Package 'beautils'

June 24, 2026

Type Package

Title Field Planning and Biostatistics Utilities

Version 0.2.0

Maintainer Tiago Olivoto <tiagoolivoto@gmail.com>

Description Provides a collection of utility functions for biostatistics, agricultural trial planning, and experimental design. Key features include generating experimental designs (like Latin Square, Alpha-Lattice by Patterson and Williams (1976) <doi:10.2307/2335087>, and Factorial), fieldbook creation, layout sketching, QR code-based label generation, and descriptive statistical tools to easily handle most common descriptive statistics for quantitative variables as described by Field, A., Miles, J., & Field, Z. (2012, ISBN:978-1-4462-0045-2).

Encoding UTF-8

RoxygenNote 7.3.3

License MIT + file LICENSE

Depends R (>= 4.1.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports cli, dplyr, FielDHub, ggplot2, glue, grid, purrr, qrencoder, rlang, rstudioapi, tidyr, uuid

NeedsCompilation no

Author Tiago Olivoto [aut, cre] (ORCID: <https://orcid.org/0000-0002-0241-9636>)

Repository CRAN

Date/Publication 2026-06-24 09:10:03 UTC

Contents

`alpha_lattice` *Generate Alpha-Lattice Design with Fieldbook and Field Map*

Description

This function generates an Alpha-Lattice design (Generalized Lattice) for experiments with a large number of treatments. It allows for the inclusion of checks, multiple locations, generates stable unique identifiers (UUID), and produces field maps (plots) with visual delimitation of incomplete blocks and repetitions.

Usage

```
alpha_lattice(
  trats,
  k,
  checks = NULL,
  reps = 3,
  seed = 123,
  serpentine = TRUE,
  exp_year = format(Sys.Date(), "%Y"),
  exp_name = "Ressacada",
  namespace = "7ac1a295-ef95-4f7c-8a35-3eed97cb256d",
```

```
  fill_color = TRUE,
  text_size = 3,
  layout = c("default", "custom"),
  layout_allocation = NULL,
  locations = 1,
  plot_start = 1,
  legend = TRUE,
  text_size_row_col = 3,
  label_top = "Coluna",
  label_bottom = "",
  label_left = "Linha",
  label_right = "",
  verbose = TRUE,
  app = FALSE
)
```

Arguments

trats	character. Vector with the names of candidate treatments (e.g., lines).
k	integer. Incomplete block size (number of plots per block).
checks	character or NULL. Vector with check treatments (controls). Total entries (length(trats) + length(checks)) must be a multiple of k.
reps	integer. Number of full repetitions (resolvable blocks). Usually >= 2.
seed	integer. Seed for reproducibility. The actual seed used in each location is seed + location_index.
serpentine	logical. If TRUE, applies serpentine order by row in the field map layout.
exp_year	character. Experiment year (e.g., "2026").
exp_name	character. Experiment or location name. Can be a vector of length equal to locations.
namespace	character. UUID string to ensure UNIQUE_ID generation is reproducible via uuid::UUIDfromName().
fill_color	logical. If TRUE, colors the plots in the map according to the repetition.
text_size	numeric. Font size for treatment names on the map.
layout	character. Defines plot disposition: • "default": Each repetition occupies a single row (columns = total treatments). • "custom": Disposition is defined by layout_allocation.
layout_allocation	numeric(2) or NULL. Vector c(nrow, ncol) defining the number of rows and columns per repetition in the field. Required if layout = "custom".
locations	integer. Number of locations or independent trials to be generated.
plot_start	integer. Starting point for plot numbering. If > 10, numbering will include the location prefix (e.g., 101, 102... for location 1; 201, 202... for location 2).
legend	logical. If TRUE, displays the legend on the map.

`text_size_row_col`
: `numeric`. Font size for the map axes.

`label_top`
: `character`. Map title.

`label_bottom`
: `character`. Map subtitle.

`label_left`
: `character`. Y-axis label.

`label_right`
: `character`. X-axis label.

`verbose`
: `logical`. If `TRUE`, prints progress messages.

`app`
: `logical`. Used internally in the Shiny app.

Details

The Alpha-Lattice design is a resolvable incomplete block design. The function uses the `FielDHub` package engine for randomization but extends its functionality by:

- **Validating the Layout**: Ensures the provided `layout_allocation` is sufficient for the number of treatments.
- **Visual Delimitation**: In the map, thick black lines separate repetitions (resolvable blocks), while dashed red lines delimit incomplete blocks (k).
- **UNIQUE_ID**: Creates a unique ID based on metadata (`YEAR`, `LOCATION`, `PLOT`, etc.), facilitating traceability in databases.

Value

A `list` containing two elements:

`fieldbook`
: A consolidated `tibble` with all locations, containing identification columns, field coordinates, incomplete blocks, and treatments.

`fieldmap`
: A `list` of `ggplot2` objects, one for each location, representing the experiment field map.

Author(s)

Tiago Olivoto <tiago.olivoto@ufsc.br>

References

Patterson, H. D., & Williams, E. R. (1976). A new class of resolvable incomplete block designs. Biometrika, 63(1), 83–92. doi:10.2307/2335087

Examples

```
# Generate an alpha-lattice design
library(beautils)
design <- alpha_lattice(
  trats = paste0("T", 1:6),
  checks = c("CHECK1", "CHECK2"),
  k = 4,
  reps = 3,
```

```
  verbose = FALSE
)
design
```

---

augmented *Augmented Block Design with Fieldbook and Field Map*

Description

Generates an augmented block design (Augmented RCBD) experiment, allowing for multiple locations, serpentine layout, field map (ggplot), and consolidated fieldbook.

Usage

```
augmented(
  lines,
  checks,
  blocks = 4,
  seed = 123,
  serpentine = TRUE,
  exp_year = format(Sys.Date(), "%Y"),
  exp_name = "Ressacada",
  namespace = "7ac1a295-ef95-4f7c-8a35-3eed97cb256d",
  fill_color = TRUE,
  text_size = 3,
  layout = c("default", "custom"),
  layout_allocation = NULL,
  locations = 1,
  plot_start = 1,
  legend = TRUE,
  text_size_row_col = 3,
  label_top = "Coluna",
  label_bottom = "",
  label_left = "Linha",
  label_right = "",
  verbose = TRUE,
  app = FALSE
)
```

Arguments

lines	character. Vector with candidate treatments (lines/new entries).
checks	character. Vector with check treatments (controls).
blocks	integer (>= 1). Number of blocks.
seed	integer. Seed for reproducibility (incremented per location).
serpentine	logical. If TRUE, applies serpentine order by row in the layout.

exp_year	character. Experiment year.
exp_name	character. Experiment name. Can be length 1 or equal to locations.
namespace	character. UUID namespace for generating stable identifiers with uuid::UUIDfromName().
fill_color	logical. If TRUE, colors the map by factor (block row).
text_size	numeric (> 0). Font size displayed on the map.
layout	character. "default" (blocks in sequence) or "custom" (defined by layout_allocation).
layout_allocation	numeric(2) or NULL. When layout = "custom", defines c(nrow, ncol) of the map.
locations	integer (>= 1). Number of locations (independent experiments).
plot_start	integer. Starting point for plot numbering. If 1 (default), plots are numbered from 1 to n. If it is a value >= 10 (e.g., 100), numbering will include the location index (e.g., 101, 102... in location 1; 201, 202... in location 2).
legend	logical. If TRUE, displays the legend on the map.
text_size_row_col	numeric. Font size for the map axes.
label_top	character. Map title.
label_bottom	character. Map subtitle.
label_left	character. Y-axis label.
label_right	character. X-axis label.
verbose	logical. If TRUE, prints progress messages.
app	logical. Used internally in the Shiny app.

Details

The function is a wrapper around FielDHub::RCBD_augmented(), adding:

- Unique identifiers (UNIQUE_ID) per plot;
- Optional serpentine layout;
- Customized layout (nrow x ncol) when layout = "custom";
- Field map in ggplot with configurable colors and labels;
- Consolidated fieldbook for all locations. Main rules and validations:
- blocks must be a positive integer;
- exp_name must have length 1 or equal to locations;
- plot_start starting point for numbering (e.g., 100);
- For layout = "custom", nrow * ncol must be >= number of plots per block.

Value

A list containing two elements:

fieldbook	A consolidated tibble with all locations, containing identification columns, field coordinates, blocks, and treatments.
fieldmap	A list of ggplot2 objects, one for each location, representing the experiment field map.

Examples

```
# Generate an augmented block design
library(beautils)
design <- augmented(
  lines = paste0("L", 1:4),
  checks = paste0("C", 1:2),
  blocks = 2,
  verbose = FALSE
)
```

`create_label` *Create plot labels in PDF*

Description

Create plot labels in PDF

Usage

```
create_label(
  width = 6,
  height = 2.5,
  page_size = NULL,
  page_width = 21,
  page_height = 29.7,
  top_mar = 1,
  bot_mar = 1,
  left_mar = 1,
  right_mar = 1,
  numrow = 10L,
  numcol = 3L,
  filename = file.path(tempdir(), "PlotLabel"),
  font_sz = 12,
  Treetag = FALSE,
  family = "sans",
  rounded = TRUE,
  print_across = TRUE,
  rect = TRUE,
  top_left_1 = NULL,
  top_left_2 = NULL,
  top_right_1 = NULL,
  top_right_2 = NULL,
  center_right_1 = NULL,
  center_right_2 = NULL,
  center_right_3 = NULL,
  bottom_left_1 = NULL,
  bottom_left_2 = NULL,
```

```
  unique_id = NULL,
  include_qr = TRUE,
  ec_level = 3,
  qr_size = NULL,
  lwd = 0.5,
  arc_size = 0.1,
  font_sz_tl1 = 12,
  font_sz_tl2 = 10,
  font_sz_tr1 = 12,
  font_sz_tr2 = 10,
  font_sz_cr1 = 8,
  font_sz_cr2 = 8,
  font_sz_cr3 = 8,
  font_sz_bl1 = 4,
  font_sz_bl2 = 8,
  ...
)
```

Arguments

`width`	Label width in centimeters.
`height`	Label height in centimeters.
`page_size`	Predefined sheet size. Can be `"A0"`, `"A1"`, `"A2"`, `"A3"`, `"A4"`, `"A5"`, `"Letter"`, `"Legal"`, `"Tabloid"` or `NULL` (default). When provided, it overrides `page_width` and `page_height`.
`page_width`	Page width in centimeters (ignored if `page_size` is provided).
`page_height`	Page height in centimeters (ignored if `page_size` is provided).
`top_mar`, `bot_mar`, `left_mar`, `right_mar`	Page margins in centimeters.
`numrow`	Number of label rows per page.
`numcol`	Number of label columns per page.
`filename`	Base name for the output PDF file.
`font_sz`	Font size.
`Treetag`	Logical; if `TRUE`, applies a tree label layout.
`family`	Font family.
`rounded`	Logical; if `TRUE`, uses rounded borders for labels.
`print_across`	Logical; if `TRUE`, fills labels row by row; if `FALSE`, column by column.
`rect`	Logical; if `TRUE`, draws a border around each label.
`top_left_1`, `top_left_2`, `top_right_1`, `top_right_2`	Text vectors for top fields.
`center_right_1`, `center_right_2`, `center_right_3`	Text vectors for central fields.
`bottom_left_1`, `bottom_left_2`	Text vectors for bottom fields.

`unique_id` Vector of unique IDs for QR code generation (mandatory when `include_qr = TRUE`).

`include_qr` Logical; if `TRUE`, includes a QR code on each label.

`ec_level` QR code error correction level (1-3).

`qr_size` QR code size in centimeters (square). If NULL, uses the default behavior (proportional to label width).

`lwd` Label border line width.

`arc_size` Arc size (rounding) when `rounded = TRUE`.

`font_sz_tl1`, `font_sz_tl2`
Font sizes for top-left fields.

`font_sz_tr1`, `font_sz_tr2`
Font sizes for top-right fields.

`font_sz_cr1`, `font_sz_cr2`, `font_sz_cr3`
Font sizes for central fields.

`font_sz_bl1`, `font_sz_bl2`
Font sizes for bottom fields.

`...` Additional arguments.

Value

No return value, called for side effects (writing labels to a PDF file).

Examples

```
# Creating a simple label manually
create_label(width = 6, height = 2.5,
             top_left_1 = "Trial 2026",
             top_right_1 = "P: 101",
             center_right_1 = "Treatment A",
             include_qr = FALSE,
             filename = file.path(tempdir(), "manual_label"))

# Creating labels with QR Code
create_label(width = 6, height = 2.5,
             top_left_1 = "Trial 2026",
             top_right_1 = "P: 102",
             center_right_1 = "G215",
             unique_id = "uuid-1234-5678",
             include_qr = TRUE,
             filename = file.path(tempdir(), "qr_label"))
```

desc_stat *Descriptive statistics*

Description

- `desc_stat()` Computes the most used measures of central tendency, position, and dispersion.
- `desc_wider()` is useful to put the variables in columns and grouping variables in rows. The table is filled with a statistic chosen with the argument `stat`.

Usage

```
desc_stat(
  .data = NULL,
  ...,
  by = NULL,
  stats = "main",
  hist = FALSE,
  level = 0.95,
  digits = 4,
  na.rm = FALSE,
  verbose = TRUE,
  plot_theme = ggplot2::theme_minimal()
)

desc_wider(.data, which)
```

Arguments

`.data`	The data to be analyzed. It can be a data frame (possible with grouped data passed from `dplyr::group_by()` or a numeric vector. For `desc_wider()` `.data` is an object of class `desc_stat`.
`...`	A single variable name or a comma-separated list of unquoted variables names. If no variable is informed, all the numeric variables from `.data` will be used. Select helpers are allowed.
`by`	One variable (factor) to compute the function by. It is a shortcut to `dplyr::group_by()`. To compute the statistics by more than one grouping variable use that function.
`stats`	The descriptive statistics to show. This is used to filter the output after computation. Defaults to `"main"` (cv, max, mean median, min, sd.amo, se, ci). Other allowed values are `"all"` to show all the statistics, `"robust"` to show robust statistics, `"quantile"` to show quantile statistics, or chose one (or more) of the following: • `"av.dev"`: average deviation. • `"ci.t"`: t-interval (95% confidence interval) of the mean. • `"ci.z"`: z-interval (95% confidence interval) of the mean. • `"cv"`: coefficient of variation.

- "iqr": interquartile range.
- "gmean": geometric mean.
- "hmean": harmonic mean.
- "Kurt": kurtosis.
- "mad": median absolute deviation.
- "max": maximum value.
- "mean": arithmetic mean.
- "median": median.
- "min": minimum value.
- "n": the length of the data.
- "n.valid": The valid (Not NA) number of elements
- "n.missing": The number of missing values
- "n.unique": The length of unique elements.
- "ps": the pseudo-sigma (iqr / 1.35).
- "q2.5", "q25", "q75", "q97.5": the percentile 2.5\ quartile, third quartile, and percentile 97.5\
- range: The range of data).
- "sd.amo", "sd.pop": the sample and population standard deviation.
- "se": the standard error of the mean.
- "skew": skewness.
- "sum". the sum of the values.
- "sum.dev": the sum of the absolute deviations.
- "ave.sq.dev": the average of the squared deviations.
- "sum.sq.dev": the sum of the squared deviations.
- "n.valid": The size of sample with valid number (not NA).
- "var.amo", "var.pop": the sample and population variance.

Use a names to select the statistics. For example, stats = c("median, mean, cv, n"). Note that the statistic names **are not** case-sensitive. Both comma or space can be used as separator.

hist	Logical argument defaults to FALSE. If hist = TRUE then a histogram is created for each selected variable.
level	The confidence level to compute the confidence interval of mean. Defaults to 0.95.
digits	The number of significant digits.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
verbose	Logical argument. If verbose = FALSE the code is run silently.
plot_theme	The graphical theme of the plot.
which	A statistic to fill the table.

Value

- desc_stats() returns a tibble with the statistics in the columns and variables (with possible grouping factors) in rows.
- desc_wider() returns a tibble with variables in columns and grouping factors in rows.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Field, A., Miles, J., & Field, Z. (2012). Discovering Statistics Using R. SAGE Publications Ltd. (ISBN: 978-1-4462-0045-2)

Examples

```
library(beautils)
#=============================================================#
# Example 1: main statistics (coefficient of variation, maximum,#
# mean, median, minimum, sample standard deviation, standard    #
# error and confidence interval of the mean) for all numeric    #
# variables in data                                             #
#=============================================================#

desc_stat(df_eucalipto())
#=============================================================#
# Example 5: Compute all statistics for all numeric variables   #
# by row.
#=============================================================#

df_eucalipto() |>
  group_by(fila) |>
  desc_stat(circunferencia, stats = c("min, max, n, mean, sd.amo, cv"))
```

`df_eucalipto` *Example of Eucalyptus Plant Data*

Description

The data contains measurements of diameter at breast height (DBH) using both a caliper (dap_suta) and a tape measure (dap_trena), as well as the circumference of eucalyptus plants. A total of 70 eucalyptus plants were evaluated, distributed across 5 rows with 14 plants per row.

Usage

```
df_eucalipto()
```

Value

A data frame

`df_linhaca` *Example of Flax Plant Data*

Description

The data contains evaluations of flax plants sown at four different times (E1 to E4). The experiment involved sampling between 51 and 66 plants per sowing time. Various plant traits were analyzed, including:

Usage

```
df_linhaca()
```

Details

- ac: capsule height
- ap: plant height
- nc: number of capsules
- ng: number of grains
- areac: capsule area
- compg: grain length
- largg: grain width
- nr: number of branches
- mc: capsule mass
- rgpla: grain yield per plant
- icc: capsule harvest index
- ngcap: number of grains per capsule
- mmg: thousand-grain mass

Value

A data frame

df_normal *Example of Normal data*

Description

The data contains the length of 205.043 flax grains that follows a normal distribution

Usage

```
df_normal()
```

Value

A data frame

df_quali *Example of Qualitative data*

Description

The data contains the area (AF) and dry matter (MST) of chicory plants evaluated at different levels of solar radiation (50%, 70%, and 100%). The experiment was conducted in a randomized complete block design with four replications.

Usage

```
df_quali()
```

Value

A data frame

df_quali_quali_ci *Example of Qualitative x Qualitative data (with significant interaction)*

Description

The data contains two qualitative factors: sulfur (2 levels) and nitrogen splitting (3 levels), with significant interaction.

Usage

```
df_quali_quali_ci()
```

Value

A data frame

`df_quali_quali_ci2` *Example of Qualitative x Qualitative data (no significant interaction)*

Description

The data contains two qualitative factors: evaluation days (3 levels) and solar radiation (3 levels), with no significant interaction.

Usage

```
df_quali_quali_ci2()
```

Value

A data frame

`df_quali_quali_si` *Example of Qualitative x Qualitative data (no significant interaction)*

Description

The data contains two qualitative factors: evaluation days (3 levels) and solar radiation (3 levels), with no significant interaction.

Usage

```
df_quali_quali_si()
```

Value

A data frame

df_quali_quanti_ci *Example of Qualitative x Quantitative data (with significant interaction)*

Description

The data contains two factors: hybrids (2 levels, qualitative) and nitrogen doses (5 levels, quantitative), with significant interaction.

Usage

```
df_quali_quanti_ci()
```

Value

A data frame

df_quali_quanti_si *Example of Qualitative x Quantitative data (no significant interaction)*

Description

The data contains two factors: hybrids (2 levels, qualitative) and nitrogen doses (5 levels, quantitative), with no significant interaction.

Usage

```
df_quali_quanti_si()
```

Value

A data frame

df_quanti_1 *Example of Quantitative data (linear trend)*

Description

The data contains maize yield evaluated at different nitrogen levels, showing a linear trend.

Usage

```
df_quanti_1()
```

Value

A data frame

df_quanti_2 *Example of Quantitative data (quadratic trend)*

Description

The data contains oat yield evaluated at different nitrogen levels, showing a quadratic trend.

Usage

```
df_quanti_2()
```

Value

A data frame

full_factorial *Full Factorial Design with Fieldbook and Plot Map*

Description

Generates a full factorial experiment (CRD or RCBD) from a combination of factors and levels, with options for additional treatments, serpentine layout, reproducible unique identifiers, and returns a consolidated fieldbook and field maps (ggplot) per location.

Usage

```
full_factorial(
  factors,
  nlevels,
  levels,
  add_trats = NULL,
  reps = 4,
  design = c("CRD", "RCBD"),
  seed = 123,
  serpentine = TRUE,
  exp_year = format(Sys.Date(), "%Y"),
  exp_name = "Ressacada",
  fill_color = TRUE,
  text_size = 3,
  layout = c("default", "custom"),
  layout_allocation = NULL,
  namespace = "7ac1a295-ef95-4f7c-8a35-3eed97cb256d",
  locations = 1,
  plot_start = 1,
  legend = TRUE,
```

```
  text_size_row_col = 3,
  label_top = "Coluna",
  label_bottom = "",
  label_left = "Linha",
  label_right = "",
  verbose = TRUE,
  app = FALSE
)
```

Arguments

factors	character. Vector with factor names (e.g., c("A", "B")).
nlevels	integer. Number of levels for each factor, in the same order as factors (e.g., c(2, 3)).
levels	character. Vector with labels for all levels, in the sequence of factors repeated by their respective levels (length must match sum(nlevels)).
add_trats	character or NULL. Names of additional treatments (e.g., checks) to be included besides the factorial combinations.
reps	integer (>= 1). Number of repetitions (blocks for RCBD; rows/stacks for CRD).
design	character. Experimental design: "CRD" or "RCBD".
seed	integer. Seed for reproducibility; it is incremental per location (seed + location_index).
serpentine	logical. If TRUE, applies "serpentine" order by row in the final layout.
exp_year	character. Experiment year (displayed in the fieldbook).
exp_name	character. Experiment name. Can be length 1 (recycled for all locations) or length equal to locations.
fill_color	logical. If TRUE, the map colors by TRT_COMB; otherwise, uses gray fill.
text_size	numeric (> 0). Font size for plot labels in the map.
layout	character. "default" (reps x ntrats matrix), "custom" (defined by layout_allocation).
layout_allocation	numeric(2) or NULL. When layout = "custom", vector c(nrow, ncol) with the number of rows and columns of the layout per repetition. Must accommodate ntrats_total = prod(nlevels) + length(add_trats).
namespace	character. UUID namespace for generating stable IDs via uuid::UUIDfromName().
locations	integer (>= 1). Number of locations (independent experiments) to be generated.
plot_start	integer. Starting point for plot numbering. If 1 (default), plots are numbered from 1 to n. If it is a value >= 10 (e.g., 100), numbering will include the location index (e.g., 101, 102... in location 1; 201, 202... in location 2).
legend	logical. If TRUE, displays the legend on the map.
text_size_row_col	numeric. Font size for the map axes.
label_top	character. Map title (top).

label_bottom	character. Map subtitle (bottom).
label_left	character. Y-axis label (left).
label_right	character. Y-axis label (right).
verbose	logical. If TRUE, prints progress messages.
app	logical. Used internally in the Shiny app.

Details

The function is a convenience wrapper around `FielDHub::full_factorial()`, adding:

- Optional inclusion of extra treatments (`add_trats`) with appropriate resampling per design;
- Generation of a layout (default or customized) with or without serpentine;
- Creation of unique and reproducible identifiers per plot (`UNIQUE_ID`) considering `ROW`, `COL`, `YEAR`, `LOCATION`, `PLOT`, `REP`, and `TRT_COMB`;
- Production of a consolidated *fieldbook* (all locations) and field maps (`ggplot`) per location.

Main rules and validations:

- `FACTORS <- rep(factors, nlevels)` must have matching `length(levels)`.
- For `layout = "custom"`, `nrow * ncol` must be `>= ntrats_total`.
- In `design = "RCBD"`, allocation is done by block (`REP`).

Value

A `list` with two components:

- `fieldbook`: `data.frame/tibble` with identification columns (`UNIQUE_ID`, `YEAR`, `LOCATION`, `ROW`, `COL`, `PLOT`, `REP`, `TRT_COMB`, `FACTOR_*`, etc.).
- `fieldmap`: `list` of `ggplot` objects (one per location) representing the field map.

Examples

```
# Generate a full factorial design
library(beautils)
design <- full_factorial(
  factors = c("A", "B"),
  nlevels = c(2, 2),
  levels = c("a1", "a2", "b1", "b2"),
  reps = 3,
  design = "RCBD",
  verbose = FALSE
)
```

`labelize` *Generate plot labels from a fieldbook*

Description

High-level wrapper around `create_label()`. This command processes automatic mappings between fieldbook columns and label fields.

Usage

```
labelize(dat, ...)
```

Arguments

`dat`	`data.frame`. Fieldbook.
`...`	Parameters passed to `create_label()`. Accepts mappings in the format `field = "Prefix: = COLUMN"`. Supported fields: `top_left_1`, `top_left_2`, `top_right_1`, `top_right_2`, `center_right_1`, `center_right_2`, `center_right_3`, `bottom_left_1`, `bottom_left_2`.

Value

No return value, called for side effects (generating and saving PDF labels).

Examples

```
# Example usage with mapping
library(beautils)
df <- unifatorial(trats = paste0("T", 1:4))
labelize(df$fieldbook,
         top_left_1 = "TRAT: = TREATMENT",
         top_right_1 = "Rep: = REP",
         center_right_1 = "Plot: = PLOT",
         center_right_2 = "Row: = ROW",
         center_right_3 = "Col: = COL",
         bottom_left_1 =  "Location: = LOCATION",
         filename = file.path(tempdir(), "labels"))
```

latin_square *Latin Square Design with Fieldbook and Field Map*

Description

Generates a Latin Square experimental design with one or more replicate squares, using `FielDHub::latin_square()` as the randomisation engine. Returns a consolidated fieldbook and a ggplot field map where each square is stacked vertically, with thick lines separating squares and tiles coloured by treatment.

Usage

```
latin_square(
  trats = NULL,
  reps = 1,
  seed = 123,
  exp_year = format(Sys.Date(), "%Y"),
  exp_name = "Ressacada",
  namespace = "7ac1a295-ef95-4f7c-8a35-3eed97cb256d",
  fill_color = TRUE,
  text_size = 3,
  locations = 1,
  plot_start = 1,
  legend = TRUE,
  text_size_row_col = 3,
  label_top = "Coluna",
  label_bottom = "",
  label_left = "Linha",
  label_right = "",
  planter = c("serpentine", "cartesian"),
  data = NULL,
  verbose = TRUE,
  app = FALSE
)
```

Arguments

trats	`character` or `NULL`. Vector of treatment labels. If `NULL` and `data` is provided, labels are taken from the data frame.
reps	`integer`. Number of full Latin Squares (replicates). Default 1.
seed	`integer`. Reproducibility seed (incremented per location).
exp_year	`character`. Experiment year.
exp_name	`character`. Experiment name (recycled or length == `locations`).
namespace	`character`. UUID namespace for `UNIQUE_ID` generation.
fill_color	`logical`. Colour tiles by treatment?
text_size	`numeric`. Font size on the map.

locations	integer. Number of independent locations.
plot_start	integer. Starting plot number.
legend	logical. Show legend on map?
text_size_row_col	numeric. Axis text size.
label_top	character. Top axis label.
label_bottom	character. Bottom axis label.
label_left	character. Left axis label.
label_right	character. Right axis label.
planter	character. "serpentine" or "cartesian".
data	data.frame or NULL. Optional data frame with columns ROW, COLUMN, TREATMENT passed to FielDHub::latin_square().
verbose	logical. Print progress messages?
app	logical. Internal Shiny flag.

Value

A list with:

- fieldbook: consolidated data.frame with UNIQUE_ID, YEAR, LOCATION, ROW, COL, SQUARE, PLOT, TREATMENT.
- fieldmap: list of ggplot objects, one per location.

Author(s)

Tiago Olivoto <tiago.olivoto@ufsc.br>

Examples

```
# Generate a Latin Square design
library(beautils)
design <- latin_square(
  trats = paste0("T", 1:4),
  reps = 2,
  verbose = FALSE
)
```

make_qrcode *Generate a QR code graphic object*

Description

Converts an identifier into a QR code matrix and returns a `grid::rasterGrob` object. This version is optimized to avoid dependency on heavy packages like `raster`.

Usage

```
make_qrcode(my_id, ec_level = 3)
```

Arguments

`my_id`	The text or ID to be encoded.
`ec_level`	Error correction level (0-3). Default is 3.

Value

A `grid::rasterGrob` object.

split_plot *Split-Plot Design with Fieldbook and Plot Map*

Description

Generates a split-plot experiment in CRD or RCBD designs, allowing for multiple locations, serpentine layout, and export of a consolidated fieldbook and field maps (ggplot).

Usage

```
split_plot(
  wholeplot,
  subplot,
  reps = 4,
  design = c("CRD", "RCBD"),
  seed = 123,
  serpentine = TRUE,
  exp_year = format(Sys.Date(), "%Y"),
  exp_name = "Ressacada",
  namespace = "7ac1a295-ef95-4f7c-8a35-3eed97cb256d",
  fill_color = TRUE,
  text_size = 3,
  locations = 1,
  plot_start = 1,
```

```
  legend = TRUE,
  text_size_row_col = 3,
  label_top = "Coluna",
  label_bottom = "",
  label_left = "Linha",
  label_right = "",
  verbose = TRUE,
  app = FALSE
)
```

Arguments

wholeplot	character. Vector with levels of the whole-plot factor.
subplot	character. Vector with levels of the sub-plot factor.
reps	integer (>= 1). Number of repetitions (blocks for RCBD).
design	character. Experimental design: "CRD" or "RCBD".
seed	integer. Seed for reproducibility (incremented per location).
serpentine	logical. If TRUE, applies serpentine order by row in the layout.
exp_year	character. Experiment year (displayed in the fieldbook).
exp_name	character. Experiment name. Can be length 1 or equal to locations.
namespace	character. UUID namespace for generating stable identifiers with uuid::UUIDfromName().
fill_color	logical. If TRUE, colors the field map by the whole-plot factor (WHOLE_PLOT).
text_size	numeric (> 0). Font size on the field map.
locations	integer (>= 1). Number of locations (independent experiments).
plot_start	integer. Starting point for numbering (see full_factorial).
legend	logical. If TRUE, displays the legend on the map.
text_size_row_col	numeric. Font size for the map axes.
label_top	character. Map title.
label_bottom	character. Map subtitle.
label_left	character. Y-axis label.
label_right	character. X-axis label.
verbose	logical. If TRUE, prints progress messages.
app	logical. Used internally in the Shiny app.

Details

The function is a convenience wrapper around FielDHub::split_plot(), with:

- Automatic design definition from design;
- Allocation of whole plots (WHOLE_PLOT) and sub-plots (SUB_PLOT);
- Creation of unique and reproducible identifiers (UNIQUE_ID) considering YEAR, LOCATION, ROW, COL, PLOT, REP, and TRT_COMB;

- Generation of maps (ggplot) with WHOLE_PLOT and SUB_PLOT labels;
- Optional serpentine layout by row.

Main rules:

- reps must be a positive integer;
- exp_name must have length 1 or be equal to locations;
- plot_start starting point for numbering (e.g., 100);
- For design = "RCBD", allocation considers blocks (REP).

Value

A list with:

- fieldbook: data.frame/tibble with experiment information, including WHOLE_PLOT, SUB_PLOT, TRT_COMB, identifiers, and coordinates (ROW, COL);
- fieldmap: list of ggplot objects (one per location) with the field map.

Examples

```
# Generate a split-plot design
library(beautils)
design <- split_plot(
  wholeplot = c("W1", "W2"),
  subplot = c("S1", "S2", "S3"),
  reps = 2,
  design = "RCBD",
  verbose = FALSE
)
design
```

---

strip_plot *Strip Plot (Experiments in Strips) with Fieldbook and Field Map*

Description

Generates a strip plot (experiments in strips) design with multiple locations, serpentine layout, field map (ggplot) with visually distinct horizontal and vertical strips, and a consolidated fieldbook. This is a wrapper around FielDHub::strip_plot().

Usage

```
strip_plot(
  hstrips,
  vstrips,
  reps = 3,
  seed = 123,
  serpentine = TRUE,
```

```
  exp_year = format(Sys.Date(), "%Y"),
  exp_name = "Ressacada",
  namespace = "7ac1a295-ef95-4f7c-8a35-3eed97cb256d",
  fill_color = TRUE,
  text_size = 3,
  locations = 1,
  plot_start = 1,
  legend = TRUE,
  text_size_row_col = 3,
  label_top = "Coluna",
  label_bottom = "",
  label_left = "Linha",
  label_right = "",
  planter = c("serpentine", "cartesian"),
  randomizeH = TRUE,
  randomizeV = FALSE,
  verbose = TRUE,
  app = FALSE
)
```

Arguments

hstrips	character. Vector with the labels/levels of the horizontal strip factor.
vstrips	character. Vector with the labels/levels of the vertical strip factor.
reps	integer (>= 1). Number of blocks (full replicates).
seed	integer. Seed for reproducibility (incremented per location).
serpentine	logical. If TRUE, applies serpentine ordering within blocks.
exp_year	character. Experiment year (used in fieldbook columns).
exp_name	character. Experiment/location name(s). Recycled or must be of length equal to locations.
namespace	character. UUID namespace for reproducible UNIQUE_ID generation via uuid::UUIDfromName().
fill_color	logical. If TRUE, tiles are colored by H_STRIP (horizontal strip factor), with V_STRIP shown by label alone.
text_size	numeric (> 0). Font size for labels on the field map.
locations	integer (>= 1). Number of locations (independent experiments).
plot_start	integer. Starting point for plot numbering. If 1 (default), plots are numbered from 1 to n. If >= 10 (e.g., 100), numbering includes the location index.
legend	logical. If TRUE, displays the legend on the map.
text_size_row_col	numeric. Font size for the map axes.
label_top	character. Map top axis label.
label_bottom	character. Map bottom axis label.
label_left	character. Map left axis label.
label_right	character. Map right axis label.

planter	character. Either "serpentine" or "cartesian" (passed to FielDHub::strip_plot()).
randomizeH	logical. Randomize horizontal strips within each replicate?
randomizeV	logical. Randomize vertical strips within each replicate?
verbose	logical. If TRUE, prints progress messages.
app	logical. Used internally in the Shiny app.

Details

The strip plot places each horizontal factor level as a full horizontal row (spanning all vertical strips) and each vertical factor level as a full vertical column (spanning all horizontal strips) within each replicate block. The field map highlights this structure: tiles are filled by H_STRIP, thick black lines separate replicate blocks, and vertical dashed lines separate the V_STRIP columns.

Value

A list with two elements:

- fieldbook: data.frame/tibble with identifiers, H_STRIP, V_STRIP, coordinates (ROW, COL), PLOT, REP, LOCATION, YEAR, UNIQUE_ID.
- fieldmap: list of ggplot objects (one per location).

Author(s)

Tiago Olivoto <tiago.olivoto@ufsc.br>

Examples

```
# Generate a strip-plot design
library(beautils)
design <- strip_plot(
  hstrips = c("H1", "H2", "H3"),
  vstrips = c("V1", "V2"),
  reps = 2,
  verbose = FALSE
)
design
```

---

unifatorial *Single Factor Design with Fieldbook and Plot Map*

Description

Generates a single factor experiment (CRD or RCBD) with multiple locations, optional serpentine layout, plot map (ggplot), and consolidated fieldbook.

Usage

```
unifatorial(
  trats,
  reps = 4,
  design = c("CRD", "RCBD"),
  seed = 123,
  serpentine = TRUE,
  exp_name = "Ressacada",
  exp_year = format(Sys.Date(), "%Y"),
  fill_color = TRUE,
  text_size = 3,
  layout = c("default", "custom"),
  namespace = "7ac1a295-ef95-4f7c-8a35-3eed97cb256d",
  layout_allocation = NULL,
  locations = 1,
  plot_start = 1,
  legend = TRUE,
  text_size_row_col = 3,
  label_top = "Coluna",
  label_bottom = "",
  label_left = "Linha",
  label_right = "",
  verbose = TRUE,
  app = FALSE
)
```

Arguments

`trats`	`character`. Vector with the treatments to be tested.
`reps`	`integer` (>= 1). Number of repetitions.
`design`	`character`. Experimental design: `"CRD"` or `"RCBD"`.
`seed`	`integer`. Seed for reproducibility (incremented per location).
`serpentine`	`logical`. If `TRUE`, applies serpentine order by row in the layout.
`exp_name`	`character`. Experiment name. Can be length 1 or equal to `locations`.
`exp_year`	`character`. Experiment year (used in fieldbook columns).
`fill_color`	`logical`. If `TRUE`, colors the map by treatments.
`text_size`	`numeric` (> 0). Font size displayed on the map.
`layout`	`character`. `"default"` (`reps` x `trats` matrix) or `"custom"` (defined by `layout_allocation`).
`namespace`	`character`. UUID namespace for generating stable identifiers with `uuid::UUIDfromName()`.
`layout_allocation`	`numeric(2)` or `NULL`. When `layout = "custom"`, defines `c(nrow, ncol)` of the map.
`locations`	`integer` (>= 1). Number of locations (independent experiments).

plot_start	integer. Starting point for plot numbering. If 1 (default), plots are numbered from 1 to n. If it is a value >= 10 (e.g., 100), numbering will include the location index (e.g., 101, 102... in location 1; 201, 202... in location 2).
legend	logical. If TRUE, displays the legend on the map.
text_size_row_col	numeric. Font size for the map axes.
label_top	character. Map title.
label_bottom	character. Map subtitle.
label_left	character. Y-axis label.
label_right	character. X-axis label.
verbose	logical. If TRUE, prints progress messages.
app	logical. Used internally in the Shiny app.

Details

The function is a wrapper around `FielDHub::CRD()` and `FielDHub::RCBD()`, adding:

- Unique identifiers (`UNIQUE_ID`) per plot;
- Optional serpentine layout;
- Customized layout (`nrow x ncol`) when `layout = "custom"`;
- Plot map in `ggplot` with configurable colors and labels;
- Consolidated fieldbook for all locations.

Main rules and validations:

- `reps` must be a positive integer;
- `exp_name` must have length 1 or be equal to `locations`;
- `plot_start` starting point for numbering (e.g., 100);
- For `layout = "custom"`, `nrow * ncol` must be >= number of treatments.

Value

A `list` with:

- `fieldbook`: `data.frame/tibble` with identifiers, treatments, and coordinates (`ROW`, `COL`, etc.);
- `fieldmap`: list of `ggplot` objects (one per location) representing the field map.

Examples

```
# Generate a single factor design
library(beautils)
design <- unifatorial(
  trats = paste0("T", 1:4),
  reps = 3,
  design = "RCBD",
  verbose = FALSE
)
design
```

utils_samples *Random Sampling*

Description

- sample_random() performs Simple Random Sampling or Stratified Random Sampling
- sample_systematic() performs systematic sampling. In this case, a regular interval of size k (k = floor(N/n)) is generated considering the population size (N) and desired sample size (n). Then, the starting member (r) is randomly chosen between 1-k. The second element is r + k, and so on.

Usage

```
sample_random(data, n, prop, by = NULL, weight = NULL)

sample_systematic(data, n, r = NULL, by = NULL)
```

Arguments

data	A data frame. If data is a grouped_df, the operation will be performed on each group (stratified).
n, prop	Provide either n, the number of rows, or prop, the proportion of rows to select. If neither are supplied, n = 1 will be used.
by	A categorical variable to compute the sample by. It is a shortcut to dplyr::group_by() that allows to group the data by one categorical variable. If more than one grouping variable needs to be used, use dplyr::group_by() to pass the data grouped.
weight	Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
r	The starting element. By default, r is randomly selected between 1:k

Value

An object of the same type as data.

Examples

```
library(beautils)
sample_random(df_eucalipto(), n = 5)
sample_random(df_eucalipto(),
              n = 3,
              by = fila)

sample_systematic(df_eucalipto(), n = 6)
```

utils_sets *Utilities for set operations for many sets*

Description

Provides alternative function to `base::union()`, `base::intersect()`, and `base::setdiff()`.

- `set_union()`: Returns the union of the sets in `...`.
- `set_intersect()`: Returns the intersect of the sets in `...`.
- `set_difference()`: Returns the difference of the sets in `...`.

Usage

```
set_intersect(..., pairs = FALSE)

set_union(..., pairs = FALSE)

set_difference(..., pairs = FALSE)
```

Arguments

`...`	A list or a comma-separated list of vectors in the same class. If vector contains duplicates they will be discarded. If the list doesn't have names the sets will be named as `"set_1"`, `"Set_2"`, `"Set_3"` and so on. If vectors are given in `...`, the set names will be named with the names of the objects provided. Data frames are also allowed, provided that common column names exist.
`pairs`	Returns the pairwise unions of the sets? Defaults to `FALSE`.

Value

A vector showing the desired operation of the sets. If `pairs = TRUE`, returns a list showing the pairwise operation of the sets.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(beautils)
(A <- letters[1:4])
(B <- letters[2:5])
(C <- letters[3:7])

set_union(A, B)
set_intersect(A, B, C)
set_difference(B, C)
```

```
# Operations with data frames
# Add a row id for better understanding
sets <- df_eucalipto()

set_1 <- sets[1:5,]
set_2 <- sets[2:6,]
set_3 <- sets[3:7,]

set_intersect(set_1, set_2)
set_difference(set_2, set_3)
set_union(set_1, set_2, set_3)
```

`utils_stats` *Useful functions for computing descriptive statistics*

Description

- **The following functions compute descriptive statistics by levels of a factor or combination of factors quickly.**
  - `cv_by()` For computing coefficient of variation.
  - `max_by()` For computing maximum values.
  - `mean_by()` For computing arithmetic means.
  - `min_by()` For compuing minimum values.
  - `n_by()` For getting the length.
  - `sd_by()` For computing sample standard deviation.
  - `var_by()` For computing sample variance.
  - `sem_by()` For computing standard error of the mean.
- **Useful functions for descriptive statistics. All of them work naturally with `\%>\%`, handle grouped data and multiple variables (all numeric variables from `.data` by default).**
  - `av_dev()` computes the average absolute deviation.
  - `ci_mean_t()` computes the t-interval for the mean.
  - `ci_mean_z()` computes the z-interval for the mean.
  - `cv()` computes the coefficient of variation.
  - `freq_table()` Computes a frequency table for either numeric and categorical/discrete data. For numeric data, it is possible to define the number of classes to be generated.
  - `hmean()`, `gmean()` computes the harmonic and geometric means, respectively. The harmonic mean is the reciprocal of the arithmetic mean of the reciprocals. The geometric mean is the *n*th root of *n* products.
  - `kurt()` computes the kurtosis like used in SAS and SPSS.
  - `range_data()` Computes the range of the values.
  - `n_valid()` The valid (not `NA`) length of a data.
  - `n_unique()` Number of unique values.

- `n_missing()` Number of missing values.
- `row_col_mean(), row_col_sum()` Adds a row with the mean/sum of each variable and a column with the the mean/sum for each row of the data.
- `sd_amo(), sd_pop()` Computes sample and populational standard deviation, respectively.
- `sem()` computes the standard error of the mean.
- `skew()` computes the skewness like used in SAS and SPSS.
- `ave_dev()` computes the average of the absolute deviations.
- `sum_dev()` computes the sum of the absolute deviations.
- `sum_sq()` computes the sum of the squared values.
- `sum_sq_dev()` computes the sum of the squared deviations.
- `var_amo(), var_pop()` computes sample and populational variance.

`desc_stat()` is wrapper function around the above ones and can be used to compute quickly all these statistics at once.

Usage

```
cv_by(.data, ..., .vars = NULL, na.rm = FALSE)

max_by(.data, ..., .vars = NULL, na.rm = FALSE)

min_by(.data, ..., .vars = NULL, na.rm = FALSE)

mean_by(.data, ..., .vars = NULL, na.rm = FALSE)

mean_by(.data, ..., .vars = NULL, na.rm = FALSE)

n_by(.data, ..., .vars = NULL, na.rm = FALSE)

sd_by(.data, ..., .vars = NULL, na.rm = FALSE)

var_by(.data, ..., .vars = NULL, na.rm = FALSE)

sem_by(.data, ..., .vars = NULL, na.rm = FALSE)

sum_by(.data, ..., .vars = NULL, na.rm = FALSE)

av_dev(.data, ..., na.rm = FALSE)

ci_mean_t(.data, ..., na.rm = FALSE, level = 0.95)

ci_mean_z(.data, ..., na.rm = FALSE, level = 0.95)

cv(.data, ..., na.rm = FALSE)

hmean(.data, ..., na.rm = FALSE)
```

```
gmean(.data, ..., na.rm = FALSE)

kurt(.data, ..., na.rm = FALSE)

n_missing(.data, ..., na.rm = FALSE)

n_unique(.data, ..., na.rm = FALSE)

n_valid(.data, ..., na.rm = FALSE)

pseudo_sigma(.data, ..., na.rm = FALSE)

range_data(.data, ..., na.rm = FALSE)

row_col_mean(.data, na.rm = FALSE)

row_col_sum(.data, na.rm = FALSE)

sd_amo(.data, ..., na.rm = FALSE)

sd_pop(.data, ..., na.rm = FALSE)

sem(.data, ..., na.rm = FALSE)

skew(.data, ..., na.rm = FALSE)

sum_dev(.data, ..., na.rm = FALSE)

ave_dev(.data, ..., na.rm = FALSE)

sum_sq_dev(.data, ..., na.rm = FALSE)

sum_sq(.data, ..., na.rm = FALSE)

var_pop(.data, ..., na.rm = FALSE)

var_amo(.data, ..., na.rm = FALSE)

freq_table(.data, var, k = NULL, digits = 3)

freq_hist(
  table,
  xlab = NULL,
  ylab = NULL,
  fill = "gray",
  color = "black",
  ygrid = TRUE
)
```

Arguments

.data	A data frame or a numeric vector.
...	The argument depends on the function used. • For *_by functions, ... is one or more categorical variables for grouping the data. Then the statistic required will be computed for all numeric variables in the data. If no variables are informed in ..., the statistic will be computed ignoring all non-numeric variables in .data. • For the other statistics, ... is a comma-separated of unquoted variable names to compute the statistics. If no variables are informed in n ..., the statistic will be computed for all numeric variables in .data.
.vars	Used to select variables in the *_by() functions. One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables. Defaults to NULL (all numeric variables are analyzed)..
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
level	The confidence level for the confidence interval of the mean. Defaults to 0.95.
var	The variable to compute the frequency table. See Details for more details.
k	The number of classes to be created. See Details for more details.
digits	The number of significant figures to show. Defaults to 2.
table	A frequency table computed with freq_table().
xlab, ylab	The x and y labels.
fill, color	The color to fill the bars and color the border of the bar, respectively.
ygrid	Shows a grid line on the y axis? Defaults to TRUE. freq_hist <- function(table,

Details

The function freq_table() computes a frequency table for either numerical or categorical variables. If a variable is categorical or discrete (integer values), the number of classes will be the number of levels that the variable contains.

If a variable (say, data) is continuous, the number of classes (k) is given by the square root of the number of samples (n) if n =< 100 or 5 * log10(n) if n > 100.

The amplitude (A) of the data is used to define the size of the class (c), given by

c = A / (n - 1)

The lower limit of the first class (LL1) is given by min(data) - c / 2. The upper limit is given by LL1 + c. The limits of the other classes are given in the same way. After the creation of the classes, the absolute and relative frequencies within each class are computed.

Value

- Functions *_by() returns a tbl_df with the computed statistics by each level of the factor(s) declared in ....

- All other functions return a named integer if the input is a data frame or a numeric value if the input is a numeric vector.
- `freq_table()` Returns a list with the frequency table and the breaks used for class definition. These breaks can be used to construct an histogram of the variable.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Ferreira, Daniel Furtado. 2009. Estatistica Basica. 2 ed. Vicosa, MG: UFLA.

Field, A., Miles, J., & Field, Z. (2012). Discovering Statistics Using R. SAGE Publications Ltd. (ISBN: 978-1-4462-0045-2)

Examples

```
library(beautils)
# means of all numeric variables by fila
mean_by(df_eucalipto(), fila, .vars = circunferencia)

# Coefficient of variation for all numeric variables
# by fila
cv_by(df_eucalipto(), fila, .vars = circunferencia)

# Skewness of a numeric vector
set.seed(1)
nvec <- rnorm(500, 10, 1)
skew(nvec)

# Confidence interval 0.95 for the mean
# All numeric variables
# Grouped by levels of fila
df_eucalipto() |>
  group_by(fila) |>
  ci_mean_t(circunferencia)


# Frequency table for variable circunferencia
df_eucalipto() |>
  freq_table(circunferencia)
```

`utils_wd` *Set and get the Working Directory quicky*

Description

- `get_wd_here()` gets the working directory to the path of the current script.
- `set_wd_here()` sets the working directory to the path of the current script.
- `open_wd_here()` Open the File Explorer at the directory path of the current script.
- `open_wd()` Open the File Explorer at the current working directory.

Usage

```
set_wd_here(path = NULL)

get_wd_here(path = NULL)

open_wd_here(path = get_wd_here())

open_wd(path = getwd())
```

Arguments

`path`	Path components below the project root. Defaults to `NULL`. This means that the directory will be set to the path of the file. If the path doesn't exist, the user will be asked if he wants to create such a folder.

Value

- `get_wd_here()` returns a full-path directory name.
- `get_wd_here()` returns a message showing the current working directory.
- `open_wd_here()` Opens the File Explorer of the path returned by `get_wd_here()`.

Examples

```
if (interactive() && requireNamespace("rstudioapi", quietly = TRUE)) {
  get_wd_here()
  set_wd_here()
  open_wd_here()
}
```

Index