  sep_suffix = "",
  leading_zero = edibble_labels_opt("leading_zero")
)

label_seq_length(
  length = 1L,
  prefix = "",
  suffix = "",
  sep_prefix = "",
```

```
  sep_suffix = "",
  leading_zero = edibble_labels_opt("leading_zero")
)
```

Arguments

`from`	An integer specifying the starting value (inclusive) of the sequence.
`to`	An integer specifying the ending value (inclusive) of the sequence.
`by`	An integer specifying the increment between values in the sequence.
`prefix`	A character string to be prepended to the labels.
`suffix`	A character string to be appended to the labels.
`sep_prefix`	A character string used to separate the prefix from the labels.
`sep_suffix`	A character string used to separate the suffix from the labels.
`leading_zero`	A logical value indicating whether to add leading zeros to the labels. If integer, then pad based on the number supplied.
`length`	An integer specifying the desired length of the sequence.

Value

A character vector containing the labels generated from the sequence.

Examples

```
label_seq_to_length(to = 10, length = 5, by = 2)
label_seq_from_to(from = 8, to = 10, leading_zero = 3)
label_seq_length(10, leading_zero = FALSE)
```

`lady_tasting_tea` *Lady tasting tea*

Description

Lady tasting tea experiment was described in Fisher (1935) to test the ability of a lady who said she tell whether the tea or milk was added first to a cup of tea.

The experiment consisted of preparing eight cups of tea, four with milk poured first and the other four with tea poured first. The lady has been told in advance that there are four of each kind of preparation.

This data consists of the same experimental structure and result but the order presented in practice is unknown.

cup The cup number.

first The cup of tea prepared with milk or tea first.

guess The guess by lady which one was poured first.

correct Whether the lady’s guess was correct.

Usage

```
lady_tasting_tea
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 8 rows and 4 columns.

Source

Fisher, Ronald (1935) The Design of Experiments.

See Also

Other experimental data: `skittles`

`latin` *Latin square designs and its generalisations as an array*

Description

Latin square designs and its generalisations as an array

Usage

```
latin_square(n, randomise = TRUE)

latin_rectangle(nr, nc, nt, randomise = TRUE)

latin_array(dim, nt, randomise = TRUE)
```

Arguments

`n, nt`	The number of treatments
`randomise`	A logical value to indicate whether the treatment allocation should be randomised. The default value is `TRUE`.
`nr`	The number of rows
`nc`	The number of columns
`dim`	A vector of integers to indicate the number of elements in each dimension.

Functions

- `latin_square()`: Latin square design
- `latin_rectangle()`: Like a Latin square design but allow different number of rows and columns
- `latin_array()`: Returns an array where it stitches up multiple Latin square/rectangle design

Examples

```
latin_square(n = 3)
latin_rectangle(3, 3, 3)
latin_array(c(3, 3, 3), 3)
```

`lvls` *Setting the traits of the levels*

Description

Setting the traits of the levels

Usage

```
lvls(value = NULL, n = NA_integer_, data = NULL, ...)
```

Arguments

`value`	A vector of the level values.
`n`	The number of replicate (if applicable).
`data`	A list or data frame of the same size as the `levels`.
`...`	Name-value pair denoting other level attributes. The value should be the same length as `levels` or a single value.

Value

An edbl_lvls object.

Examples

```
lvls(c("A", "B"))
```

`menu_bibd` *Balance incomplete block design*

Description

Some combinations of parameter values cannot create a balanced incomplete block design.

Usage

```
menu_bibd(
  t = random_integer_small(min = 3),
  k = random_integer_small(max = t - 1),
  r = random_integer_small(),
  seed = random_seed_number()
)
```

Arguments

`t`	The number of treatments.
`k`	The size of the block. This should be less than the number of treatments.
`r`	The number of replications for each treatment level.
`seed`	A scalar value for computational reproducibility.

Value

A recipe for balance incomplete block design.

See Also

Other recipe-designs: `menu_crd()`, `menu_factorial()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_split()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_bibd(t = 3, k = 2, r = 4)
```

`menu_crd` *Completely randomised design*

Description

Completely randomised design

Usage

```
menu_crd(
  t = random_integer_small(),
  n = random_integer_medium(min = t),
  r = NULL,
  seed = random_seed_number()
)
```

Arguments

`t`	The number of treatment levels
`n`	The number of experimental units
`r`	(Optional) The number of replicates.
`seed`	A scalar value for computational reproducibility.

Value

A recipe for completely randomised design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_factorial()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_split()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_crd(t = 3, n = 10)
```

`menu_factorial` *Prepare a factorial design*

Description

Prepare a factorial design

Usage

```
menu_factorial(
  trt = c(random_integer_small(), random_integer_small()),
  r = random_integer_small(),
  design = c("crd", "rcbd"),
  seed = random_seed_number()
)
```

Arguments

`trt`	A vector of the number of levels for each treatment factor.
`r`	The number of replications for each treatment level.
`design`	The unit structure: "crd" or "rcbd". The default is "crd".
`seed`	A scalar value for computational reproducibility.

Value

A recipe for factorial design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_split()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_factorial(trt = c(3, 2), r = 2, design = "crd")
```

`menu_graeco` *Graeco-Latin Square Design*

Description

Graeco-Latin Square Design

Usage

```
menu_graeco(t = random_integer_small(), seed = random_seed_number())
```

Arguments

`t`	The number of treatments.
`seed`	A scalar value for computational reproducibility.

Value

A recipe for Graeco-Latin square design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_factorial()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_split()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_graeco(t = 3)
```

`menu_hyper_graeco` *Hyper-Graeco-Latin Square Design*

Description

Hyper-Graeco-Latin Square Design

Usage

```
menu_hyper_graeco(t = random_integer_small(), seed = random_seed_number())
```

Arguments

`t`	The number of treatments
`seed`	A scalar value for computational reproducibility.

Value

A recipe Hyper-Graeco-Latin square design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_factorial()`, `menu_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_split()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_hyper_graeco(t = 3)
```

`menu_lsd` *Prepare classical Latin square design*

Description

Prepare classical Latin square design

Usage

```
menu_lsd(t = random_integer_small(), seed = random_seed_number())
```

Arguments

`t`	The number of treatments
`seed`	A scalar value for computational reproducibility.

Value

A recipe Latin square design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_factorial()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_rcbd()`, `menu_split()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_lsd(t = 3)
```

`menu_rcbd` *Prepare a randomised complete block design*

Description

Prepare a randomised complete block design

Usage

```
menu_rcbd(
  t = random_integer_small(),
  r = random_integer_small(),
  seed = random_seed_number()
)
```

Arguments

`t`	The number of treatments.
`r`	The number of replications for each treatment level.
`seed`	A scalar value for computational reproducibility.

Value

A recipe for randomised complete block design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_factorial()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_split()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_rcbd(t = 3, r = 2)
```

`menu_split` *Split-unit design*

Description

Originally referred to as split-plot design when it was first used.

Usage

```
menu_split(
  t1 = random_integer_small(),
  t2 = random_integer_small(),
  r = random_integer_small(),
  seed = random_seed_number()
)
```

Arguments

`t1`	The number of treatment levels for the main plots.
`t2`	The number of treatment levels for the subplots.
`r`	The number of replications for each treatment level.
`seed`	A scalar value for computational reproducibility.

Value

A recipe split-plot design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_factorial()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_strip()`, `menu_youden()`

Examples

```
menu_split(t1 = 3, t2 = 2, r = 4)
```

`menu_strip` *Strip-unit design*

Description

Strip-unit design

Usage

```
menu_strip(
  t1 = random_integer_small(),
  t2 = random_integer_small(),
  r = random_integer_small(),
  seed = random_seed_number()
)
```

Arguments

`t1`	The number of treatment levels for the main plots.
`t2`	The number of treatment levels for the subplots.
`r`	The number of replications for each treatment level.
`seed`	A scalar value for computational reproducibility.

Value

A recipe strip-unit design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_factorial()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_split()`, `menu_youden()`

Examples

```
menu_strip(t1 = 3, t2 = 3, r = 2)
```

`menu_youden` *Youden square design*

Description

Youden square design

Usage

```
menu_youden(
  nc = random_integer_small(),
  t = random_integer_small(min = nc + 1),
  seed = random_seed_number()
)
```

Arguments

`nc`	The number of columns.
`t`	The number of treatments.
`seed`	A scalar value for computational reproducibility.

Value

A recipe Youden square design.

See Also

Other recipe-designs: `menu_bibd()`, `menu_crd()`, `menu_factorial()`, `menu_graeco()`, `menu_hyper_graeco()`, `menu_lsd()`, `menu_rcbd()`, `menu_split()`, `menu_strip()`

Examples

```
menu_youden(nc = 4, t = 5)
```

`nested_in` *Specify the nesting or conditional structure for units or treatments*

Description

Conditional treatment is different to nested units as the levels are assumed to be distinct for the latter but not for the former.

Usage

```
nested_in(x, ...)

conditioned_on(x, ...)
```

Arguments

`x`	The name of the parent unit to nest under.
`...`	a single number OR a sequence of two-sided formula where the left-hand side corresponds to the name of the level (or the level number) of `x` and the right-hand side is an integer specifying the number of levels nested under the corresponding levels.

Details

Currently when specifying conditional treatment, only character vectors are accepted on the RHS.

Value

A nested level.

See Also

See `set_units()` for examples of how to use this.

Examples

```
design("Split-Plot Design | Split-Unit Design") %>%
  set_units(mainplot = 60,
            subplot = nested_in(mainplot, 10))
```

`nesting_structure` *Get the nesting structure for the units*

Description

Get the nesting structure for the units

Usage

```
nesting_structure(design)
```

Arguments

`design`	An edibble design

Value

Return a named list. Only shows the direct parent.

Examples

```
nesting_structure(takeout(menu_split()))
```

`new_edibble` *An edibble table constructor*

Description

This helps to construct a new edibble table which is a special type of tibble.

Usage

```
new_edibble(.data, ..., .design = NULL, .class = NULL)

as_edibble(.data, ...)
```

Arguments

`.data`	data frame or list of the same size.
`...`	Passed to `new_tibble`.
`.design`	An edibble graph object.
`.class`	Subclasses for edibble table. The default is NULL.

Value

An edibble table.

`order_trts` *A custom ordering algorithm*

Description

A custom ordering algorithm

Usage

```
order_trts(x, ...)
```

Arguments

`x`	A string specifying the class
`...`	Other arguments.

`plot.edbl_design` *Interactive plot of the edibble design*

Description

Interactive plot of the edibble design

Usage

```
## S3 method for class 'edbl_design'
plot(
  x,
  which = c("factors", "levels"),
  width = "100%",
  height = NULL,
  seed = 1,
  title = NULL,
  subtitle = NULL,
  footer = NULL,
  background = "transparent",
  view = c("show-buttons", "hide-buttons", "static"),
  ...
)

## S3 method for class 'edbl_table'
plot(x, ...)

plot_fct_graph(
  x,
```

```
  width = "100%",
  height = NULL,
  seed = 1,
  title = NULL,
  subtitle = NULL,
  footer = NULL,
  background = "transparent",
  view = c("show-buttons", "hide-buttons", "static"),
  ...
)

plot_lvl_graph(
  x,
  width = "100%",
  height = NULL,
  seed = 1,
  title = NULL,
  subtitle = NULL,
  footer = NULL,
  background = "transparent",
  view = c("show-buttons", "hide-buttons", "static"),
  ...
)
```

Arguments

`x`	An edibble design.
`which`	A string of either "factors" or "levels".
`width, height`	The width and height of the plot.
`seed`	A seed number so same plot is always generated.
`title, subtitle, footer`	The title, subtitle or footer of the plot. By default it uses the name from the `x` object as the title while rest is empty. To modify the look of the text, you can pass a character string consisting of valid for input style value in an HTML object, e.g. "font-size: 18px;font-family:serif;" as a named vector where the name corresponds to the text to display, e.g. `c("Title" = "font-size:20px;")`.
`background`	The background color of the plot. Default is transparent. The input can be a color name (e.g. "white"), a HEX value ("#FFFFFF"), or rgb/rgba in the format like rgba(0, 0, 0, 0).
`view`	A string of either "show-buttons" (default), "hide-buttons", "static"
`...`	Currently unused.

Value

A plot.

Examples

```
plot(takeout(menu_crd(t = 4, n = 20)))
```

`Provenance` *An object to query, record and modify an edibble graph*

Description

An object to query, record and modify an edibble graph

An object to query, record and modify an edibble graph

Details

The Provenance contains a set of operations to manipulate the nodes and edges of the edibble graph object.

Active bindings

`fct_nodes` Get the factor nodes

`lvl_nodes` Get the level nodes

`fct_edges` Get the factor edges

`lvl_edges` Get the level edges

`fct_n` Get the number of nodes in factor graph

`lvl_n` Get the number of nodes in level graph

`rcrd_ids` Get the ids for all edbl_rcrd factors.

`unit_ids` Get the ids for all edbl_unit factors.

`trt_ids` Get the ids for all edbl_trt factors.

`is_connected` Check if nodes are connected. Get a new factor id. Get a new level id. Given a particular DAG, return a topological order Remember that there could be more than one order.

Methods

Public methods:

- `Provenance$new()`
- `Provenance$set_title()`
- `Provenance$set_name()`
- `Provenance$set_validation()`
- `Provenance$set_simulate()`
- `Provenance$reactivate()`
- `Provenance$deactivate()`
- `Provenance$fct_id()`
- `Provenance$fct_id_parent()`

- `Provenance$fct_id_child()`
- `Provenance$fct_id_ancestor()`
- `Provenance$fct_id_descendant()`
- `Provenance$fct_id_leaves()`
- `Provenance$lvl_id()`
- `Provenance$lvl_id_parent()`
- `Provenance$lvl_id_child()`
- `Provenance$lvl_id_ancestor()`
- `Provenance$fct_id_from_lvl_id()`
- `Provenance$fct_id_from_lvl_values()`
- `Provenance$lvl_id_from_fct_id()`
- `Provenance$fct_names()`
- `Provenance$unit_names()`
- `Provenance$trt_names()`
- `Provenance$rcrd_names()`
- `Provenance$rcrd_class()`
- `Provenance$lvl_values()`
- `Provenance$unit_values()`
- `Provenance$trt_values()`
- `Provenance$rcrd_values()`
- `Provenance$fct_role()`
- `Provenance$fct_levels()`
- `Provenance$fct_levels_id_to_edbl_fct()`
- `Provenance$fct_levels_id_to_value()`
- `Provenance$fct_levels_value_to_id()`
- `Provenance$fct_exists()`
- `Provenance$trt_exists()`
- `Provenance$unit_exists()`
- `Provenance$rcrd_exists()`
- `Provenance$append_fct_nodes()`
- `Provenance$append_lvl_nodes()`
- `Provenance$append_fct_edges()`
- `Provenance$append_lvl_edges()`
- `Provenance$serve_units()`
- `Provenance$serve_trts()`
- `Provenance$serve_rcrds()`
- `Provenance$make_trts_table()`
- `Provenance$graph_subset()`
- `Provenance$save_seed()`
- `Provenance$get_title()`
- `Provenance$get_validation()`
- `Provenance$get_trail()`

- `Provenance$get_graph()`
- `Provenance$get_seed()`
- `Provenance$get_session_info()`
- `Provenance$get_edibble_version()`
- `Provenance$get_simulate()`
- `Provenance$get_simulate_result_env()`
- `Provenance$mapping()`
- `Provenance$mapping_to_unit()`
- `Provenance$record_step()`
- `Provenance$lvl_mapping()`
- `Provenance$record_track_external()`
- `Provenance$fct_id_links()`
- `Provenance$fct_graph_components()`
- `Provenance$lvl_graph_components()`
- `Provenance$clone()`

Method `new()`**:** Initialise function

Usage:

```
Provenance$new(graph = NULL)
```

Arguments:

`graph` An edibble graph.

Method `set_title()`**:** Set the title.

Usage:

```
Provenance$set_title(title)
```

Arguments:

`title` The title of the experiment

Method `set_name()`**:** Set the name.

Usage:

```
Provenance$set_name(name)
```

Arguments:

`name` The name of the edibble graph object.

Method `set_validation()`**:** Set the validation.

Usage:

```
Provenance$set_validation(validation, type = "rcrds")
```

Arguments:

`validation` The validation statement.

`type` The type of validation.

Method `set_simulate()`**:** Set the simulation process

Usage:

```
Provenance$set_simulate(name, process, rcrds)
```

Arguments:

`name` The name of the process

`process` A function to simulate the record

`rcrds` The record factor name simulating for.

Method `reactivate()`**:** Reactivate the graph in the provenance object.

Usage:

```
Provenance$reactivate(
  design,
  overwrite = c("graph", "anatomy", "recipe", "validation", "simulate",
    "simualte_result")
)
```

Arguments:

`design` An edibble design

`overwrite` A vector of character to overwrite from the supplied design object.

Method `deactivate()`**:** Deactivate the provenance object.

Usage:

```
Provenance$deactivate(delete = c("graph", "anatomy", "recipe", "validation"))
```

Arguments:

`delete` A vector of character to delete.

Method `fct_id()`**:** Get the id based on either the name of the factor node. If none supplied then it will give all.

Usage:

```
Provenance$fct_id(name = NULL, role = NULL)
```

Arguments:

`name` The name of the node.

`role` The role for the node.

Method `fct_id_parent()`**:** Get the factor parent ids

Usage:

```
Provenance$fct_id_parent(id = NULL, role = NULL, type = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

`type` The type of edge link.

Method `fct_id_child()`**:** Get the factor child ids. If `role` is supplied then the child has to fit `role`

Usage:

```
Provenance$fct_id_child(id = NULL, role = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

Method `fct_id_ancestor()`**:** Get the factor ancestor ids

Usage:

```
Provenance$fct_id_ancestor(id = NULL, role = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

Method `fct_id_descendant()`**:** Get the factor descendant ids

Usage:

```
Provenance$fct_id_descendant(id = NULL, role = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

Method `fct_id_leaves()`**:** Get the leave factor ids.

Usage:

```
Provenance$fct_id_leaves(role = NULL)
```

Arguments:

`role` The role for the node.

Method `lvl_id()`**:** Get the id based on name of level node. Assumes that level ids obtained are all from the same fid

Usage:

```
Provenance$lvl_id(value = NULL, role = NULL, fid = NULL)
```

Arguments:

`value` The value of the node.

`role` The role for the node.

`fid` The factor id.

Method `lvl_id_parent()`**:** Get the level parent ids

Usage:

```
Provenance$lvl_id_parent(id = NULL, role = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

Method `lvl_id_child()`**:** Get the level child ids

Usage:

```
Provenance$lvl_id_child(id = NULL, role = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

Method `lvl_id_ancestor()`**:** Get the level ancestor ids

Usage:

```
Provenance$lvl_id_ancestor(id = NULL, role = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

Method `fct_id_from_lvl_id()`**:** Find the factor id from level ids.

Usage:

```
Provenance$fct_id_from_lvl_id(id = NULL, fid_search = NULL)
```

Arguments:

`id` The id of the corresponding node.

`fid_search` A vector of fids to search from.

Method `fct_id_from_lvl_values()`**:** Find the factor id from level values.

Usage:

```
Provenance$fct_id_from_lvl_values(value = NULL, fid_search = NULL)
```

Arguments:

`value` The value of the node.

`fid_search` A vector of fids to search from.

Method `lvl_id_from_fct_id()`**:** Find the level id from the given fid

Usage:

```
Provenance$lvl_id_from_fct_id(fid = NULL)
```

Arguments:

`fid` The factor id.

Method `fct_names()`**:** Get the factor names based on id or role

Usage:

```
Provenance$fct_names(id = NULL, role = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

Method `unit_names()`**:** Get the unit names

Usage:

```
Provenance$unit_names(id = NULL)
```

Arguments:

`id` The id of the corresponding node.

Method `trt_names()`**:** Get the treatment names

Usage:

```
Provenance$trt_names(id = NULL)
```

Arguments:

`id` The id of the corresponding node.

Method `rcrd_names()`**:** Get the record names.

Usage:

```
Provenance$rcrd_names(id = NULL)
```

Arguments:

`id` The id of the corresponding node.

Method `rcrd_class()`**:** Get the class for record with validation.

Usage:

```
Provenance$rcrd_class(name = NULL)
```

Arguments:

`name` The name of the node.

Method `lvl_values()`**:** Get the level values based on id or role cannot have just role only defined. id must be from the same fid

Usage:

```
Provenance$lvl_values(id = NULL, role = NULL, fid = NULL)
```

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

`fid` The factor id.

Method `unit_values()`**:** Get the unit values.

Usage:

```
Provenance$unit_values(id = NULL, fid = NULL)
```

Arguments:

`id` The id of the corresponding node.

`fid` The factor id.

Method `trt_values()`**:** Get the treatment values.

Usage:

```
Provenance$trt_values(id = NULL, fid = NULL)
```

Arguments:

`id` The id of the corresponding node.

`fid` The factor id.

Method `rcrd_values()`**:** Get the record values.

Usage:

```
Provenance$rcrd_values(uid = NULL, fid = NULL)
```

Arguments:

`uid` The unit level id

`fid` The factor id.

Method `fct_role()`**:** Get the role of the vertex given the factor id

Usage:

```
Provenance$fct_role(id = NULL)
```

Arguments:

`id` The id of the corresponding node.

Method `fct_levels()`**:** Get the levels for each factor

Usage:

```
Provenance$fct_levels(id = NULL, name = NULL, return = c("id", "value"))
```

Arguments:

`id` The id of the corresponding node.

`name` The name of the node.

`return` To return in "id" or "value" format.

Method `fct_levels_id_to_edbl_fct()`**:** Factor levels to edble factor

Usage:

```
Provenance$fct_levels_id_to_edbl_fct(fct_levels, role)
```

Arguments:

`fct_levels` The factor levels in id.

`role` The role for the node.

Method `fct_levels_id_to_value()`**:** Get the factor levels in value given id format

Usage:

```
Provenance$fct_levels_id_to_value(fct_levels)
```

Arguments:

`fct_levels` A list of factor levels in id format.

Method `fct_levels_value_to_id()`**:** Get the factor levels in id given value format.

Usage:

```
Provenance$fct_levels_value_to_id(fct_levels)
```

Arguments:

`fct_levels` A list of factor levels in id format.

Method `fct_exists()`**:** One of `name`, `id` or `role` is defined to check if it exists. If more than one of the arguments `name`, `id` and `role` are supplied, then the intersection of it will be checked.

Usage:

```
Provenance$fct_exists(id = NULL, name = NULL, role = NULL, abort = TRUE)
```

Arguments:

`id` The id of the corresponding node.

`name` The name of the node.

`role` The role for the node.

`abort` Whether to abort.

Method `trt_exists():` Check if treatment exists.

Usage:

```
Provenance$trt_exists(id = NULL, name = NULL, abort = TRUE)
```

Arguments:

`id` The id of the corresponding node.

`name` The name of the node.

`abort` Whether to abort.

Method `unit_exists():` Check if unit exists.

Usage:

```
Provenance$unit_exists(id = NULL, name = NULL, abort = TRUE)
```

Arguments:

`id` The id of the corresponding node.

`name` The name of the node.

`abort` Whether to abort.

Method `rcrd_exists():` Check if record exists.

Usage:

```
Provenance$rcrd_exists(id = NULL, name = NULL, abort = TRUE)
```

Arguments:

`id` The id of the corresponding node.

`name` The name of the node.

`abort` Whether to abort.

Method `append_fct_nodes():` Given node data, append the factor nodes

Usage:

```
Provenance$append_fct_nodes(name, role, attrs = NULL)
```

Arguments:

`name` The name of the node.

`role` The role for the node.

`attrs` The attributes.

Method `append_lvl_nodes():` Given node data, append the level nodes

Usage:

```
Provenance$append_lvl_nodes(
  value,
  n = NULL,
  label = NULL,
  attrs = NULL,
  fid = NULL
)
```

Arguments:

`value` The value of the node.

`n` The number of replications.

`label` The labels for the levels.

`attrs` The attributes.

`fid` The factor id.

Method `append_fct_edges()`**:** Given edge data, append the factor edges

Usage:

```
Provenance$append_fct_edges(from, to, type = NULL, group = FALSE, attrs = NULL)
```

Arguments:

`from` The node id from.

`to` The node id to.

`type` The type of edges.

`group` A logical value to indicate whether to create new group id or not.

`attrs` The attributes.

Method `append_lvl_edges()`**:** Given edge data, append the level edges

Usage:

```
Provenance$append_lvl_edges(from, to, attrs = NULL)
```

Arguments:

`from` The node id from.

`to` The node id to.

`attrs` The attributes.

Method `serve_units()`**:** Serve the units.

Usage:

```
Provenance$serve_units(id = NULL, return = c("id", "value"))
```

Arguments:

`id` The id of the corresponding node.

`return` To return in "id" or "value" format.

Method `serve_trts()`**:** Serve treatments

Usage:

```
Provenance$serve_trts(id = NULL, return = c("id", "value"))
```

Arguments:

`id` The id of the corresponding node.

`return` To return in "id" or "value" format.

Method `serve_rcrds()`**:** Serve records

Usage:

```
Provenance$serve_rcrds(id = NULL, return = c("id", "value"))
```

Arguments:

`id` The id of the corresponding node.

`return` To return in "id" or "value" format.

Method `make_trts_table()`**:** Make the treatments table

Usage:

```
Provenance$make_trts_table(id = NULL, return = c("id", "value"))
```

Arguments:

`id` The id of the corresponding node.

`return` To return in "id" or "value" format.

Returns: A treatment table

Method `graph_subset()`**:** Subset graph

Usage:

```
Provenance$graph_subset(
  id = NULL,
  include = c("self", "child", "parent", "ancestors")
)
```

Arguments:

`id` The id of the corresponding node.

`include` "self" for only input id, "child" for child also, "parent" for parent also, nodes immediately related, and "ancestors" for all ancestors

Returns: subsetted graph

Method `save_seed()`**:** Save the seed

Usage:

```
Provenance$save_seed(seed, type)
```

Arguments:

`seed` A seed.

`type` Type.

Method `get_title()`**:** Get the title

Usage:

```
Provenance$get_title()
```

Method `get_validation()`**:** Get the validation

Usage:

```
Provenance$get_validation(type = NULL)
```

Arguments:

`type` A type.

Method `get_trail():` Get the trail.

Usage:

```
Provenance$get_trail()
```

Method `get_graph():` Get the graph

Usage:

```
Provenance$get_graph()
```

Method `get_seed():` Get the seed

Usage:

```
Provenance$get_seed()
```

Method `get_session_info():` Get the session information

Usage:

```
Provenance$get_session_info()
```

Method `get_edibble_version():` Get the edibble version.

Usage:

```
Provenance$get_edibble_version()
```

Method `get_simulate():` Get the simulation information

Usage:

```
Provenance$get_simulate(name = NULL)
```

Arguments:

`name` The process name. Only one name allowed.

Method `get_simulate_result_env():` Get the simulation results

Usage:

```
Provenance$get_simulate_result_env(name = NULL)
```

Arguments:

`name` The process name. Only one name allowed.

Method `mapping():` Mapping of a role to role

Usage:

```
Provenance$mapping(role_from, role_to)
```

Arguments:

`role_from` The role from.

`role_to` The role to.

Method `mapping_to_unit():` Mapping of an id to a unit

Usage:

`Provenance$mapping_to_unit(id = NULL)`

Arguments:

`id` The id of the corresponding node.

Method `record_step()`**:** Record step.

Usage:

`Provenance$record_step()`

Method `lvl_mapping()`**:** Get the level edges by factor

Usage:

`Provenance$lvl_mapping(from, to, return = c("vector", "table"))`

Arguments:

`from, to` The factor id.

`return` To return in "id" or "value" format.

Method `record_track_external()`**:** Record track external.

Usage:

`Provenance$record_track_external(code)`

Arguments:

`code` The code to record.

Method `fct_id_links()`**:** Find all id that is linked.

Usage:

`Provenance$fct_id_links(id = NULL, role = NULL, link = c("direct", "indirect"))`

Arguments:

`id` The id of the corresponding node.

`role` The role for the node.

`link` Whether the link should be direct or indirect

Returns: id of linked factors, excluding itself.

Method `fct_graph_components()`**:** Get the nodes with components (subgraph number)

Usage:

`Provenance$fct_graph_components(id = NULL)`

Arguments:

`id` The id of the corresponding node.

Method `lvl_graph_components()`**:** Get the nodes with components (subgraph number)

Usage:

`Provenance$lvl_graph_components()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`Provenance$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`rescale_values` *Rescale a numerical vector*

Description

Similar to `scales::rescale()` but it has a different behaviour when only upper or lower bound is given.

Usage

```
rescale_values(x, lower = NA, upper = NA)
```

Arguments

`x`	A numerical vector.
`lower`	The lower bound.
`upper`	The upper bound.

`scan_menu` *Find the short names of the named designs*

Description

Find the short names of the named designs

Usage

```
scan_menu(packages = NULL, exclude = NULL)
```

Arguments

`packages`	A character vector containing the package names to search named designs from. By default it will search edibble and other packages loaded.
`exclude`	A character vector denoting the packages to exclude search from.

Value

A data.frame with package, name, arguments, and full name.

Examples

```
scan_menu()
```

`serve_table` *Serve edibble table*

Description

This converts an edibble graph object to a data frame called edibble. This function should be used when the design is in the final form (or close to the final form). The table can only be formed when the variables can be reconciled, otherwise it will be a data frame with zero rows.

Usage

```
serve_table(
  .edibble = NULL,
  label_nested = NULL,
  fail = c("error", "warn", "ignore"),
  .record = TRUE
)
```

Arguments

`.edibble`	An edibble design (`edbl_design`), an edibble data frame (`edbl_table`) or an object that contains the edibble data frame in the attribute `design`.
`label_nested`	The columns to show nested labels (if available). Tidyselect compatible.
`fail`	What to do when failing to convert graph to table.
`.record`	A logical value. This indicates whether to record this code step. The default is TRUE. It should remain TRUE unless this function is used as a wrapper in other code.

Value

An `edbl` data frame with columns defined by vertices and rows displayed only if the vertices are connected and reconcile for output.

See Also

Other user-facing functions: `allot_trts()`, `allot_units()`, `design()`, `expect_rcrds()`, `export_design()`, `set_rcrds()`, `set_trts()`, `set_units()`

Examples

```
design("Completely Randomised Design") %>%
  set_units(unit = 28) %>%
  set_trts(trt = 6) %>%
  allot_trts(trt ~ unit) %>%
  assign_trts("random", seed = 521) %>%
  serve_table()
```

`set_attrs` *Set the experimental context as metadata*

Description

These are structured information that can be encoded in into the design object. By encoding this information, you can make it interoperable. If you use `export_design()`, the information is exported to the title sheet of the excel output.

Usage

```
set_attrs(.edibble = design(), ...)
```

Arguments

`.edibble`	An edibble table or design.
`...`	A series of name-value pairs where the name corresponds to the name of the metadata nad the value corresponds to the actual metadata value. If the name is omitted, then no name to the metadata is assigned for the corresponding value.

Examples

```
des <- set_attrs(design(aim = "Testing for new flu vaccine.",
                        contact = "emi.tanaka (at) anu.edu",
                       "Funded by Better Experiments Institute.") )

des$context
```

`set_rcrds` *Set records for given unit*

Description

This function creates new nodes to edibble graph with the name corresponding to either the intended response that will be measured or a variable to be recorded. Avoid record names staring with a "." as these are reserved for other purposes downstream.

Usage

```
set_rcrds(
  .edibble = NULL,
  ...,
  .name_repair = c("check_unique", "unique", "universal", "minimal"),
  .record = TRUE
)

set_rcrds_of(.edibble = NULL, ...)
```

Arguments

`.edibble`	An edibble design (`edbl_design`), an edibble data frame (`edbl_table`) or an object that contains the edibble data frame in the attribute `design`.
`...`	Name-value pair. The value should correspond to a single name of the unit defined in `set_units`. The name should be the name of the record variable.
`.name_repair`	Same as the argument in `tibble::tibble()`.
`.record`	A logical value. This indicates whether to record this code step. The default is TRUE. It should remain TRUE unless this function is used as a wrapper in other code.

Value

An edibble design.

See Also

Other user-facing functions: `allot_trts()`, `allot_units()`, `design()`, `expect_rcrds()`, `export_design()`, `serve_table()`, `set_trts()`, `set_units()`

Examples

```
takeout(menu_crd(t = 4, n = 10)) %>%
  set_rcrds(y = unit)

takeout(menu_crd(t = 4, n = 10)) %>%
  set_rcrds_of(unit = "y")
```

`set_trts` *Set the treatment variables*

Description

This function add a special class, called `edbl_trt`, of edibble variables.

Usage

```
set_trts(
  .edibble = design(),
  ...,
  .name_repair = c("check_unique", "unique", "universal", "minimal"),
  .record = TRUE
)
```

Arguments

`.edibble`	An edibble design (`edbl_design`), an edibble data frame (`edbl_table`) or an object that contains the edibble data frame in the attribute `design`.
`...`	Either a name-value pair or a series of the names.
`.name_repair`	Same as the argument in `tibble::tibble()`.
`.record`	A logical value. This indicates whether to record this code step. The default is TRUE. It should remain TRUE unless this function is used as a wrapper in other code.

Value

An edibble design.

Definition of *treatment*

The word *treatment* is sometimes used to refer to one of these variables. When there are more than one treatment variables then this unfortunately confuses whether treatment refers to the variable or the combination of all treatment variables.

Treatment is the whole description of what is applied in an experiment.

See Also

Other user-facing functions: `allot_trts()`, `allot_units()`, `design()`, `expect_rcrds()`, `export_design()`, `serve_table()`, `set_rcrds()`, `set_units()`

Examples

```
design() %>%
  set_trts(pesticide = c("A", "B", "C"),
           dosage = c(0, 10, 20, 30, 40))
```

`set_units` *Set units used in experiment*

Description

This function sets new edibble variables of class `edbl_unit`. More specifically, this means that new nodes are added to the `edbl_graph`.

Usage

```
set_units(
  .edibble = design(),
  ...,
  .name_repair = c("check_unique", "unique", "universal", "minimal"),
  .record = TRUE
)
```

Arguments

`.edibble`	An edibble design (`edbl_design`), an edibble data frame (`edbl_table`) or an object that contains the edibble data frame in the attribute `design`.
`...`	Either a name-value pair or a series of the names.
`.name_repair`	Same as the argument in `tibble::tibble()`.
`.record`	A logical value. This indicates whether to record this code step. The default is TRUE. It should remain TRUE unless this function is used as a wrapper in other code.

Value

An edibble design.

Definition of *unit*

A *unit*, much like *factor*, is an over-used word but due to lack of a better word, edibble uses the word "unit" to refer to any entity, physical or otherwise, that pertain to the experiment. This function doen't explicitly distinguish between experimental or observational units, nor is a unit limited to these type of units. A unit in edibble can be a blocking factor or even a discrete time unit.

Limitations

Currently a unit should only have a discrete set of levels and you need to know the number of levels prior to setting the units.

See Also

Other user-facing functions: `allot_trts()`, `allot_units()`, `design()`, `expect_rcrds()`, `export_design()`, `serve_table()`, `set_rcrds()`, `set_trts()`

Examples

```
# 30 rats
design() %>%
  set_units(rat = 30) %>%
  serve_table()

# 4 girls named "Anna", "Betty", "Carol", "Diana"
design() %>%
  set_units(girl = c("Anna", "Betty", "Carol", "Diana")) %>%
```

```
  serve_table()

# 3 companies, with 10 boxes each
design() %>%
  set_units(company = c("A", "B", "C"),
                box = nested_in(company, 10))

# 2 classes, one with 10 students, the other with 20 students
design() %>%
  set_units(class = 2,
            student = nested_in(class,
                                1 ~ 10,
                                2 ~ 20))

# 4 countries with 10 people from Australia & New Zealand and 20 from the rest
design() %>%
  set_units(country = c("AU", "NZ", "USA", "JPN"),
            person = nested_in(country,
                               c("AU", "NZ") ~ 10,
                                           . ~ 20)) %>%
  serve_table()
```

`simulate_process` *Simulation process*

Description

This function to create and store functions to simulate the records.

Usage

```
simulate_process(.data, ...)
```

Arguments

`.data`	An edibble table.
`...`	A name-value pair where the name should correspond to either the record name that you are simulating or a process name if the return object is a data frame with columns corresponding to the name of the records. The value must be a function with set default arguments. The return object of this function should be either a vector or a data frame with the column names corresponding to the record names. The size should correspond to the number of columns.

Details

When creating a function, internally you can refer to any of the factors without referring to the actual data. The data referred to is expected to be from the full data. Like in tidyverse, syntax `.data` is reserved for the full data and `.env` can be used to refer to environment variables.

You can use the syntax `n()` to refer to `nrow(.data)` or `n(fct)` where `fct` corresponds to unquoted factor name. The return value will be the number of the observed number of levels of factor `fct` in the data. For `n(fct1, fct2)` it will return the observed number of distinct interaction levels for `fct1` and `fct2`.

Note that you can actually put as many process as you like if you use a process name (starting with a dot), even if this is for the same record factor.

`simulate_rcrds` *Simulate records*

Description

Simulate records

Usage

```
simulate_rcrds(.data, ..., .seed = NULL, .nsim = 1L)
```

Arguments

`.data`	An edibble data
`...`	A name-value pair where the name should correspond to the names used in the `simulate_process()`. The value should be returned from calling `with_params()`.
`.seed`	An optional seed value.
`.nsim`	The number of times to simulate data.

Examples

```
design() %>%
 set_units(unit = 4) %>%
 set_trts(trt = 2) %>%
 allot_table(trt ~ unit) %>%
 set_rcrds(y = unit) %>%
 simulate_process(y = function() {
            res <- rnorm(n())
            res
 }) %>%
 simulate_rcrds(y = with_params(), .nsim = 3)
```

`skittles` *Skittles experiment*

Description

This contains the data from the skittle experiment conducted by Nick Tierney. The goal of the experiment was to assess if people can discern the flavour of the skittle (indicated by color of the skittle) based on taste alone. The participants are blindfolded.

The experiment had 3 participants with each participant tasting 10 skittles, 2 of each 5 color, in a random order.

skittle_type The type of skittle. Coincides with `real_skittle`.

person The participant.

order The order the skittle was tasted.

choice The participant's choice.

real_skittle The actual skittle color.

Usage

```
skittles
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 30 rows and 6 columns.

Source

https://github.com/njtierney/skittles

See Also

Other experimental data: `lady_tasting_tea`

`split_by` *Split or count the data according to certain factors*

Description

This function has a similar result with `split()` where it returns a named list with names corresponding to the levels of the separating factor (or concatenated strings if multiple separating factors). The key differences to `split()`, are that the splitting factor does not appear in the elements of the list and only linked factors and their ancestors appear in the output, e.g. if treatment is applied to wholeplot and subplots are nested within subplots, then the subplot will not be shown in the output if split by treatment.

Usage

```
split_by(.data, ..., .sep = ":", .remove_empty = TRUE)

count_by(.data, ..., .remove_empty = TRUE)
```

Arguments

`.data`	An edibble table.
`...`	The factors to split or count by. You cannot split by a record factor or a factor that uniquely indexes the smallest unit in the design. You cannot also combine treatment and unit factors together.
`.sep`	The separator to use if more than one factor to split by.
`.remove_empty`	Remove empty combinations. Default is TRUE.

Value

A named list.

See Also

`pivot_wider_by()`

Examples

```
spd <- takeout(menu_split())
split(spd, spd$trt1)
spd %>% split_by(trt1)
spd %>% split_by(trt2)
spd %>% split_by(mainplot)
spd %>% count_by(trt1)

fac <- takeout(menu_factorial(trt = c(2, 2, 2)))
fac %>% count_by(where(~is_trt(.x)))
```

`takeout` *Create a named experimental design*

Description

This function generates a named experimental design by supplying the selected menu named design and prints out by default

You can find the available recipes with `scan_menu()`.

Usage

```
takeout(recipe = NULL, show = TRUE)
```

Arguments

`recipe`	A named design object. This should be typically generated from a function with prefix `menu_`. If nothing is supplied, it will randomly select one.
`show`	A logical value to indicate whether the code should be shown or not. Default is TRUE.

Value

A recipe design.

See Also

See `scan_menu()` for finding the short names of the named experimental designs.

Examples

```
takeout(menu_crd(n = 50, t = 5))
# if you omit the design parameters then it will use the default
# (which may be random)
takeout(menu_crd())
# if you don't give any short names then it will generate a random one
takeout()
```

`trts_table` *Treatments table*

Description

Treatments table

Usage

```
trts_table(.edibble)
```

Arguments

`.edibble`	An edibble table

`utility-edibble-var` *Utility functions for edibble variable*

Description

The S3 methods for `edbl_fct` objects have the same expected output that of a factor.

Other functions are utility functions related to `edbl_fct` object.

Usage

```
## S3 method for class 'edbl_fct'
as.character(x, ...)

## S3 method for class 'edbl_fct'
as.integer(x, ...)

is_fct(x)

is_unit(x)

is_trt(x)

is_rcrd(x)
```

Arguments

`x`	An `edbl_fct` object.
`...`	Ignored.

Value

A character vector.

`with_params` *This is a helper function to set the parameter values*

Description

This is a helper function to set the parameter values

Usage

```
with_params(..., .censor = NA, .aggregate = NULL)
```

Arguments

`...`	A series of name-value pair that are inputs used for the simulation process.
`.censor`	The value to censor if it outside the valid values. If the value has a lower and upper bound then it should be a vector of size 2. Use -Inf or Inf if you don't want to censor either value. You can use a list if you want a different censoring for different records where the name corresponds to the name of the record. If you want to apply a default value/function for censoring then use the name ".default". You can use a function instead of a value. The function may be specified by as a lambda function. The object `.lower` and `.upper` are special reserved values, corresponding to the limits given from valid values, that can be used within this function.
`.aggregate`	The function for aggregation if the response values differ within the same unit level for the record. Use `NA` if you don't want to aggregate. By default, it will get the mean or mode depending on the encoding (numeric is mean, mode for character or factor), or if absent, based on returned encoding. It can be a named list where the names correspond to the record name and the values corresponding to a function.

See Also

`simulate_rcrds()`

`with_value` *Validation values*

Description

This creates a list that is used later for creating data validation rules when the data is exported.

Usage

```
with_value(
  operator = c("=", "==", ">=", "<=", "<", ">", "!="),
  value = NULL,
  between = NULL,
  not_between = NULL
)
```

Arguments

`operator`	Operator to apply.
`value`	An optional value related to operator
`between, not_between`	An optional numerical vector of size two where the first entry is the minimum value and the second entry is the maximum value. For `between`, the value is valid if within the range of minimum and maximum value inclusive. For `not_between`, the value must lie outside of these values.

Value

A list with two elements `operator` and `value`.

`with_variables` *A helper function to set variables that the record is dependent on.*

Description

The other options give are characteristics of the record (not the independent variables). Warning: none of the other options work at the moment!

Usage

```
with_variables(
  ...,
  .missing = FALSE,
  .interaction = random_true_false(),
  .discrete = FALSE,
  .linear = random_true_false(),
  .error_dist = NULL
)
```

Arguments

`...`	A series of factors in which the record is explicitly dependent upon (tidyselect campatible).
`.missing`	A logical value indicating whether there should be some missing values. Default is FALSE. The missing values are introduced at random. It can also be numeric of between 0 and 1 giving the proportion of missing values.
`.interaction`	Whether there should be treatment interaction effects.
`.discrete`	Whether to make the response value discrete or not.
`.linear`	Whether to include non-linear term or not. The value is always additive.
`.error_dist`	The random distribution to use for numerical values (either "normal", "uniform", "exponential", "gamma", "beta", "cauchy", "chisq", "f", "t", "poisson", "weibull"). The default choice is random out of these with higher chances of "normal".

See Also

`autofill_rcrds()`

Index