Package ‘edibble’

August 26, 2022

Title Designing Comparative Experiments

Version 0.1.1

Description A system to facilitate designing 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.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

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

Suggests testthat (>= 3.0.0), rmarkdown, openxlsx, visNetwork, covr

Depends R (>= 2.10)

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 2022-08-26 06:22:35 UTC

R topics documented:

edibble-package *edibble: Designing Comparative Experiments*

Description

A system to facilitate designing 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.

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`

allot — *Define the possible allocation of treatments to units*

Description

This function adds the edges between variable nodes to specify the mapping of units to treatment. This function does not actually assign specific treatment levels onto actual units.

Usage

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

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

allot_table(
  .edibble,
  ...,
  order = "random",
  seed = NULL,
  constrain = nesting_structure(.edibble)
)
```

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`.
`...`	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.
`order`	A character vector signifying the apportion of treatments to units. The value should be either "random", "systematic" or "systematic-random". "random" allocates the treatment randomly to units based on specified allotment with restrictions implied by unit structure. "systematic" allocates the treatment in a systematic order to units. "systematic-random" 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.

Value

Return an edibble design.

See Also

assign

Other user-facing functions: `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)
```

`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

```
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))
anatomy(split)
```

as.data.frame.edbl_table

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

assign

Assign treatments or units to units

Description

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

Usage

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

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

Arguments

`.design`	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" or "systematic-random". "random" allocates the treatment randomly to units based on specified allotment with restrictions implied by unit structure. "systematic" allocates the treatment in a systematic order to units. "systematic-random" 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_data_frame` *Convert an edibble data frame to normal data frame*

Description

A patch function where there is an issue with edbl factors

Usage

```
as_data_frame(.data)
```

Arguments

`.data`	can be a list or data frame

Value

A data.frame.

`cook_design` *Cook the design in the kitchen*

Description

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

Usage

```
cook_design(x)
```

Arguments

`x`	An edibble object.

Value

A Kitchen object.

Examples

```
cook_design(takeout())
```

`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(
  ...,
  prefix = NULL,
  suffix = NULL,
  leading0 = NULL,
  sep = NULL,
  attrs = NULL
)
```

Arguments

`...`	a sequence of units
`prefix`	Currently not implemented.The prefix of the label.
`suffix`	Currently not implemented.The suffix of the label.
`leading0`	Currently not implemented.Whether there should be a leading 0 if labels are made.
`sep`	Currently not implemented.A separator added between prefix and the number if prefix is empty.
`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(name = NULL, .record = TRUE, seed = NULL, kitchen = Kitchen)

redesign(
  .data,
  name = NULL,
  .record = TRUE,
  seed = NULL,
  kitchen = Kitchen,
  ...
)
```

Arguments

`name`	Optional name used as title for printing the design.
`.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.
`kitchen`	An environment setup in a manner to manipulate, extract and query information on the design.
`.data`	An edibble table.
`...`	Either a name-value pair or a series of the names.

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`, `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_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(edibble)

fct_edges(edibble)

lvl_nodes(edibble)

lvl_edges(edibble)
```

Arguments

`edibble`	An edibble object.

`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, ...)
```

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`.

Value

An edibble design.

See Also

Other user-facing functions: allot, 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, date = Sys.Date(), overwrite = FALSE)
```

Arguments

.data	An edibble data frame or design.
file	File, including the path, to export the data to.
author	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.
overwrite	A logical indicating whether to overwrite exisitng file or not.

Value

The input data object.

See Also

Other user-facing functions: allot, design(), expect_rcrds(), serve_table(), set_rcrds(), set_trts(), set_units()

fct_attrs *Setting the traits of factors*

Description

This function is used to set characteristics of the factors.

Usage

```
fct_attrs(
  levels = NULL,
  label = NULL,
  description = NULL,
  unit_of_measure = NULL,
  class = NULL,
  ...
)
```

Arguments

`levels`	An `edbl_lvls` object that should contain information about the levels in the factor.
`label`	A string that denotes the long name of the factor.
`description`	The text description of the factor.
`unit_of_measure`	A string denoting the unit of measurement if applicable.
`class`	An optional subclass.
`...`	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.

Value

An `edbl_lvls` object.

See Also

lvl_traits

Examples

```
fct_attrs(levels = c("A", "B"))
```

`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.

`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())
```

`Kitchen` *A manipulator for the edbl_design.*

Description

A manipulator for the edbl_design.

A manipulator for the edbl_design.

Details

Internal functions should create a new Kitchen object. The Kitchen contains a set of operations to manipulate the nodes and edges of the edibble design object.

Public fields

`design` An edibble design object Initialise function

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

`fct_last_id` Get the last factor id.

`lvl_last_id` Get the last level id.

`fct_leaves` Get the leave factor ids.

`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.

`trt_names` Get the node labels for treatments

`unit_names` Get the node labels for units

`rcrd_names` Get the node labels for record

`is_connected` Check if nodes are connected.

Methods

Public methods:

- `Kitchen$new()`
- `Kitchen$fct_id()`
- `Kitchen$lvl_id()`
- `Kitchen$fct_names()`
- `Kitchen$lvl_names()`
- `Kitchen$append_fct_nodes()`
- `Kitchen$append_lvl_nodes()`
- `Kitchen$append_fct_edges()`
- `Kitchen$append_lvl_edges()`
- `Kitchen$fct_class()`
- `Kitchen$lvl_class()`
- `Kitchen$fct_child()`
- `Kitchen$lvl_child()`
- `Kitchen$fct_parent()`
- `Kitchen$lvl_parent()`
- `Kitchen$fct_ancestor()`
- `Kitchen$lvl_ancestor()`
- `Kitchen$fct_levels()`
- `Kitchen$setup_data()`
- `Kitchen$add_anatomy()`
- `Kitchen$fct_exists()`
- `Kitchen$trts_exists()`
- `Kitchen$units_exists()`
- `Kitchen$rcrds_exists()`
- `Kitchen$clone()`

Method `new()`**:**

Usage:

```
Kitchen$new(design = NULL)
```

Arguments:

`design` An edibble design.

Method `fct_id()`**:** Get the id based on either the name of the factor node or the class.

Usage:

```
Kitchen$fct_id(name = NULL, class = NULL)
```

Arguments:

`name` The name of the vertex.
`class` The class for the vertex/node.

Method `lvl_id()`**:** Get the id based on name of level node

Usage:

```
Kitchen$lvl_id(name = NULL, class = NULL)
```

Arguments:

`name` The name of the vertex.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$fct_names(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

Method `lvl_names()`**:** Get the level names based on id or class

Usage:

```
Kitchen$lvl_names(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$append_fct_nodes(data)
```

Arguments:

`data` The nodes data

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

Usage:

```
Kitchen$append_lvl_nodes(data)
```

Arguments:

`data` The nodes data

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

Usage:

```
Kitchen$append_fct_edges(data)
```

Arguments:

`data` The nodes data

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

Usage:

```
Kitchen$append_lvl_edges(data)
```

Arguments:

`data` The nodes data

Method `fct_class()`**:** Get the class of the vertex given the factor id

Usage:

```
Kitchen$fct_class(id = NULL)
```

Arguments:

`id` The id of the corresponding node.

Method `lvl_class()`**:** Get the class of the vertex given the level id

Usage:

```
Kitchen$lvl_class(id = NULL)
```

Arguments:

`id` The id of the corresponding node.

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

Usage:

```
Kitchen$fct_child(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$lvl_child(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$fct_parent(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$lvl_parent(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$fct_ancestor(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$lvl_ancestor(id = NULL, class = NULL)
```

Arguments:

`id` The id of the corresponding node.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$fct_levels(id = NULL, name = NULL)
```

Arguments:

`id` The id of the corresponding node.

`name` The name of the vertex.

Method `setup_data()`**:** Setup the node and edge data

Usage:

```
Kitchen$setup_data(fresh, name, class)
```

Arguments:

`fresh` The value of the new graph structure to add.

`name` The name of the vertex.

`class` The class for the vertex/node.

Method `add_anatomy()`**:** Add the anatomy structure

Usage:

```
Kitchen$add_anatomy(fresh, name, class)
```

Arguments:

`fresh` The value of the new graph structure to add.

`name` The name of the vertex.

`class` The class for the vertex/node.

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

Usage:

```
Kitchen$fct_exists(name = NULL, id = NULL, class = NULL, abort = TRUE)
```

Arguments:

`name` The name of the vertex.

`id` The id of the corresponding node.

`class` The class for the vertex/node.

`abort` A logical value to indicate whether to abort if it doesn't exist.

Method `trts_exists()`**:** Check if treatment exists.

Usage:

```
Kitchen$trts_exists(abort = TRUE)
```

Arguments:

`abort` Whether to abort.

Method `units_exists()`**:** Check if unit exists.

Usage:

```
Kitchen$units_exists(abort = TRUE)
```

Arguments:

`abort` Whether to abort.

Method `rcrds_exists()`**:** Check if record exists.

Usage:

```
Kitchen$rcrds_exists(abort = TRUE)
```

Arguments:

`abort` Whether to abort.

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

Usage:

```
Kitchen$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`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(3, c(3, 3, 3))
```

`lvl_attrs` *Setting the traits of the levels*

Description

Use this function to create a "vector" of levels. The vector is actually comprised of a data frame with a column `labels` and other columns with corresponding level attribute (if any). This data frame can be accessed with `lvl_data()`.

Usage

```
lvl_attrs(
  levels = NULL,
  labels = NULL,
  prefix = "",
  suffix = "",
  sep = edibble_labels_opt("sep"),
  include_leading_zero = edibble_labels_opt("leading_zero"),
  data = NULL,
  ...
)
```

Arguments

`levels`	A vector that either denotes the index number or short name of the levels.
`labels`	An optional character vector that is the long name format of `levels`.
`prefix`	The prefix of the labels.
`suffix`	The suffix of the labels.
`sep`	A string to add between `prefix` and `levels`.
`include_leading_zero`	A logical value to indicate whether there should be a leading zero added to level indexes. This is ignored if `levels` is not numeric.
`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

```
lvl_attrs(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 structure for units*

Description

Specify the nesting structure for units

Usage

```
nested_in(
  x,
  ...,
  prefix = "",
  suffix = "",
  leading0 = FALSE,
  sep = edibble_labels_opt("sep"),
  attrs = NULL
)
```

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.
`prefix`	The prefix of the label.
`suffix`	The suffix of the label.
`leading0`	Whether there should be a leading 0 if labels are made.
`sep`	A separator added between prefix and the number if prefix is empty.
`attrs`	A named vector where names and values correspond to attribute names and values of the variable, or a data frame.

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, ...)

edibble(
  .data,
  name = NULL,
  .record = TRUE,
  seed = NULL,
  kitchen = Kitchen,
  ...
)
```

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.
`name`	Optional name used as title for printing the design.
`.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.
`kitchen`	An environment setup in a manner to manipulate, extract and query information on the design.

Value

An edibble table.

`pivot_trts_widelist` *Pivot treatments to a wider list or table format*

Description

Pivot treatments to a wider list or table format

Usage

```
pivot_trts_widelist(.data, trts = NULL, fcts = NULL, drop = FALSE)

pivot_trts_widetable(.data, trts = NULL, fcts = NULL)
```

Arguments

`.data`	An edibble table.
`trts`	A vector of treatment (tidyselect compatible). By default it is NULL and includes all the treatments.
`fcts`	A vector of factors in the edibble table.
`drop`	Whether the resulting list should drop to a vector within each list element if there is only one column. Default is FALSE.

Value

A named list where elements are the data and the names are treatments.

Examples

```
pivot_trts_widelist(takeout(menu_crd(t = 5, n = 20)))
```

`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)))
```

`record_step` *Record the coding step*

Description

Call this function in functions that modify the edibble design or table so the step is tracked. The output of functions using `record_step()` should be returning an edibble design or table.

Usage

```
record_step()
```

Value

Returns nothing.

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

Description

Find the short names of the named designs

Usage

```
scan_menu(pkgs = NULL)
```

Arguments

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

Value

A character vector of the short names of the named menu designs.

Examples

```
scan_menu()
```

`select_units` *Select a subset of units from a cooked design*

Description

Select a subset of units from a cooked design

Usage

```
select_units(prep, ...)
```

Arguments

`prep`	A cooked design.
`...`	The units to select.

Value

An edibble design.

`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, use_labels = FALSE, ..., .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`.
`use_labels`	To show the labels instead of names.
`...`	Either a name-value pair or a series of the names.
`.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, 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_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.

Usage

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

set_rcrds_of(.edibble, ...)
```

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`, `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,
  ...,
  .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`, `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,
  ...,
  .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`, `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()
```

`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`

`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()
```

`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_edibble_var(x)

is_edibble_unit(x)

is_edibble_trt(x)

is_edibble_rcrd(x)
```

Arguments

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

Value

A character vector.

`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`.

Index