Package ‘databook’

February 17, 2025

Title A set of the `data_book` functions used in R-Instat

Version 0.1.3

Description This package provides tools for managing and manipulating data frames. It includes functions for renaming columns, setting hidden and protected columns, applying filters, and managing row and column selections. The package also supports adding metadata, creating custom objects, and generating graphs, making it a versatile tool for data analysis and visualisation.

License LGPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports chillR,
circular,
clipr,
data.table,
dplyr,
e1071,
ggplot2,
gridExtra,
Hmisc,
hydroGOF,
lazyeval,
lubridate,
imputeTS,
janitor,
magrittr,
patchwork,
plyr,
purrr,
R6,
reshape2,
robustbase,
sjlabelled,
sjmisc,
stringr,
tibble,
tidyselect,
verification,
Weighted.Desc.Stat,

weights,
zoo

Remotes IDEMSInternational/instatExtras

Contents

`BIAS` *Calculate Bias*

Description

Computes the bias using the `verification::verify` function.

Usage

```
BIAS(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The bias.

`calc_from_convert` *Convert calculation list to a specific format*

Description

Convert calculation list to a specific format

Usage

```
calc_from_convert(x)
```

Arguments

`x`	A list of calculations.

Value

A formatted list of calculations.

---

check_filter *Check and update filter object parameters*

Description

Check and update filter object parameters

Usage

```
check_filter(filter_obj)
```

Arguments

`filter_obj`	A filter object to check and update.

Value

The updated filter object.

---

count_calc *Count Matching Elements*

Description

Counts the number of elements in a dataset that satisfy a specified condition.

Usage

```
count_calc(x, count_test = "==", count_value, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`count_test`	Character. The comparison operator (e.g., "==", ">=").
`count_value`	Numeric. The value to compare against.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The count of matching elements.

cp *Calculate Coefficient of Persistence*

Description

Computes the coefficient of persistence using the `hydroGOF::cp` function.

Usage

```
cp(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The coefficient of persistence.

d *Calculate Index of Agreement*

Description

Computes the index of agreement using the `hydroGOF::d` function.

Usage

```
d(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The index of agreement.

`DataBook` *DataBook Class*

Description

An R6 class to manage a collection of data tables along with their metadata and other associated properties.

Usage

```
DataBook$new(data_tables = list(), instat_obj_metadata = list(),
             data_tables_variables_metadata = rep(list(data.frame()), length(data_tables)),
                 data_tables_metadata = rep(list(list()), length(data_tables)),
                  data_tables_filters = rep(list(list()), length(data_tables)),
             data_tables_column_selections = rep(list(list()), length(data_tables)),
                   imported_from = as.list(rep("", length(data_tables))),
                   messages = TRUE, convert = TRUE, create = TRUE)
```

Format

An R6 class object.

Methods

`summary(data_name, columns_to_summarise, summaries, factors = c(), store_results = FALSE, drop = FALS`
Perform and Return Summaries for a Data Object

`summary_table(data_name, columns_to_summarise = NULL, summaries, factors = c(), store_table = FALSE, s`
Generate a Summary Table

`set_data(new_data, messages, check_names)` Sets the data for the DataSheet object.

`standardise_country_names(data_name, country_columns = c())` Standardizes country names in the specified data table.

`define_as_climatic(data_name, types, key_col_names, key_name)` Defines a data table as climatic data.

`define_corruption_outputs(data_name, output_columns = c())` Defines corruption output columns in the specified data table.

`define_red_flags(data_name, red_flags = c())` Defines red flag columns in the specified data table.

`define_as_procurement(data_name, primary_types = c(), calculated_types = c(), country_data_name, cou`
Defines a data table as procurement data.

`define_as_procurement_country_level_data(data_name, contract_level_data_name, types = c(), auto_ge`
Defines a data table as procurement country-level data.

`get_CRI_component_column_names(data_name)` Gets the names of CRI component columns in the specified data table.

`get_red_flag_column_names(data_name)` Gets the names of red flag columns in the specified data table.

`get_CRI_column_names(data_name)` Gets the names of CRI columns in the specified data table.

`get_corruption_column_name(data_name, type)` Gets the name of the corruption column in the specified data table.

`import_data(data_tables = list(), data_tables_variables_metadata = rep(list(data.frame()),length(d`
Imports data into the DataBook from a list of data tables and their metadata.

`replace_instat_object(new_instat_object)` Replaces the current instat object with a new one.

`set_data_objects(new_data_objects)` Sets the data objects for the DataBook.

`copy_data_object(data_name, new_name, filter_name = "", column_selection_name = "", reset_row_names =`
Copies a data object with optional filtering and column selection.

`import_RDS(data_RDS, keep_existing = TRUE, overwrite_existing = FALSE, include_objects = TRUE, incluc`
Imports data from an RDS file into the DataBook.

`clone_data_object(curr_data_object, include_objects = TRUE, include_metadata = TRUE, include_logs =`
Clones a data object with options to include various components.

`clone_instat_calculation(curr_instat_calculation, ...)` Clones an instat calculation.

`import_from_ODK(username, form_name, platform)` Imports data from ODK (Open Data Kit).

`set_meta(new_meta)` Sets the metadata for the DataBook.

`set_objects(new_objects)` Sets the objects for the DataBook.

`set_scalars(new_scalars)` Sets scalar values in the DataBook.

`set_undo_history(new_undo_history)` Set the undo history for the DataBook.

`get_scalars(data_name)` Retrieve scalars for a specific data object or overall DataBook.

`set_scalar_names(data_name, as_list = FALSE, excluded_items = c(), ...)` Retrieve scalar names for a specific data table.

`get_scalar_value(data_name, scalar_name)` Retrieve the value of a specific scalar for a given data object.

`add_scalar(data_name, scalar_name = "", scalar_value)` Add a scalar to a specific data object or the overall DataBook.

`set_enable_disable_undo(data_name, disable_undo)` Enable or disable undo functionality for a specific data object.

`is_undo(data_name)` Check if undo functionality is enabled for a specific data object.

`has_undo_history(data_name)` Check if there is undo history for a specific data object.

`undo_last_action(data_name)` Undo the last action for a specific data object.

`redo_last_action(data_name)` Redo the last undone action for a specific data object.

`get_column_climatic_type(data_name, col_name, attr_name)` Retrieve the climatic type attribute for a specific column.

`append_data_object(name, obj, add_to_graph_book = TRUE)` Appends a data object to the DataBook.

`get_data_objects(data_name, as_list = FALSE, ...)` Gets data objects from the DataBook.

`get_data_frame(data_name, convert_to_character = FALSE, stack_data = FALSE, include_hidden_columns =`
Gets a data frame from the DataBook with various options.

`get_variables_metadata(data_name, data_type = "all", convert_to_character = FALSE, property, column,`
Gets the variables metadata for the specified data table.

`get_column_data_types(data_name, columns)` Gets the data types of the specified columns in the data table.

`get_column_labels(data_name, columns)` Gets the labels of the specified columns in the data table.

`get_data_frame_label(data_name, use_current_filter = FALSE)` Gets the label of the data frame.

`get_data_frame_metadata(data_name, label, include_calculated = TRUE, excluded_not_for_display = TRU`
Gets the metadata of the data frame.

`get_combined_metadata(convert_to_character = FALSE)` Gets combined metadata from all data tables.

`get_metadata(name, ...)` Gets metadata for the specified name.

`get_data_names(as_list = FALSE, include, exclude, excluded_items, include_hidden = TRUE, ...)`
Gets the names of the data tables in the DataBook.

`get_data_changed(data_name)` Checks if the data has changed.

`get_variables_metadata_changed(data_name)` Checks if the variables metadata has changed.

`get_metadata_changed(data_name)` Checks if the metadata has changed.

`get_calculations(data_name)` Gets the calculations for the specified data table.

`get_calculation_names(data_name, as_list = FALSE, excluded_items = c())` Gets the names of the calculations for the specified data table.

`dataframe_count()` Gets the count of data frames in the DataBook.

`set_data_frames_changed(data_name = "", new_val)` Sets the changed status for data frames.

`set_variables_metadata_changed(data_name = "", new_val)` Sets the changed status for variables metadata.

`set_metadata_changed(data_name = "", new_val)` Sets the changed status for metadata.

`add_columns_to_data(data_name, col_name = "", col_data, use_col_name_as_prefix = FALSE, hidden = FALS`
Adds columns to the specified data table.

`get_columns_from_data(data_name, col_names, from_stacked_data = FALSE, force_as_data_frame = FALSE,`
Gets columns from the specified data table.

`create_graph_data_book()` Creates a graph data book.

`add_object(data_name = NULL, object_name = NULL, object_type_label, object_format, object)`
Adds an object to the DataBook.

`get_object_names(data_name = NULL, object_type_label = NULL, as_list = FALSE, ...)` Gets the names of the objects in the DataBook.

`get_objects(data_name = NULL, object_type_label = NULL)` Gets the objects from the DataBook.

`get_object(data_name = NULL, object_name)` Gets a specific object from the DataBook.

`get_object_data(data_name = NULL, object_name, as_file = FALSE)` Gets the data of a specific object from the DataBook.

`get_objects_data(data_name = NULL, object_names = NULL, as_files = FALSE)` Gets the data of multiple objects from the DataBook.

`get_last_object_data(object_type_label, as_file = TRUE)` Gets the data of the last object of a specified type from the DataBook.

`rename_object(data_name, object_name, new_name, object_type = "object")` Renames an object in the DataBook.

`delete_objects(data_name, object_names, object_type = "object")` Deletes objects from the DataBook.

`reorder_objects(data_name, new_order)` Reorders the objects in the DataBook.

`get_from_object(data_name, object_name, value1, value2, value3)` Gets values from a specified object in the DataBook.

`add_filter(data_name, filter, filter_name = "", replace = TRUE, set_as_current_filter = FALSE, na.rm =`
Adds a filter to the specified data table.

`add_filter_as_levels(data_name, filter_levels, column)` Adds filter levels to the specified column.

`current_filter(data_name)` Gets the current filter for the specified data table.

`set_current_filter(data_name, filter_name = "")` Sets the current filter for the specified data table.

`get_filter(data_name, filter_name)` Gets a filter by name from the specified data table.

`get_filter_as_logical(data_name, filter_name)` Gets a filter as a logical vector from the specified data table.

`get_current_filter(data_name)` Gets the current filter for the specified data table.

`get_filter_row_names(data_name, filter_name)` Gets the row names that match a specified filter in the data table.

`get_current_filter_name(data_name)` Gets the name of the current filter for the specified data table.

`get_filter_names(data_name, as_list = FALSE, include = list(), exclude = list(), excluded_items = c())`
Gets the names of the filters in the specified data table.

`remove_current_filter(data_name)` Removes the current filter from the specified data table.

`filter_applied(data_name)` Checks if a filter is applied to the specified data table.

`filter_string(data_name, filter_name)` Gets the filter string for a specified filter in the data table.

`get_filter_as_instat_calculation(data_name, filter_name)` Gets a filter as an instat calculation from the specified data table.

`add_column_selection(data_name, column_selection, name = "", replace = TRUE, set_as_current = FALSE,`
Adds a column selection to the specified data table.

`current_column_selection(data_name)` Gets the current column selection for the specified data table.

`set_current_column_selection(data_name, name = "")` Sets the current column selection for the specified data table.

`get_column_selection(data_name, name)` Gets a column selection by name from the specified data table.

`get_column_selection_column_names(data_name, filter_name)` Gets the column names for a specified filter in the data table.

`get_column_selected_column_names(data_name, column_selection_name = "")` Gets the names of the selected columns in the specified data table.

`get_current_column_selection(data_name)` Gets the current column selection for the specified data table.

`get_current_column_selection_name(data_name)` Gets the name of the current column selection for the specified data table.

`get_column_selection_names(data_name, as_list = FALSE, include = list(), exclude = list(), excluded_i`
Gets the names of the column selections in the specified data table.

`remove_current_column_selection(data_name)` Removes the current column selection from the specified data table.

`column_selection_applied(data_name)` Checks if a column selection is applied to the specified data table.

`replace_value_in_data(data_name, col_names, rows, old_value, old_is_missing = FALSE, start_value = N`
  Replaces values in the specified columns and rows of the data table.

`paste_from_clipboard(data_name, col_names, start_row_pos = 1, first_clip_row_is_header = TRUE, clip_`
  Pastes data from the clipboard into the specified columns of the data table.

`rename_column_in_data(data_name, column_name = NULL, new_val = NULL, label = "", type = "single", .fn,`
  Renames a column in the specified data table.

`frequency_tables(data_name, x_col_names, y_col_name, n_column_factors = 1, store_results = TRUE, dro`
  Creates frequency tables for the specified data table.

`anova_tables(data_name, x_col_names, y_col_name, signif.stars = FALSE, sign_level = FALSE, means = FA`
  Creates ANOVA tables for the specified data table.

`cor(data_name, x_col_names, y_col_name, use = "everything", method = c("pearson", "kendall", "spearma`
  Calculates correlations for the specified columns in the data table.

`remove_columns_in_data(data_name, cols, allow_delete_all = FALSE)` Removes columns from the specified data table.

`remove_rows_in_data(data_name, row_names)` Removes rows from the specified data table.

`get_next_default_column_name(data_name, prefix)` Gets the next default column name for the specified data table.

`get_column_names(data_name, as_list = FALSE, include = list(), exclude = list(), excluded_items = c(),`
  Gets the column names in the specified data table.

`reorder_columns_in_data(data_name, col_order)` Reorders the columns in the specified data table.

`insert_row_in_data(data_name, start_row, row_data = c(), number_rows = 1, before = FALSE)`
  Inserts rows into the specified data table.

`get_data_frame_length(data_name, use_current_filter = FALSE)` Gets the length of the data frame in the specified data table.

`get_next_default_dataframe_name(prefix, include_index = TRUE, start_index = 1)` Gets the next default name for a data frame in the DataBook.

`delete_dataframes(data_names, delete_graph_book = TRUE)` Deletes data frames from the DataBook.

`remove_link(link_name)` Removes a link from the DataBook.

`get_column_factor_levels(data_name, col_name = "")` Gets the factor levels of a column in the specified data table.

`get_factor_data_frame(data_name, col_name = "", include_levels = TRUE, include_NA_level = FALSE)`
  Gets a factor data frame for the specified column in the data table.

`sort_dataframe(data_name, col_names = c(), decreasing = FALSE, na.last = TRUE, by_row_names = FALSE, r`
  Sorts the specified data table.

`rename_dataframe(data_name, new_value = "", label = "")` Renames the specified data table.

`convert_column_to_type(data_name, col_names = c(), to_type, factor_values = NULL, set_digits, set_de`
  Converts the specified columns to a different type in the data table.

`append_to_variables_metadata(data_name, col_names, property, new_val = "")` Appends a new value to the specified property in the variables metadata for the given columns in the specified data table.

`append_to_dataframe_metadata(data_name, property, new_val = "")` Appends a new value to the specified property in the dataframe metadata for the specified data table.

`append_to_metadata(property, new_val = "", allow_override_special = FALSE)` Appends a new value to the specified property in the overall metadata, with an option to override special properties.

`add_metadata_field(data_name, property, new_val = "")` Adds a new metadata field to the specified data table or to the overall metadata.

`reorder_dataframes(data_frames_order)` Reorders the data frames in the object based on the provided order.

`copy_columns(data_name, col_names = "", copy_to_clipboard = FALSE)` Copies the specified columns from the given data table, with an option to copy to the clipboard.

`drop_unused_factor_levels(data_name, col_name)` Drops unused factor levels from the specified column in the given data table.

`set_factor_levels(data_name, col_name, new_labels, new_levels, set_new_labels = TRUE)` Sets new factor levels and labels for the specified column in the given data table.

`edit_factor_level(data_name, col_name, old_level, new_level)` Edits an existing factor level in the specified column of the given data table.

`set_factor_reference_level(data_name, col_name, new_ref_level)` Sets a new reference level for the specified factor column in the given data table.

`get_column_count(data_name, use_column_selection = FALSE)` Returns the count of columns in the specified data table, with an option to use the current column selection.

`reorder_factor_levels(data_name, col_name, new_level_names)` Reorders the factor levels in the specified column of the given data table.

`get_data_type(data_name, col_name)` Returns the data type of the specified column in the given data table.

`copy_data_frame(data_name, new_name, label = "", copy_to_clipboard = FALSE)` Copies the specified data table to a new data table with an optional new name, label, and option to copy to the clipboard.

`copy_col_metadata_to_clipboard(data_name, property_names)` Copies the specified column metadata properties from the given data table to the clipboard.

`copy_data_frame_metadata_to_clipboard(data_name, property_names)` Copies the specified data frame metadata properties from the given data table to the clipboard.

`copy_to_clipboard(content)` Copies the given content to the clipboard.

`set_hidden_columns(data_name, col_names = c())` Sets the specified columns in the given data table to be hidden.

`unhide_all_columns(data_name)` Unhides all columns in the specified data table.

`set_hidden_data_frames(data_names = c())` Sets the specified data frames to be hidden.

`get_hidden_data_frames()` Returns the names of all hidden data frames.

`set_row_names(data_name, row_names)` Sets new row names for the specified data table.

`get_row_names(data_name)` Returns the row names of the specified data table.

`set_protected_columns(data_name, col_names)` Sets the specified columns in the given data table to be protected.

`get_metadata_fields(data_name, include_overall, as_list = FALSE, include, exclude, excluded_items =` Returns the metadata fields for the specified data table and overall metadata, with options to include or exclude specific fields.

`freeze_columns(data_name, column)` Freezes the specified columns in the given data table.

`unfreeze_columns(data_name)` Unfreezes all columns in the specified data table.

`is_variables_metadata(data_name, property, column, return_vector = FALSE)` Checks if the specified property is part of the variables metadata for the given column in the specified data table.

`data_frame_exists(data_name)` Checks if the specified data table exists in the object.

`add_key(data_name, col_names, key_name)` Adds a key to the specified data table using the given columns and key name.

`is_key(data_name, col_names)` Checks if the specified columns form a key in the given data table.

`has_key(data_name)` Checks if the specified data table has a key.

`get_keys(data_name, key_name)` Returns the keys for the specified data table and key name.

`add_new_comment(data_name, row = "", column = "", comment)` Adds a new comment to the specified data table, optionally specifying the row and column.

`get_comments(data_name, comment_id)` Returns the comments for the specified data table and comment ID.

`get_links(link_name, ...)` Returns the links for the specified link name or all links if no name is provided.

`set_structure_columns(data_name, struc_type_1 = c(), struc_type_2 = c(), struc_type_3 = c())` Sets the structure columns for the specified data table.

`add_dependent_columns(data_name, columns, dependent_cols)` Adds dependent columns to the specified columns in the given data table.

`set_column_colours(data_name, columns, colours)` Sets the colors for the specified columns in the given data table.

`has_colours(data_name, columns)` Checks if the specified columns in the given data table have colors.

`remove_column_colours(data_name)` Removes colors from all columns in the specified data table.

`set_column_colours_by_metadata(data_name, columns, property)` Sets the colors for the specified columns in the given data table based on the specified metadata property.

`graph_one_variable(data_name, columns, numeric = "geom_boxplot", categorical = "geom_bar", character` Creates a graph for one variable in the specified data table with options for the type of graph, axis scaling, and other parameters.

`make_date_yeardoy(data_name, year, doy, base, doy_typical_length = "366")` Creates a date column from the specified year and day of year columns in the given data table.

`make_date_yearmonthday(data_name, year, month, day, f_year, f_month, f_day, year_format, month_forma` Creates a date column from the specified year, month, and day columns in the given data table, with options for formatting.

`set_contrasts_of_factor(data_name, col_name, new_contrasts, defined_contr_matrix)` Sets the contrasts for the specified factor column in the given data table.

`create_factor_data_frame(data_name, factor, factor_data_frame_name, include_contrasts = FALSE, repl` Creates a new data frame for the specified factor column in the given data table, with options to include contrasts and summary counts.

`split_date(data_name, col_name = "", year_val = FALSE, year_name = FALSE, leap_year = FALSE, month_val` Splits the specified date column into multiple components such as year, month, day, etc. in the given data table.

`import_SST(dataset, data_from = 5, data_names = c())` Imports SST data from the specified dataset and data source, creating data tables with the specified names.

`make_inventory_plot(data_name, date_col, station_col = NULL, year_col = NULL, doy_col = NULL, element_`
  Creates an inventory plot for the specified data table with various customisation options.

`import_NetCDF(nc, path, name, only_data_vars = TRUE, keep_raw_time = TRUE, include_metadata = TRUE, bo`
  Imports data from a NetCDF file, with options to specify the data variables, time format, metadata inclusion, and boundaries.

`infill_missing_dates(data_name, date_name, factors, start_month, start_date, end_date, resort = TRUE`
  Infills missing dates in the specified data table using the provided date column and factors.

`get_key_names(data_name, include_overall = TRUE, include, exclude, include_empty = FALSE, as_list = FA`
  Returns the key names for the specified data table, with options to include overall keys, exclude specific keys, and return as a list.

`remove_key(data_name, key_name)` Removes the specified key from the given data table.

`add_climdex_indices(data_name, climdex_output, freq = "annual", station, year, month)`
  Adds climdex indices to the specified data table, with options for frequency, station, year, and month.

`is_metadata(data_name, str)` Checks if the specified string is part of the metadata for the given data table.

`get_climatic_column_name(data_name, col_name)` Returns the climatic column name for the specified column in the given data table.

`merge_data(data_name, new_data, by = NULL, type = "left", match = "all")` Merges new data into the specified data table using the provided columns and merge type.

`get_corruption_data_names()` Returns the names of all data tables with corruption data.

`get_corruption_contract_data_names()` Returns the names of all data tables with corruption contract data.

`get_database_variable_names(query, data_name, include_overall = TRUE, include, exclude, include_empt`
  Returns the database variable names for the specified query and data table, with options to include overall variables, exclude specific variables, and return as a list.

`get_nc_variable_names(file = "", as_list = FALSE, ...)` Returns the variable names from the specified NetCDF file, with an option to return as a list.

`has_database_connection()` Checks if there is a database connection.

`database_connect(dbname, user, host, port, drv = RMySQL::MySQL())` Connects to a database using the provided credentials and driver.

`get_database_connection()` Returns the current database connection.

`set_database_connection(dbi_connection)` Sets the database connection to the specified DBI connection object.

`database_disconnect()` Disconnects from the current database.

`import_from_climsoft(stationfiltercolumn = "stationId", stations = c(), elementfiltercolumn = "eleme`
  Imports data from CLIMSOFT using the specified filters and options for observation data, flags, and unstacking.

`import_from_iri(download_from, data_file, data_frame_name, location_data_name, path, X1, X2 = NA, Y1`
  Imports data from IRI using the specified parameters for download, file path, coordinates, and area type.

`export_workspace(data_names, file, include_graphs = TRUE, include_models = TRUE, include_metadata = T`
  Exports the workspace to a file, including the specified data tables, graphs, models, and metadata.

`set_links(new_links)` Sets the links in the object to the specified new links.

`display_daily_graph(data_name, date_col = NULL, station_col = NULL, year_col = NULL, doy_col = NULL, c`
: Displays a daily graph for the specified data table with options for columns, element, colors, and limits.

`create_variable_set(data_name, set_name, columns)` Creates a variable set with the specified name and columns in the given data table.

`update_variable_set(data_name, set_name, columns, new_set_name)` Updates the specified variable set with new columns and optionally a new name in the given data table.

`delete_variable_sets(data_name, set_names)` Deletes the specified variable sets from the given data table.

`get_variable_sets_names(data_name, include_overall = TRUE, include, exclude, include_empty = FALSE, a`
: Returns the names of variable sets for the specified data table, with options to include overall sets, exclude specific sets, and return as a list.

`get_variable_sets(data_name, set_names, force_as_list = FALSE)` Returns the specified variable sets from the given data table, with an option to force the result as a list.

`crops_definitions(data_name, year, station, rain, day, rain_totals, plant_days, plant_lengths, start`
: Defines crop parameters for the specified data table using the provided columns and options for seasons, days, and properties.

`tidy_climatic_data(x, format, stack_cols, day, month, year, stack_years, station, element, element_n`
: Converts wide-format daily climatic data to long format using the specified columns and options for format, elements, and validation.

`get_geometry(data)` Returns the geometry column for the specified data table.

`package_check(package)` Checks if the specified package is installed and returns information about its version and availability.

`download_from_IRI(source, data, path = tempdir(), min_lon, max_lon, min_lat, max_lat, min_date, max_c`
: Downloads data from IRI using the specified source, data, coordinates, date range, and options for download type and import.

`patch_climate_element(data_name, date_col_name = "", var = "", vars = c(), max_mean_bias = NA, max_std`
: Patches the specified climate element in the given data table using the provided columns and options for bias, time interval, and station.

`visualize_element_na(data_name, element_col_name, element_col_name_imputed, station_col_name, x_ax`
: Visualizes missing data for the specified element in the given data table using the provided columns and options for labels, legend, orientation, and measure.

`get_data_entry_data(data_name, station, date, elements, view_variables, station_name, type, start_d`
: Returns data entry data for the specified data table using the provided columns and options for date range, variables, and type.

`save_data_entry_data(data_name, new_data, rows_changed, comments_list = list(), add_flags = FALSE, .`
: Saves data entry data to the specified data table with options for adding comments, flags, and rows changed.

`import_from_cds(user, dataset, elements, start_date, end_date, lon, lat, path, import = FALSE, new_nam`
: Imports data from CDS using the specified user, dataset, elements, date range, coordinates, and options for file path and import.

`add_flag_fields(data_name, col_names, key_column_names)` Adds flag fields to the specified columns in the given data table, using the provided key columns.

`remove_empty(data_name, which = c("rows","cols"))` Removes empty rows or columns from the specified data table.

`replace_values_with_NA(data_name, row_index, column_index)` Replaces values with NA in the specified rows and columns of the given data table.

`has_labels(data_name, col_names)` Checks if the specified columns in the given data table have labels.

`wrap_or_unwrap_data(data_name, col_name, column_data, width, wrap = TRUE)` Wraps or unwraps the specified column data in the given data table to the specified width.

`anova_tables2(data_name, x_col_names, y_col_name, total = TRUE, signif.stars = FALSE, sign_level = FA`
Generate ANOVA tables for specified columns in a dataset.

`define_as_options_by_context(data_name, obyc_types = NULL, key_columns = NULL)` Define options by context for a specified dataset.

`display_daily_table(data_name, climatic_element, date_col, year_col, station_col, Misscode, Tracec`
Display a daily summary table for a specified climatic data element.

`add_comment(new_comment)` Adds a new `instat_comment` object to the data sheet if the key is defined and valid.

`delete_comment(comment_id)` Deletes a comment from the data sheet based on the comment ID.

`get_comment_ids()` Retrieves all comment IDs currently stored in the data sheet.

`get_comments_as_data_frame()` Converts all comments in the data sheet to a data frame format for easier inspection and analysis.

`update_links_rename_data_frame(old_data_name, new_data_name)` This function updates all links that reference a data frame with a specified old name, renaming it to a new name.

`update_links_rename_column(data_name, old_column_name, new_column_name)` This function updates all links referencing a column in a data frame with a specified old column name, renaming it to a new column name.

`add_link(from_data_frame, to_data_frame, link_pairs, type, link_name)` This function adds a new link between two data frames with the specified link pairs and type. It will check if the link already exists or if the link columns are keys.

`get_link_names(data_name, include_overall = TRUE, include, exclude, include_empty = FALSE, as_list =`
Retrieves the names of all links involving a specified data frame, with options to include or exclude specific types.

`link_exists_from(curr_data_frame, link_pairs)` Verifies if a link exists from a specific data frame with given link pairs.

`link_exists_between(from_data_frame, to_data_frame, ordered = FALSE)` This function checks if there is an ordered or unordered link between two specified data frames.

`get_link_between(from_data_frame, to_data_frame, ordered = FALSE)` Retrieves the link definition between two specified data frames.

`link_exists_from_by_to(first_data_frame, link_pairs, second_data_frame)` This function checks if a link exists from `first_data_frame` to `second_data_frame` using the specified `link_pairs` columns.

`get_linked_to_data_name(from_data_frame, link_cols = c(), include_self = FALSE)` This function returns the names of data frames linked to `from_data_frame`. Optionally, includes `from_data_frame` itself in the output if `include_self` is TRUE. Filters results by `link_cols`, if provided.

`get_linked_to_definition(from_data_frame, link_pairs)` This function returns a list of the target data frame and matched columns.

`get_possible_linked_to_definition(from_data_frame, link_pairs)` This function attempts to find a linked data frame that matches `link_pairs`. Recursively explores links between multiple data frames.

`get_equivalent_columns(from_data_name, columns, to_data_name)` This function returns columns in `to_data_name` equivalent to `columns` in `from_data_name`. Recursively searches links between multiple data frames.

`link_between_containing(from_data_frame, containing_columns, to_data_frame)` This function returns columns in `to_data_frame` corresponding to `containing_columns` in `from_data_frame` if a link exists between them.

`view_link(link_name)` Displays the details of a specified link.

`apply_calculation(calc)` Apply a Calculation to Data in the DataBook

`save_calculation(end_data_frame, calc)` Save a Calculation to a Data Frame

`apply_instat_calculation(calc, curr_data_list, previous_manipulations = list(), param_list = list()`
Apply an Instat Calculation

`run_instat_calculation(calc, display = TRUE, param_list = list())` Run an Instat Calculation and Display Results

`get_corresponding_link_columns(first_data_frame_name, first_data_frame_columns, second_data_frame`
Get Corresponding Link Columns

`get_link_columns_from_data_frames(first_data_frame_name, first_data_frame_columns, second_data_fr`
Get Link Columns Between Data Frames

`save_calc_output(calc, curr_data_list, previous_manipulations)` Save the Output of a Calculation

`convert_linked_variable(from_data_frame, link_cols)` Convert Linked Variable to Matching Class

`remove_unused_station_year_combinations(data_name, year, station)` Remove Unused Station-Year Combinations

`append_summaries_to_data_object(out, data_name, columns_to_summarise, summaries, factors = c(), summ`
Append Summaries to a Data Object

`calculate_summary(data_name, columns_to_summarise = NULL, summaries, factors = c(), store_results = T`
Calculate Summaries for a Data Object

@section Active bindings:

`data_objects_changed` Logical indicating whether the data objects have changed.

@export

Active bindings

`data_objects_changed` Logical indicating whether the data objects have changed.

Methods

Public methods:

- `DataBook$new()`
- `DataBook$standardise_country_names()`
- `DataBook$define_as_climatic()`
- `DataBook$define_corruption_outputs()`
- `DataBook$define_red_flags()`
- `DataBook$define_as_procurement()`
- `DataBook$define_as_procurement_country_level_data()`
- `DataBook$get_CRI_component_column_names()`

- DataBook$get_red_flag_column_names()
- DataBook$get_CRI_column_names()
- DataBook$get_corruption_column_name()
- DataBook$import_data()
- DataBook$replace_instat_object()
- DataBook$set_data_objects()
- DataBook$copy_data_object()
- DataBook$import_RDS()
- DataBook$clone_data_object()
- DataBook$clone_instat_calculation()
- DataBook$import_from_ODK()
- DataBook$set_meta()
- DataBook$set_objects()
- DataBook$set_undo_history()
- DataBook$set_scalars()
- DataBook$get_scalars()
- DataBook$get_scalar_names()
- DataBook$get_scalar_value()
- DataBook$add_scalar()
- DataBook$set_enable_disable_undo()
- DataBook$is_undo()
- DataBook$has_undo_history()
- DataBook$undo_last_action()
- DataBook$redo_last_action()
- DataBook$get_column_climatic_type()
- DataBook$append_data_object()
- DataBook$get_data_objects()
- DataBook$get_data_frame()
- DataBook$get_variables_metadata()
- DataBook$get_column_data_types()
- DataBook$get_column_labels()
- DataBook$get_data_frame_label()
- DataBook$get_data_frame_metadata()
- DataBook$get_combined_metadata()
- DataBook$get_metadata()
- DataBook$get_data_names()
- DataBook$get_data_changed()
- DataBook$get_variables_metadata_changed()
- DataBook$get_metadata_changed()
- DataBook$get_calculations()
- DataBook$get_calculation_names()
- DataBook$dataframe_count()
- DataBook$set_data_frames_changed()
- DataBook$set_variables_metadata_changed()
- DataBook$set_metadata_changed()

- DataBook$add_columns_to_data()
- DataBook$get_columns_from_data()
- DataBook$create_graph_data_book()
- DataBook$add_object()
- DataBook$get_object_names()
- DataBook$get_objects()
- DataBook$get_object()
- DataBook$get_object_data()
- DataBook$get_objects_data()
- DataBook$get_last_object_data()
- DataBook$rename_object()
- DataBook$delete_objects()
- DataBook$reorder_objects()
- DataBook$get_from_object()
- DataBook$add_filter()
- DataBook$add_filter_as_levels()
- DataBook$current_filter()
- DataBook$set_current_filter()
- DataBook$get_filter()
- DataBook$get_filter_as_logical()
- DataBook$get_current_filter()
- DataBook$get_filter_row_names()
- DataBook$get_current_filter_name()
- DataBook$get_filter_names()
- DataBook$remove_current_filter()
- DataBook$filter_applied()
- DataBook$filter_string()
- DataBook$get_filter_as_instat_calculation()
- DataBook$add_column_selection()
- DataBook$current_column_selection()
- DataBook$set_current_column_selection()
- DataBook$get_column_selection()
- DataBook$get_column_selection_column_names()
- DataBook$get_column_selected_column_names()
- DataBook$get_current_column_selection()
- DataBook$get_current_column_selection_name()
- DataBook$get_column_selection_names()
- DataBook$remove_current_column_selection()
- DataBook$column_selection_applied()
- DataBook$replace_value_in_data()
- DataBook$paste_from_clipboard()
- DataBook$rename_column_in_data()
- DataBook$frequency_tables()
- DataBook$anova_tables()
- DataBook$cor()

- DataBook$remove_columns_in_data()
- DataBook$remove_rows_in_data()
- DataBook$get_next_default_column_name()
- DataBook$get_column_names()
- DataBook$reorder_columns_in_data()
- DataBook$insert_row_in_data()
- DataBook$get_data_frame_length()
- DataBook$get_next_default_dataframe_name()
- DataBook$delete_dataframes()
- DataBook$remove_link()
- DataBook$get_column_factor_levels()
- DataBook$get_factor_data_frame()
- DataBook$sort_dataframe()
- DataBook$rename_dataframe()
- DataBook$convert_column_to_type()
- DataBook$append_to_variables_metadata()
- DataBook$append_to_dataframe_metadata()
- DataBook$append_to_metadata()
- DataBook$add_metadata_field()
- DataBook$reorder_dataframes()
- DataBook$copy_columns()
- DataBook$drop_unused_factor_levels()
- DataBook$set_factor_levels()
- DataBook$edit_factor_level()
- DataBook$set_factor_reference_level()
- DataBook$get_column_count()
- DataBook$reorder_factor_levels()
- DataBook$get_data_type()
- DataBook$copy_data_frame()
- DataBook$copy_col_metadata_to_clipboard()
- DataBook$copy_data_frame_metadata_to_clipboard()
- DataBook$copy_to_clipboard()
- DataBook$set_hidden_columns()
- DataBook$unhide_all_columns()
- DataBook$set_hidden_data_frames()
- DataBook$get_hidden_data_frames()
- DataBook$set_row_names()
- DataBook$get_row_names()
- DataBook$set_protected_columns()
- DataBook$get_metadata_fields()
- DataBook$freeze_columns()
- DataBook$unfreeze_columns()
- DataBook$is_variables_metadata()
- DataBook$data_frame_exists()
- DataBook$add_key()

- `DataBook$is_key()`
- `DataBook$has_key()`
- `DataBook$get_keys()`
- `DataBook$add_new_comment()`
- `DataBook$get_comments()`
- `DataBook$get_links()`
- `DataBook$set_structure_columns()`
- `DataBook$add_dependent_columns()`
- `DataBook$set_column_colours()`
- `DataBook$has_colours()`
- `DataBook$remove_column_colours()`
- `DataBook$set_column_colours_by_metadata()`
- `DataBook$graph_one_variable()`
- `DataBook$make_date_yearmonthday()`
- `DataBook$make_date_yeardoy()`
- `DataBook$set_contrasts_of_factor()`
- `DataBook$create_factor_data_frame()`
- `DataBook$split_date()`
- `DataBook$make_inventory_plot()`
- `DataBook$import_NetCDF()`
- `DataBook$infill_missing_dates()`
- `DataBook$get_key_names()`
- `DataBook$remove_key()`
- `DataBook$add_climdex_indices()`
- `DataBook$is_metadata()`
- `DataBook$get_climatic_column_name()`
- `DataBook$merge_data()`
- `DataBook$get_corruption_data_names()`
- `DataBook$get_corruption_contract_data_names()`
- `DataBook$get_database_variable_names()`
- `DataBook$get_nc_variable_names()`
- `DataBook$has_database_connection()`
- `DataBook$database_connect()`
- `DataBook$get_database_connection()`
- `DataBook$set_database_connection()`
- `DataBook$database_disconnect()`
- `DataBook$get_db_table_row_count()`
- `DataBook$import_climsoft_metadata()`
- `DataBook$import_climsoft_data()`
- `DataBook$import_from_iri()`
- `DataBook$export_workspace()`
- `DataBook$set_links()`
- `DataBook$display_daily_graph()`
- `DataBook$create_variable_set()`
- `DataBook$update_variable_set()`

- DataBook$delete_variable_sets()
- DataBook$get_variable_sets_names()
- DataBook$get_variable_sets()
- DataBook$crops_definitions()
- DataBook$tidy_climatic_data()
- DataBook$get_geometry()
- DataBook$package_check()
- DataBook$download_from_IRI()
- DataBook$patch_climate_element()
- DataBook$visualize_element_na()
- DataBook$get_data_entry_data()
- DataBook$save_data_entry_data()
- DataBook$import_from_cds()
- DataBook$add_flag_fields()
- DataBook$remove_empty()
- DataBook$replace_values_with_NA()
- DataBook$has_labels()
- DataBook$wrap_or_unwrap_data()
- DataBook$anova_tables2()
- DataBook$display_daily_table()
- DataBook$add_comment()
- DataBook$delete_comment()
- DataBook$get_comment_ids()
- DataBook$get_comments_as_data_frame()
- DataBook$define_as_options_by_context()
- DataBook$update_links_rename_data_frame()
- DataBook$update_links_rename_column()
- DataBook$add_link()
- DataBook$get_link_names()
- DataBook$link_exists_from()
- DataBook$link_exists_between()
- DataBook$get_link_between()
- DataBook$link_exists_from_by_to()
- DataBook$get_linked_to_data_name()
- DataBook$get_linked_to_definition()
- DataBook$get_possible_linked_to_definition()
- DataBook$get_equivalent_columns()
- DataBook$link_between_containing()
- DataBook$view_link()
- DataBook$apply_calculation()
- DataBook$save_calculation()
- DataBook$apply_instat_calculation()
- DataBook$run_instat_calculation()
- DataBook$get_corresponding_link_columns()
- DataBook$get_link_columns_from_data_frames()

- DataBook$save_calc_output()
- DataBook$append_summaries_to_data_object()
- DataBook$calculate_summary()
- DataBook$summary()
- DataBook$convert_linked_variable()
- DataBook$remove_unused_station_year_combinations()
- DataBook$summary_table()
- DataBook$import_SST()
- DataBook$clone()

Method `new()`**:** Initialize a new DataBook object.

Usage:

```
DataBook$new(
  data_tables = list(),
  instat_obj_metadata = list(),
 data_tables_variables_metadata = rep(list(data.frame()), length(data_tables)),
  data_tables_metadata = rep(list(list()), length(data_tables)),
  data_tables_filters = rep(list(list()), length(data_tables)),
  data_tables_column_selections = rep(list(list()), length(data_tables)),
  imported_from = as.list(rep("", length(data_tables))),
  messages = TRUE,
  convert = TRUE,
  create = TRUE
)
```

Arguments:

`data_tables` A list of data frames to be included in the DataBook.

`instat_obj_metadata` Metadata for the instat object.

`data_tables_variables_metadata` A list of data frames, each containing metadata for the corresponding data table.

`data_tables_metadata` A list of lists, each containing metadata for the corresponding data table.

`data_tables_filters` A list of lists, each containing filter information for the corresponding data table.

`data_tables_column_selections` A list of lists, each containing column selection information for the corresponding data table.

`imported_from` A list of strings indicating the source from which each data table was imported.

`messages` A boolean indicating whether to display messages.

`convert` A boolean indicating whether to perform data conversion.

`create` A boolean indicating whether to create new data objects.

Method `standardise_country_names()`**:** Standardise country names in the specified data table.

Usage:

```
DataBook$standardise_country_names(data_name, country_columns = c())
```

Arguments:

`data_name` The name of the data table.

`country_columns` A vector of column names containing country data.

Method `define_as_climatic()`**:** Define a data table as climatic data.

Usage:

```
DataBook$define_as_climatic(data_name, types, key_col_names, key_name)
```

Arguments:

`data_name` The name of the data table.

`types` A vector specifying the types of climatic data.

`key_col_names` A vector of column names to be used as keys.

`key_name` The name of the key.

Method `define_corruption_outputs()`**:** Define corruption output columns in the specified data table.

Usage:

```
DataBook$define_corruption_outputs(data_name, output_columns = c())
```

Arguments:

`data_name` The name of the data table.

`output_columns` A vector of column names to be defined as corruption outputs.

Method `define_red_flags()`**:** Define red flag columns in the specified data table.

Usage:

```
DataBook$define_red_flags(data_name, red_flags = c())
```

Arguments:

`data_name` The name of the data table.

`red_flags` A vector of column names to be defined as red flags.

Method `define_as_procurement()`**:** Define a data table as procurement data.

Usage:

```
DataBook$define_as_procurement(
  data_name,
  primary_types = c(),
  calculated_types = c(),
  country_data_name,
  country_types,
  auto_generate = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`primary_types` A vector of primary types of procurement data.

`calculated_types` A vector of calculated types of procurement data.

`country_data_name` The name of the country-level data table.

`country_types` A vector of types for the country-level data.

`auto_generate` A boolean indicating whether to auto-generate procurement types.

Method `define_as_procurement_country_level_data()`**:** Define a data table as procurement country-level data.

Usage:

```
DataBook$define_as_procurement_country_level_data(
  data_name,
  contract_level_data_name,
  types = c(),
  auto_generate = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`contract_level_data_name` The name of the contract-level data table.

`types` A vector of types for the procurement country-level data.

`auto_generate` A boolean indicating whether to auto-generate procurement types.

Method `get_CRI_component_column_names()`**:** Get the names of CRI component columns in the specified data table.

Usage:

```
DataBook$get_CRI_component_column_names(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: A vector of CRI component column names.

Method `get_red_flag_column_names()`**:** Get the names of red flag columns in the specified data table.

Usage:

```
DataBook$get_red_flag_column_names(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: A vector of red flag column names.

Method `get_CRI_column_names()`**:** Get the names of CRI columns in the specified data table.

Usage:

```
DataBook$get_CRI_column_names(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: A vector of CRI column names.

Method `get_corruption_column_name()`**:** Get the name of the corruption column in the specified data table.

Usage:

```
DataBook$get_corruption_column_name(data_name, type)
```

Arguments:

`data_name` The name of the data table.

`type` The type of the corruption column.

Returns: The name of the corruption column.

Method `import_data()`**:** Imports data tables and their associated metadata into the DataBook object.

Usage:

```
DataBook$import_data(
  data_tables = list(),
 data_tables_variables_metadata = rep(list(data.frame()), length(data_tables)),
  data_tables_metadata = rep(list(list()), length(data_tables)),
  data_tables_filters = rep(list(list()), length(data_tables)),
  data_tables_column_selections = rep(list(list()), length(data_tables)),
  imported_from = as.list(rep("", length(data_tables))),
  data_names = NULL,
  messages = TRUE,
  convert = TRUE,
  create = TRUE,
  prefix = TRUE,
  add_to_graph_book = TRUE
)
```

Arguments:

`data_tables` A list of data tables to be imported.

`data_tables_variables_metadata` Metadata for the variables of each data table.

`data_tables_metadata` General metadata for each data table.

`data_tables_filters` Filters applied to each data table.

`data_tables_column_selections` Column selections for each data table.

`imported_from` The origin/source of the imported data.

`data_names` Optional names for the data tables.

`messages` A boolean indicating if messages should be displayed.

`convert` A boolean indicating if data conversion should occur.

`create` A boolean to create new data objects.

`prefix` A boolean indicating whether to prefix data names.

`add_to_graph_book` A boolean to add the data to a graph book.

Method `replace_instat_object()`**:** Replaces the instat object in the DataBook.

Usage:

```
DataBook$replace_instat_object(new_instat_object)
```

Arguments:

`new_instat_object` The new instat object to replace the existing one.

Method `set_data_objects()`**:** Sets the data objects within the DataBook.

Usage:

```
DataBook$set_data_objects(new_data_objects)
```

Arguments:

`new_data_objects` A list of data objects to be set.

Method `copy_data_object()`**:** Copies a data object with an optional filter and column selection.

Usage:

```
DataBook$copy_data_object(
  data_name,
  new_name,
  filter_name = "",
  column_selection_name = "",
  reset_row_names = TRUE
)
```

Arguments:

`data_name` The name of the data object to copy.

`new_name` The new name for the copied data object.

`filter_name` Optional filter to apply during the copy.

`column_selection_name` Optional column selection to apply during the copy.

`reset_row_names` A boolean indicating whether to reset row names.

Method `import_RDS()`**:** Imports data from an RDS file into the DataBook.

Usage:

```
DataBook$import_RDS(
  data_RDS,
  keep_existing = TRUE,
  overwrite_existing = FALSE,
  include_objects = TRUE,
  include_metadata = TRUE,
  include_logs = TRUE,
  include_filters = TRUE,
  include_column_selections = TRUE,
  include_calculations = TRUE,
  include_comments = TRUE
)
```

Arguments:

`data_RDS` The RDS file containing data.

`keep_existing` A boolean to keep existing data.

`overwrite_existing` A boolean to overwrite existing data if necessary.

`include_objects` A boolean to include objects in the import.

`include_metadata` A boolean to include metadata in the import.

`include_logs` A boolean to include logs in the import.

`include_filters` A boolean to include filters in the import.

`include_column_selections` A boolean to include column selections in the import.

`include_calculations` A boolean to include calculations in the import.

`include_comments` A boolean to include comments in the import.

Method `clone_data_object()`**:** Clones a data object with options to include metadata, logs, filters, etc.

Usage:

```
DataBook$clone_data_object(
  curr_data_object,
  include_objects = TRUE,
  include_metadata = TRUE,
  include_logs = TRUE,
  include_filters = TRUE,
  include_column_selections = TRUE,
  include_calculations = TRUE,
  include_comments = TRUE,
  include_scalars = TRUE,
  ...
)
```

Arguments:

`curr_data_object` The current data object to be cloned.

`include_objects` A boolean to include objects in the clone.

`include_metadata` A boolean to include metadata in the clone.

`include_logs` A boolean to include logs in the clone.

`include_filters` A boolean to include filters in the clone.

`include_column_selections` A boolean to include column selections in the clone.

`include_calculations` A boolean to include calculations in the clone.

`include_comments` A boolean to include comments in the clone.

`include_scalars` A boolean to include scalars in the clone.

`...` Additional arguments passed to other methods.

Method `clone_instat_calculation()`**:** Clones an instat calculation with manipulations and sub-calculations.

Usage:

```
DataBook$clone_instat_calculation(curr_instat_calculation, ...)
```

Arguments:

`curr_instat_calculation` The current instat calculation to be cloned.

`...` Additional arguments passed to other methods.

Method `import_from_ODK()`**:** Imports data from an ODK platform.

Usage:

```
DataBook$import_from_ODK(username, form_name, platform)
```

Arguments:

`username` The username for ODK.

`form_name` The name of the ODK form.

`platform` The platform used for ODK.

Method `set_meta()`**:** Sets metadata for the DataBook.

Usage:

```
DataBook$set_meta(new_meta)
```

Arguments:

`new_meta` A list of metadata to be set.

Method `set_objects()`**:** Sets objects in the DataBook.

Usage:

```
DataBook$set_objects(new_objects)
```

Arguments:

`new_objects` A list of objects to be set.

Method `set_undo_history()`**:** Set the undo history for the DataBook.

Usage:

```
DataBook$set_undo_history(new_undo_history)
```

Arguments:

`new_undo_history` List, new undo history to set.

Method `set_scalars()`**:** Set the scalars for the DataBook.

Usage:

```
DataBook$set_scalars(new_scalars)
```

Arguments:

`new_scalars` List, new scalars to set.

Method `get_scalars()`**:** Retrieve scalars for a specific data object or overall DataBook.

Usage:

```
DataBook$get_scalars(data_name)
```

Arguments:

`data_name` Character, the name of the data object to retrieve scalars for. Defaults to overall DataBook if NULL or `overall_label`.

Returns: List of scalars.

Method `get_scalar_names()`**:** Retrieve scalar names for a specific data table.

Usage:

```
DataBook$get_scalar_names(
  data_name,
  as_list = FALSE,
  excluded_items = c(),
  ...
)
```

Arguments:

`data_name` The name of the data table.

`as_list` A boolean indicating whether to return results as a list.

`excluded_items` A vector of excluded items.

`...` Additional arguments passed to other methods.

Method `get_scalar_value()`**:** Retrieve the value of a specific scalar for a given data object.

Usage:

```
DataBook$get_scalar_value(data_name, scalar_name)
```

Arguments:

`data_name` Character, the name of the data object.

`scalar_name` Character, the name of the scalar to retrieve.

Returns: The value of the specified scalar.

Method `add_scalar()`**:** Add a scalar to a specific data object or the overall DataBook.

Usage:

```
DataBook$add_scalar(data_name, scalar_name = "", scalar_value)
```

Arguments:

`data_name` Character, the name of the data object. Adds to the overall DataBook if NULL or `overall_label`.

`scalar_name` Character, the name of the scalar. Defaults to a generated name if missing.

`scalar_value` The value of the scalar to add.

Method `set_enable_disable_undo()`**:** Enable or disable undo functionality for a specific data object.

Usage:

```
DataBook$set_enable_disable_undo(data_name, disable_undo)
```

Arguments:

`data_name` Character, the name of the data object.

`disable_undo` Logical, whether to disable undo functionality.

Method `is_undo()`**:** Check if undo functionality is enabled for a specific data object.

Usage:

```
DataBook$is_undo(data_name)
```

Arguments:

`data_name` Character, the name of the data object.

Returns: Logical, whether undo is enabled.

Method `has_undo_history()`**:** Check if there is undo history for a specific data object.

Usage:

```
DataBook$has_undo_history(data_name)
```

Arguments:

`data_name` Character, the name of the data object.

Returns: Logical, whether undo history exists.

Method `undo_last_action()`**:** Undo the last action for a specific data object.

Usage:

```
DataBook$undo_last_action(data_name)
```

Arguments:

`data_name` Character, the name of the data object.

Method `redo_last_action()`**:** Redo the last undone action for a specific data object.

Usage:

```
DataBook$redo_last_action(data_name)
```

Arguments:

`data_name` Character, the name of the data object.

Method `get_column_climatic_type()`**:** Retrieve the climatic type attribute for a specific column in a given data object.

Usage:

```
DataBook$get_column_climatic_type(data_name, col_name, attr_name)
```

Arguments:

`data_name` Character, the name of the data object.

`col_name` Character, the name of the column.

`attr_name` Character, the name of the attribute to retrieve.

Returns: The value of the specified attribute, or NULL if not available.

Method `append_data_object()`**:** Appends a data object to the DataBook.

Usage:

```
DataBook$append_data_object(name, obj, add_to_graph_book = TRUE)
```

Arguments:

`name` The name of the data object.

`obj` The data object to append.

`add_to_graph_book` A boolean to add the data to the graph book.

Method `get_data_objects()`**:** Retrieve data objects from the DataBook by name.

Usage:

```
DataBook$get_data_objects(data_name, as_list = FALSE, ...)
```

Arguments:

`data_name` The name or index of the data object(s) to retrieve.

`as_list` A boolean to return the data objects as a list (default: FALSE).

`...` Additional arguments passed to other methods.

Method `get_data_frame()`**:** Retrieve data frames from the DataBook.

Usage:

```
DataBook$get_data_frame(
  data_name,
  convert_to_character = FALSE,
  stack_data = FALSE,
  include_hidden_columns = TRUE,
  use_current_filter = TRUE,
  ...
)
```

Arguments:

`data_name` The name of the data frame to retrieve.

`convert_to_character` A boolean indicating whether to convert data to character type (default: FALSE).

`stack_data` A boolean to stack data (default: FALSE).

`include_hidden_columns` A boolean to include hidden columns (default: TRUE).

`use_current_filter` A boolean to apply the current filter (default: TRUE).

`...` Additional arguments passed to other methods.

Method `get_variables_metadata()`**:** Retrieve metadata for variables in a data frame.

Usage:

```
DataBook$get_variables_metadata(
  data_name,
  data_type = "all",
  convert_to_character = FALSE,
  property,
  column,
  ...
)
```

Arguments:

`data_name` The name of the data frame.

`data_type` The type of data to retrieve (default: "all").

`convert_to_character` A boolean indicating whether to convert data to character type (default: FALSE).

`property` The specific property to retrieve.

`column` The column for which metadata is to be retrieved.

`...` Additional arguments passed to other methods.

Method `get_column_data_types()`**:** Retrieve data types for specific columns in a data frame.

Usage:

```
DataBook$get_column_data_types(data_name, columns)
```

Arguments:

`data_name` The name of the data frame.

`columns` A vector of columns to retrieve the data types for.

Method `get_column_labels()`**:** Retrieve labels for specific columns in a data frame.

Usage:

```
DataBook$get_column_labels(data_name, columns)
```

Arguments:

`data_name` The name of the data frame.

`columns` A vector of columns to retrieve the labels for.

Method `get_data_frame_label()`**:** Retrieve the label of a data frame.

Usage:

```
DataBook$get_data_frame_label(data_name, use_current_filter = FALSE)
```

Arguments:

`data_name` The name of the data frame.

`use_current_filter` A boolean indicating whether to use the current filter (default: FALSE).

Method `get_data_frame_metadata()`**:** Retrieve metadata for a data frame.

Usage:

```
DataBook$get_data_frame_metadata(
  data_name,
  label,
  include_calculated = TRUE,
  excluded_not_for_display = TRUE
)
```

Arguments:

`data_name` The name of the data frame.

`label` The label for the metadata to retrieve.

`include_calculated` A boolean indicating whether to include calculated columns (default: TRUE).

`excluded_not_for_display` A boolean to exclude columns not for display (default: TRUE).

Method `get_combined_metadata()`**:** Retrieve combined metadata across all data objects in the DataBook.

Usage:

```
DataBook$get_combined_metadata(convert_to_character = FALSE)
```

Arguments:

`convert_to_character` A boolean to convert the metadata to a character matrix (default: FALSE).

Method `get_metadata()`**:** Retrieve metadata for a specific property.

Usage:

```
DataBook$get_metadata(name, ...)
```

Arguments:

`name` The name of the metadata to retrieve.

`...` Additional arguments passed to other methods.

Method `get_data_names()`**:** Retrieve data names from the data book.

Usage:

```
DataBook$get_data_names(
  as_list = FALSE,
  include,
  exclude,
  excluded_items,
  include_hidden = TRUE,
  ...
)
```

Arguments:

`as_list` A boolean indicating whether to return results as a list.

`include` A vector of names to include.

`exclude` A vector of names to exclude.

`excluded_items` A vector of excluded items.

`include_hidden` A boolean indicating whether to include hidden items.

`...` Additional arguments passed to other methods.

Method `get_data_changed()`**:** Check if data has changed.

Usage:

```
DataBook$get_data_changed(data_name)
```

Arguments:

`data_name` The name of the data table to check.

Method `get_variables_metadata_changed()`**:** Check if variables metadata has changed.

Usage:

```
DataBook$get_variables_metadata_changed(data_name)
```

Arguments:

`data_name` The name of the data table to check.

Method `get_metadata_changed()`**:** Check if metadata has changed.

Usage:

```
DataBook$get_metadata_changed(data_name)
```

Arguments:

`data_name` The name of the data table to check.

Method `get_calculations()`**:** Retrieve calculations for a specific data table.

Usage:

```
DataBook$get_calculations(data_name)
```

Arguments:

`data_name` The name of the data table.

Method `get_calculation_names()`**:** Retrieve calculation names for a specific data table.

Usage:

```
DataBook$get_calculation_names(
  data_name,
  as_list = FALSE,
  excluded_items = c()
)
```

Arguments:

`data_name` The name of the data table.

`as_list` A boolean indicating whether to return results as a list.

`excluded_items` A vector of excluded items.

Method `dataframe_count()`**:** Get the count of data frames in the DataBook.

Usage:

```
DataBook$dataframe_count()
```

Returns: An integer representing the number of data frames.

Method `set_data_frames_changed()`**:** Set the data changed status for the specified data frame.

Usage:

```
DataBook$set_data_frames_changed(data_name = "", new_val)
```

Arguments:

`data_name` The name of the data frame. Defaults to an empty string.

`new_val` A boolean indicating the new changed status.

Method `set_variables_metadata_changed()`**:** Set the variables metadata changed status for the specified data frame.

Usage:

```
DataBook$set_variables_metadata_changed(data_name = "", new_val)
```

Arguments:

`data_name` The name of the data frame. Defaults to an empty string.

`new_val` A boolean indicating the new changed status.

Method `set_metadata_changed()`**:** Set the metadata changed status for the specified data frame.

Usage:

```
DataBook$set_metadata_changed(data_name = "", new_val)
```

Arguments:

`data_name` The name of the data frame. Defaults to an empty string.

`new_val` A boolean indicating the new changed status.

Method `add_columns_to_data()`**:** Add columns to the specified data frame.

Usage:

```
DataBook$add_columns_to_data(
  data_name,
  col_name = "",
  col_data,
  use_col_name_as_prefix = FALSE,
  hidden = FALSE,
  before,
  adjacent_column = "",
  num_cols,
  require_correct_length = TRUE,
  keep_existing_position = TRUE
)
```

Arguments:

`data_name` The name of the data frame.

`col_name` The name of the new column.

`col_data` The data for the new column.

`use_col_name_as_prefix` A boolean indicating if the column name should be prefixed.

`hidden` A boolean indicating if the column should be hidden.

`before` The name of the column before which to insert the new column.

`adjacent_column` The name of an adjacent column.

`num_cols` The number of columns to add.

`require_correct_length` A boolean indicating if the lengths of the data should match.

`keep_existing_position` A boolean indicating if the existing column position should be maintained.

Method `get_columns_from_data()`**:** Get specified columns from the data frame.

Usage:

```
DataBook$get_columns_from_data(
  data_name,
  col_names,
  from_stacked_data = FALSE,
  force_as_data_frame = FALSE,
  use_current_filter = TRUE,
  remove_labels = FALSE,
  drop_unused_filter_levels = FALSE
)
```

Arguments:

`data_name` The name of the data frame.

`col_names` A vector of column names to retrieve.

`from_stacked_data` A boolean indicating if the data is stacked.

`force_as_data_frame` A boolean indicating if the output should be a data frame.

`use_current_filter` A boolean indicating if the current filter should be used.

`remove_labels` A boolean indicating if labels should be removed.

`drop_unused_filter_levels` A boolean indicating if unused filter levels should be dropped.

Method `create_graph_data_book()`**:** Create a new graph data book and assign it to the global environment.

Usage:

```
DataBook$create_graph_data_book()
```

Method `add_object()`**:** Add an object to the DataBook.

Usage:

```
DataBook$add_object(
  data_name = NULL,
  object_name = NULL,
  object_type_label,
  object_format,
  object
)
```

Arguments:

`data_name` The name of the data frame.

`object_name` The name of the object. If NULL, a default name is generated.

`object_type_label` The label for the object type.

`object_format` The format of the object.

`object` The object to add.

Method `get_object_names()`**:** Get the names of objects in the DataBook.

Usage:

```
DataBook$get_object_names(
  data_name = NULL,
  object_type_label = NULL,
  as_list = FALSE,
  ...
)
```

Arguments:

`data_name` The name of the data frame.

`object_type_label` The label for the object type.

`as_list` A boolean indicating if the output should be a list.

`...` Additional arguments passed to other methods.

Returns: A vector of object names.

Method `get_objects()`**:** Get objects from the DataBook.

Usage:

```
DataBook$get_objects(data_name = NULL, object_type_label = NULL)
```

Arguments:

`data_name` The name of the data frame.

`object_type_label` The label for the object type.

Returns: A list of objects.

Method `get_object()`**:** Get a specific object from the DataBook.

Usage:

```
DataBook$get_object(data_name = NULL, object_name)
```

Arguments:

`data_name` The name of the data frame.

`object_name` The name of the object to retrieve.

Returns: The requested object, or NULL if not found.

Method `get_object_data()`**:** Get the data of a specific object.

Usage:

```
DataBook$get_object_data(data_name = NULL, object_name, as_file = FALSE)
```

Arguments:

`data_name` The name of the data frame.

`object_name` The name of the object.

`as_file` A boolean indicating if the output should be formatted as a file.

Returns: The object data or NULL if not found.

Method `get_objects_data()`**:** Retrieve data objects from a specified data table.

Usage:

```
DataBook$get_objects_data(
  data_name = NULL,
  object_names = NULL,
  as_files = FALSE
)
```

Arguments:

`data_name` The name of the data table.

`object_names` A character vector of object names to retrieve.

`as_files` A boolean indicating whether to return data as files.

Method `get_last_object_data()`**:** Retrieve the last object data based on the specified type label.

Usage:

```
DataBook$get_last_object_data(object_type_label, as_file = TRUE)
```

Arguments:

`object_type_label` The label of the object type to retrieve.

`as_file` A boolean indicating whether to return the data as a file.

Method `rename_object()`**:** Rename an object within a specified data table.

Usage:

```
DataBook$rename_object(
  data_name,
  object_name,
  new_name,
  object_type = "object"
)
```

Arguments:

`data_name` The name of the data table.

`object_name` The current name of the object to rename.

`new_name` The new name for the object.

`object_type` The type of the object being renamed.

Method `delete_objects()`**:** Delete specified objects from a data table.

Usage:

```
DataBook$delete_objects(data_name, object_names, object_type = "object")
```

Arguments:

`data_name` The name of the data table.

`object_names` A character vector of object names to delete.

`object_type` The type of the objects being deleted.

Method `reorder_objects()`**:** Reorder objects in a specified data table.

Usage:

```
DataBook$reorder_objects(data_name, new_order)
```

Arguments:

`data_name` The name of the data table.

`new_order` A character vector specifying the new order of object names.

Method `get_from_object()`**:** Retrieve a value from a specified object.

Usage:

```
DataBook$get_from_object(data_name, object_name, value1, value2, value3)
```

Arguments:

`data_name` The name of the data table.

`object_name` The name of the object to retrieve data from.

`value1` The first value/key to retrieve.

`value2` The second value/key to retrieve.

`value3` The third value/key to retrieve.

Method `add_filter()`**:** Add a filter to the specified data table.

Usage:

```
DataBook$add_filter(
  data_name,
  filter,
  filter_name = "",
  replace = TRUE,
  set_as_current_filter = FALSE,
  na.rm = TRUE,
  is_no_filter = FALSE,
  and_or = "&",
  inner_not = FALSE,
  outer_not = FALSE
)
```

Arguments:

`data_name` The name of the data table.

`filter` The filter to apply.

`filter_name` The name of the filter.

`replace` A boolean indicating whether to replace existing filters.

`set_as_current_filter` A boolean indicating whether to set this filter as the current filter.

`na.rm` A boolean indicating whether to remove NA values.

`is_no_filter` A boolean indicating whether this is a "no filter".

`and_or` A string indicating whether to apply AND or OR logic.

`inner_not` A boolean for inner negation.

`outer_not` A boolean for outer negation.

Method `add_filter_as_levels()`**:** Add a filter as levels for a specified column in the data table.

Usage:

```
DataBook$add_filter_as_levels(data_name, filter_levels, column)
```

Arguments:

`data_name` The name of the data table.
`filter_levels` The levels of the filter to add.
`column` The column to apply the filter to.

Method `current_filter()`**:** Retrieve the current filter applied to a specified data table.

Usage:

```
DataBook$current_filter(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: The current filter.

Method `set_current_filter()`**:** Set the current filter for a specified data table.

Usage:

```
DataBook$set_current_filter(data_name, filter_name = "")
```

Arguments:

`data_name` The name of the data table.
`filter_name` The name of the filter to set as current.

Method `get_filter()`**:** Retrieve a specific filter by name.

Usage:

```
DataBook$get_filter(data_name, filter_name)
```

Arguments:

`data_name` The name of the data table.
`filter_name` The name of the filter to retrieve.

Returns: The requested filter.

Method `get_filter_as_logical()`**:** Retrieve a filter as a logical vector.

Usage:

```
DataBook$get_filter_as_logical(data_name, filter_name)
```

Arguments:

`data_name` The name of the data table.
`filter_name` The name of the filter to retrieve.

Returns: A logical vector representing the filter.

Method `get_current_filter()`**:** Retrieve the currently applied filter for a specified data table.

Usage:

```
DataBook$get_current_filter(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: The currently applied filter.

Method `get_filter_row_names()`**:** Retrieve row names based on a specific filter for a data table.

Usage:

```
DataBook$get_filter_row_names(data_name, filter_name)
```

Arguments:

`data_name` The name of the data table.

`filter_name` The name of the filter to use for retrieval.

Returns: A vector of row names.

Method `get_current_filter_name()`**:** Retrieve the name of the current filter for a specified data table.

Usage:

```
DataBook$get_current_filter_name(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: The name of the current filter.

Method `get_filter_names()`**:** Retrieve the names of all filters for a specified data table.

Usage:

```
DataBook$get_filter_names(
  data_name,
  as_list = FALSE,
  include = list(),
  exclude = list(),
  excluded_items = c()
)
```

Arguments:

`data_name` The name of the data table.

`as_list` A boolean indicating whether to return as a list.

`include` A list of items to include.

`exclude` A list of items to exclude.

`excluded_items` A vector of excluded items.

Returns: A vector or list of filter names.

Method `remove_current_filter()`**:** Remove the current filter from a specified data table.

Usage:

```
DataBook$remove_current_filter(data_name)
```

Arguments:

`data_name` The name of the data table.

Method `filter_applied()`**:** Check if a filter is applied to a specified data table.

Usage:

```
DataBook$filter_applied(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: A boolean indicating if a filter is applied.

Method `filter_string()`**:** Retrieve the string representation of a specific filter.

Usage:

```
DataBook$filter_string(data_name, filter_name)
```

Arguments:

`data_name` The name of the data table.

`filter_name` The name of the filter to retrieve the string for.

Returns: A string representation of the filter.

Method `get_filter_as_instat_calculation()`**:** Retrieve a filter as an Instat calculation.

Usage:

```
DataBook$get_filter_as_instat_calculation(data_name, filter_name)
```

Arguments:

`data_name` The name of the data table.

`filter_name` The name of the filter to retrieve.

Returns: The filter as an Instat calculation.

Method `add_column_selection()`**:** Add a column selection to a data table.

Usage:

```
DataBook$add_column_selection(
  data_name,
  column_selection,
  name = "",
  replace = TRUE,
  set_as_current = FALSE,
  is_everything = FALSE,
  and_or = "|"
)
```

Arguments:

`data_name` The name of the data table.

`column_selection` The selection of columns.

`name` Optional name for the selection.

`replace` Logical indicating if the selection should replace an existing one.

`set_as_current` Logical indicating if the selection should be set as current.

`is_everything` Logical indicating if all columns should be selected.

`and_or` String indicating the logical operator to use.

Method `current_column_selection()`**:** Get the current column selection for a data table.

Usage:

```
DataBook$current_column_selection(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: The current column selection.

Method `set_current_column_selection()`**:** Set the current column selection for a data table.

Usage:

```
DataBook$set_current_column_selection(data_name, name = "")
```

Arguments:

`data_name` The name of the data table.

`name` The name of the column selection to set as current.

Method `get_column_selection()`**:** Get a specific column selection from a data table.

Usage:

```
DataBook$get_column_selection(data_name, name)
```

Arguments:

`data_name` The name of the data table.

`name` The name of the column selection to retrieve.

Returns: The specified column selection.

Method `get_column_selection_column_names()`**:** Get column names from a specified filter in a data table.

Usage:

```
DataBook$get_column_selection_column_names(data_name, filter_name)
```

Arguments:

`data_name` The name of the data table.

`filter_name` The name of the filter.

Returns: Logical representation of the filter applied to column names.

Method `get_column_selected_column_names()`**:** Get column names selected in a specific column selection.

Usage:

```
DataBook$get_column_selected_column_names(
  data_name,
  column_selection_name = ""
)
```

Arguments:

`data_name` The name of the data table.

`column_selection_name` Optional name of the column selection.

Returns: A vector of selected column names.

Method `get_current_column_selection()`**:** Get the current column selection for a data table.

Usage:

```
DataBook$get_current_column_selection(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: The current column selection object.

Method `get_current_column_selection_name()`**:** Get the name of the current column selection for a data table.

Usage:

```
DataBook$get_current_column_selection_name(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: The name of the current column selection.

Method `get_column_selection_names()`**:** Get names of all column selections for a data table.

Usage:

```
DataBook$get_column_selection_names(
  data_name,
  as_list = FALSE,
  include = list(),
  exclude = list(),
  excluded_items = c()
)
```

Arguments:

`data_name` The name of the data table.

`as_list` Logical indicating if results should be returned as a list.

`include` List of items to include.

`exclude` List of items to exclude.

`excluded_items` Optional vector of excluded items.

Returns: A list or vector of column selection names.

Method `remove_current_column_selection()`**:** Remove the current column selection for a data table.

Usage:

```
DataBook$remove_current_column_selection(data_name)
```

Arguments:

`data_name` The name of the data table.

Method `column_selection_applied()`**:** Check if a column selection has been applied to a data table.

Usage:

```
DataBook$column_selection_applied(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: Logical indicating if a selection has been applied.

Method `replace_value_in_data()`**:** Replace values in the data of a specified table.

Usage:

```
DataBook$replace_value_in_data(
  data_name,
  col_names,
  rows,
  old_value,
  old_is_missing = FALSE,
  start_value = NA,
  end_value = NA,
  new_value,
  new_is_missing = FALSE,
  closed_start_value = TRUE,
  closed_end_value = TRUE,
  locf = FALSE,
  from_last = FALSE
)
```

Arguments:

`data_name` The name of the data table.
`col_names` Names of the columns to modify.
`rows` The rows to be modified.
`old_value` The value to replace.
`old_is_missing` Logical indicating if old_value is missing.
`start_value` Optional starting value for replacement.
`end_value` Optional ending value for replacement.
`new_value` The new value to replace with.
`new_is_missing` Logical indicating if new_value is missing.
`closed_start_value` Logical for start value closure.
`closed_end_value` Logical for end value closure.
`locf` Logical indicating if last observation carried forward should be used.
`from_last` Logical indicating if the replacement should start from the last.

Method `paste_from_clipboard()`**:** Paste data from the clipboard into a specified data table.

Usage:

```
DataBook$paste_from_clipboard(
  data_name,
  col_names,
  start_row_pos = 1,
  first_clip_row_is_header = TRUE,
  clip_board_text
)
```

Arguments:

`data_name` The name of the data table.
`col_names` Names of the columns to paste into.
`start_row_pos` Position to start pasting in the data.
`first_clip_row_is_header` Logical indicating if the first row of the clipboard is a header.
`clip_board_text` The text to paste from the clipboard.

Method `rename_column_in_data()`**:** Rename a column in the data of a specified table.

Usage:

```
DataBook$rename_column_in_data(
  data_name,
  column_name = NULL,
  new_val = NULL,
  label = "",
  type = "single",
  .fn,
  .cols = everything(),
  new_column_names_df,
  new_labels_df,
  ...
)
```

Arguments:

`data_name` The name of the data table.
`column_name` Optional name of the column to rename.
`new_val` The new name for the column.

`label` Optional label for the column.

`type` Type of renaming operation.

`.fn` Optional function for renaming.

`.cols` Optional columns to rename.

`new_column_names_df` Data frame with new column names.

`new_labels_df` Data frame with new labels.

`...` Additional arguments passed to other methods.

Method `frequency_tables()`**:** Generate frequency tables for specified columns in a data table.

Usage:

```
DataBook$frequency_tables(
  data_name,
  x_col_names,
  y_col_name,
  n_column_factors = 1,
  store_results = TRUE,
  drop = TRUE,
  na.rm = FALSE,
  summary_name = NA,
  include_margins = FALSE,
  return_output = TRUE,
  treat_columns_as_factor = FALSE,
  page_by = "default",
  as_html = TRUE,
  signif_fig = 2,
  na_display = "",
  na_level_display = "NA",
  weights = NULL,
  caption = NULL,
  result_names = NULL,
  percentage_type = "none",
  perc_total_columns = NULL,
  perc_total_factors = c(),
  perc_total_filter = NULL,
  perc_decimal = FALSE,
  margin_name = "(All)",
  additional_filter,
  ...
)
```

Arguments:

`data_name` The name of the data table.

`x_col_names` Names of columns for x-axis.

`y_col_name` Name of column for y-axis.

`n_column_factors` Number of column factors.

`store_results` Logical indicating if results should be stored.

`drop` Logical indicating if to drop unused levels.

`na.rm` Logical indicating if NA values should be removed.

`summary_name` Optional name for the summary.

`include_margins` Logical indicating if margins should be included.

`return_output` Logical indicating if output should be returned.

`treat_columns_as_factor` Logical indicating if columns should be treated as factors.

`page_by` Pagination option.

`as_html` Logical indicating if output should be in HTML format.

`signif_fig` Number of significant figures.

`na_display` Optional display for NA values.

`na_level_display` Optional display for NA levels.

`weights` Optional weights for frequency calculations.

`caption` Optional caption for the output.

`result_names` Optional names for results.

`percentage_type` Type of percentage calculation.

`perc_total_columns` Optional columns for total percentages.

`perc_total_factors` Optional factors for total percentages.

`perc_total_filter` Optional filter for total percentages.

`perc_decimal` Logical indicating if percentages should be decimal.

`margin_name` Name for the margin.

`additional_filter` Optional additional filter.

`...` Additional parameters.

Method `anova_tables()`**:** Generate ANOVA tables for specified columns in a data table.

Usage:

```
DataBook$anova_tables(
  data_name,
  x_col_names,
  y_col_name,
  signif.stars = FALSE,
  sign_level = FALSE,
  means = FALSE
)
```

Arguments:

`data_name` The name of the data table.

`x_col_names` Names of columns for x-axis.

`y_col_name` Name of column for y-axis.

`signif.stars` Logical indicating if significance stars should be shown.

`sign_level` Logical indicating if significance level should be displayed.

`means` Logical indicating if means should be displayed.

Method `cor()`**:** Calculate correlation between specified columns in a data table.

Usage:

```
DataBook$cor(
  data_name,
  x_col_names,
  y_col_name,
  use = "everything",
  method = c("pearson", "kendall", "spearman")
)
```

Arguments:

`data_name` The name of the data table.

`x_col_names` Names of columns for x-axis.

`y_col_name` Name of column for y-axis.

`use` How to handle missing values.

`method` The method to use for correlation.

Method `remove_columns_in_data()`**:** Remove specified columns from a data table.

Usage:

```
DataBook$remove_columns_in_data(data_name, cols, allow_delete_all = FALSE)
```

Arguments:

`data_name` The name of the data table.

`cols` The columns to remove.

`allow_delete_all` Logical indicating if all columns can be deleted.

Method `remove_rows_in_data()`**:** Remove specified rows from a data table.

Usage:

```
DataBook$remove_rows_in_data(data_name, row_names)
```

Arguments:

`data_name` The name of the data table.

`row_names` The names of the rows to remove.

Method `get_next_default_column_name()`**:** Get the next available default column name for a data table.

Usage:

```
DataBook$get_next_default_column_name(data_name, prefix)
```

Arguments:

`data_name` The name of the data table.

`prefix` The prefix for the new column name.

Returns: The next available default column name.

Method `get_column_names()`**:** Retrieve the column names of a data table.

Usage:

```
DataBook$get_column_names(
  data_name,
  as_list = FALSE,
  include = list(),
  exclude = list(),
  excluded_items = c(),
  max_no,
  use_current_column_selection = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`as_list` A boolean indicating whether to return the names as a list.

`include` A list of column names to include.

`exclude` A list of column names to exclude.

`excluded_items` A vector of items to be excluded.

max_no The maximum number of column names to return.

use_current_column_selection A boolean indicating whether to use the current column selection.

Method `reorder_columns_in_data()`**:** Reorder the columns in a data table.

Usage:

```
DataBook$reorder_columns_in_data(data_name, col_order)
```

Arguments:

data_name The name of the data table.

col_order A vector specifying the new order of columns.

Method `insert_row_in_data()`**:** Insert rows into a data table.

Usage:

```
DataBook$insert_row_in_data(
  data_name,
  start_row,
  row_data = c(),
  number_rows = 1,
  before = FALSE
)
```

Arguments:

data_name The name of the data table.

start_row The row number to start inserting at.

row_data A vector of data for the new row(s).

number_rows The number of rows to insert.

before A boolean indicating whether to insert before the start_row.

Method `get_data_frame_length()`**:** Get the length of a data frame.

Usage:

```
DataBook$get_data_frame_length(data_name, use_current_filter = FALSE)
```

Arguments:

data_name The name of the data frame.

use_current_filter A boolean indicating whether to use the current filter.

Method `get_next_default_dataframe_name()`**:** Get the next default name for a data frame.

Usage:

```
DataBook$get_next_default_dataframe_name(
  prefix,
  include_index = TRUE,
  start_index = 1
)
```

Arguments:

prefix The prefix for the new data frame name.

include_index A boolean indicating whether to include an index.

start_index The starting index for naming.

Method `delete_dataframes()`**:** Delete specified data frames.

Usage:

```
DataBook$delete_dataframes(data_names, delete_graph_book = TRUE)
```

Arguments:

`data_names` A vector of data frame names to delete.

`delete_graph_book` A boolean indicating whether to delete the associated graph book.

Method `remove_link()`**:** Remove a link by its name.

Usage:

```
DataBook$remove_link(link_name)
```

Arguments:

`link_name` The name of the link to remove.

Method `get_column_factor_levels()`**:** Get the factor levels of a specified column in a data table.

Usage:

```
DataBook$get_column_factor_levels(data_name, col_name = "")
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column to check.

Method `get_factor_data_frame()`**:** Get the data frame for a specified factor column.

Usage:

```
DataBook$get_factor_data_frame(
  data_name,
  col_name = "",
  include_levels = TRUE,
  include_NA_level = FALSE
)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column to retrieve.

`include_levels` A boolean indicating whether to include levels.

`include_NA_level` A boolean indicating whether to include NA as a level.

Method `sort_dataframe()`**:** Sort a data frame by specified column(s).

Usage:

```
DataBook$sort_dataframe(
  data_name,
  col_names = c(),
  decreasing = FALSE,
  na.last = TRUE,
  by_row_names = FALSE,
  row_names_as_numeric = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to sort by.

`decreasing` A boolean indicating whether to sort in decreasing order.

`na.last` A boolean indicating how to handle NA values.

`by_row_names` A boolean indicating whether to sort by row names.

`row_names_as_numeric` A boolean indicating whether row names should be treated as numeric.

Method `rename_dataframe()`**:** Rename a data frame and optionally update its label.

Usage:

```
DataBook$rename_dataframe(data_name, new_value = "", label = "")
```

Arguments:

`data_name` The current name of the data frame.

`new_value` The new name for the data frame.

`label` An optional label for the data frame.

Method `convert_column_to_type()`**:** Convert specified columns to a different type.

Usage:

```
DataBook$convert_column_to_type(
  data_name,
  col_names = c(),
  to_type,
  factor_values = NULL,
  set_digits,
  set_decimals = FALSE,
  keep_attr = TRUE,
  ignore_labels = FALSE,
  keep.labels = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to convert.

`to_type` The target data type.

`factor_values` Optional values for factors.

`set_digits` Number of digits to set.

`set_decimals` A boolean indicating whether to set decimals.

`keep_attr` A boolean indicating whether to keep attributes.

`ignore_labels` A boolean indicating whether to ignore labels.

`keep.labels` A boolean indicating whether to keep labels.

Method `append_to_variables_metadata()`**:** Appends a new property and its value to the metadata of specified columns in a data table.

Usage:

```
DataBook$append_to_variables_metadata(
  data_name,
  col_names,
  property,
  new_val = ""
)
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to which the property should be appended.

`property` The name of the property to append.

`new_val` The value of the property to append. Default is an empty string.

Returns: None

Method `append_to_dataframe_metadata()`**:** Appends a new property and its value to the metadata of a data table.

Usage:

```
DataBook$append_to_dataframe_metadata(data_name, property, new_val = "")
```

Arguments:

`data_name` The name of the data table.

`property` The name of the property to append.

`new_val` The value of the property to append. Default is an empty string.

Returns: None

Method `append_to_metadata()`**:** Appends a new property and its value to the metadata of the current object.

Usage:

```
DataBook$append_to_metadata(
  property,
  new_val = "",
  allow_override_special = FALSE
)
```

Arguments:

`property` The name of the property to append.

`new_val` The value of the property to append. Default is an empty string.

`allow_override_special` Boolean flag to allow overriding special properties. Default is FALSE.

Returns: None

Method `add_metadata_field()`**:** Adds a new metadata field and its value to the specified data table or all data tables.

Usage:

```
DataBook$add_metadata_field(data_name, property, new_val = "")
```

Arguments:

`data_name` The name of the data table. Use overall_label to apply to all data tables.

`property` The name of the property to append.

`new_val` The value of the property to append. Default is an empty string.

Returns: None

Method `reorder_dataframes()`**:** Reorders the dataframes in the object according to the specified order.

Usage:

```
DataBook$reorder_dataframes(data_frames_order)
```

Arguments:

`data_frames_order` A vector specifying the new order of dataframes.

Returns: None

Method `copy_columns()`**:** Copies specified columns from a data table to another location or clipboard.

Usage:

```
DataBook$copy_columns(data_name, col_names = "", copy_to_clipboard = FALSE)
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to copy.

`copy_to_clipboard` Boolean flag to copy to clipboard. Default is FALSE.

Returns: None

Method `drop_unused_factor_levels()`**:** Drops unused levels from a factor column in the specified data table.

Usage:

```
DataBook$drop_unused_factor_levels(data_name, col_name)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column.

Returns: None

Method `set_factor_levels()`**:** Sets new levels for a factor column in the specified data table.

Usage:

```
DataBook$set_factor_levels(
  data_name,
  col_name,
  new_labels,
  new_levels,
  set_new_labels = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column.

`new_labels` A vector of new labels for the factor levels.

`new_levels` A vector of new levels.

`set_new_labels` Boolean flag to set new labels. Default is TRUE.

Returns: None

Method `edit_factor_level()`**:** Edits a level in a factor column in the specified data table.

Usage:

```
DataBook$edit_factor_level(data_name, col_name, old_level, new_level)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column.

`old_level` The old level to replace.

`new_level` The new level to set.

Returns: None

Method `set_factor_reference_level()`**:** Sets the reference level for a factor column in the specified data table.

Usage:

```
DataBook$set_factor_reference_level(data_name, col_name, new_ref_level)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column.

`new_ref_level` The new reference level.

Returns: None

Method `get_column_count()`**:** Returns the number of columns in the specified data table.

Usage:

```
DataBook$get_column_count(data_name, use_column_selection = FALSE)
```

Arguments:

`data_name` The name of the data table.

`use_column_selection` Boolean flag to use column selection. Default is FALSE.

Returns: The number of columns.

Method `reorder_factor_levels()`**:** Reorders the levels of a factor column in the specified data table.

Usage:

```
DataBook$reorder_factor_levels(data_name, col_name, new_level_names)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column.

`new_level_names` A vector specifying the new order of factor levels.

Returns: None

Method `get_data_type()`**:** Returns the data type of the specified column in the given data table.

Usage:

```
DataBook$get_data_type(data_name, col_name)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column.

Returns: The data type of the column.

Method `copy_data_frame()`**:** Copies a data frame to a new name or clipboard.

Usage:

```
DataBook$copy_data_frame(
  data_name,
  new_name,
  label = "",
  copy_to_clipboard = FALSE
)
```

Arguments:

`data_name` The name of the data table.

`new_name` The new name for the copied data frame.

`label` A label for the new data frame. Default is an empty string.

`copy_to_clipboard` Boolean flag to copy to clipboard. Default is FALSE.

Returns: None

Method `copy_col_metadata_to_clipboard():` Copies the metadata of specified columns to the clipboard.

Usage:

```
DataBook$copy_col_metadata_to_clipboard(data_name, property_names)
```

Arguments:

`data_name` The name of the data table.

`property_names` A vector of property names to copy. Default is all properties.

Returns: None

Method `copy_data_frame_metadata_to_clipboard():` Copies the metadata of the specified data table to the clipboard.

Usage:

```
DataBook$copy_data_frame_metadata_to_clipboard(data_name, property_names)
```

Arguments:

`data_name` The name of the data table.

`property_names` A vector of property names to copy. Default is all properties.

Returns: None

Method `copy_to_clipboard():` Copies the specified content to the clipboard.

Usage:

```
DataBook$copy_to_clipboard(content)
```

Arguments:

`content` The content to copy to the clipboard.

Returns: None

Method `set_hidden_columns():` Sets the specified columns as hidden in the given data table.

Usage:

```
DataBook$set_hidden_columns(data_name, col_names = c())
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to set as hidden.

Returns: None

Method `unhide_all_columns():` Unhides all columns in the specified data table or all data tables if data_name is missing.

Usage:

```
DataBook$unhide_all_columns(data_name)
```

Arguments:

`data_name` The name of the data table. If missing, applies to all data tables.

Returns: None

Method `set_hidden_data_frames()`**:** Sets the specified data tables as hidden.

Usage:

```
DataBook$set_hidden_data_frames(data_names = c())
```

Arguments:

`data_names` A vector of data table names to set as hidden.

Returns: None

Method `get_hidden_data_frames()`**:** Returns a list of hidden data tables.

Usage:

```
DataBook$get_hidden_data_frames()
```

Returns: A vector of hidden data table names.

Method `set_row_names()`**:** Sets the row names for the specified data table.

Usage:

```
DataBook$set_row_names(data_name, row_names)
```

Arguments:

`data_name` The name of the data table.

`row_names` A vector of row names to set.

Returns: None

Method `get_row_names()`**:** Returns the row names of the specified data table.

Usage:

```
DataBook$get_row_names(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: A vector of row names.

Method `set_protected_columns()`**:** Sets the specified columns as protected in the given data table.

Usage:

```
DataBook$set_protected_columns(data_name, col_names)
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to set as protected.

Returns: None

Method `get_metadata_fields()`**:** Returns the metadata fields of the specified data table.

Usage:

```
DataBook$get_metadata_fields(
  data_name,
  include_overall,
  as_list = FALSE,
  include,
  exclude,
  excluded_items = c()
)
```

Arguments:

`data_name` The name of the data table.

`include_overall` Boolean flag to include overall metadata fields. Default is TRUE.

`as_list` Boolean flag to return the result as a list. Default is FALSE.

`include` A vector of metadata fields to include. Default is all fields.

`exclude` A vector of metadata fields to exclude. Default is none.

`excluded_items` A vector of metadata fields to exclude. Default is an empty vector.

Returns: A vector or list of metadata fields.

Method `freeze_columns()`**:** Freezes the specified columns in the given data table.

Usage:

```
DataBook$freeze_columns(data_name, column)
```

Arguments:

`data_name` The name of the data table.

`column` A vector of column names to freeze.

Returns: None

Method `unfreeze_columns()`**:** Unfreezes all columns in the specified data table.

Usage:

```
DataBook$unfreeze_columns(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: None

Method `is_variables_metadata()`**:** Checks if the specified property is metadata for the given columns in the data table.

Usage:

```
DataBook$is_variables_metadata(
  data_name,
  property,
  column,
  return_vector = FALSE
)
```

Arguments:

`data_name` The name of the data table.

`property` The name of the property to check.

`column` The name of the column.

`return_vector` Boolean flag to return the result as a vector. Default is FALSE.

Returns: A boolean value indicating if the property is metadata for the columns.

Method `data_frame_exists()`**:** Checks if the specified data table exists.

Usage:

```
DataBook$data_frame_exists(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: A boolean value indicating if the data table exists.

Method `add_key()`**:** Adds a key to the specified columns in the given data table.

Usage:

```
DataBook$add_key(data_name, col_names, key_name)
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to add as keys.

`key_name` The name of the key.

Returns: None

Method `is_key()`**:** Checks if the specified columns are keys in the given data table.

Usage:

```
DataBook$is_key(data_name, col_names)
```

Arguments:

`data_name` The name of the data table.

`col_names` A vector of column names to check.

Returns: A boolean value indicating if the columns are keys.

Method `has_key()`**:** Checks if the specified data table has a key.

Usage:

```
DataBook$has_key(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: A boolean value indicating if the data table has a key.

Method `get_keys()`**:** Returns the keys of the specified data table.

Usage:

```
DataBook$get_keys(data_name, key_name)
```

Arguments:

`data_name` The name of the data table.

`key_name` The name of the key. Default is all keys.

Returns: A list of keys.

Method `add_new_comment()`**:** Adds a new comment to the specified row and column in the given data table.

Usage:

```
DataBook$add_new_comment(data_name, row = "", column = "", comment)
```

Arguments:

`data_name` The name of the data table.

`row` The name of the row.

`column` The name of the column.

`comment` The comment text.

Returns: None

Method `get_comments()`**:** Returns the comments for the specified data table and comment ID.

Usage:

```
DataBook$get_comments(data_name, comment_id)
```

Arguments:

`data_name` The name of the data table.

`comment_id` The ID of the comment.

Returns: A data frame of comments.

Method `get_links()`**:** Returns the links for the specified link name or all links.

Usage:

```
DataBook$get_links(link_name, ...)
```

Arguments:

`link_name` The name of the link. Default is all links.

`...` Additional arguments passed to other methods.

Returns: A list of links.

Method `set_structure_columns()`**:** Sets the structure columns for the specified data table.

Usage:

```
DataBook$set_structure_columns(
  data_name,
  struc_type_1 = c(),
  struc_type_2 = c(),
  struc_type_3 = c()
)
```

Arguments:

`data_name` The name of the data table.

`struc_type_1` A vector of column names for the first structure type.

`struc_type_2` A vector of column names for the second structure type.

`struc_type_3` A vector of column names for the third structure type.

Returns: None

Method `add_dependent_columns()`**:** Adds dependent columns to the specified columns in the given data table.

Usage:

```
DataBook$add_dependent_columns(data_name, columns, dependent_cols)
```

Arguments:

`data_name` The name of the data table.

`columns` A vector of column names to add dependents to.

`dependent_cols` A vector of dependent column names.

Returns: None

Method `set_column_colours()`**:** Sets the colours for the specified columns in the given data table.

Usage:

```
DataBook$set_column_colours(data_name, columns, colours)
```

Arguments:

`data_name` The name of the data table.

`columns` A vector of column names to set colours for.

`colours` A vector of colours.

Returns: None

Method `has_colours()`: Checks if the specified columns have colours in the given data table.

Usage:

```
DataBook$has_colours(data_name, columns)
```

Arguments:

`data_name` The name of the data table.

`columns` A vector of column names to check.

Returns: A boolean value indicating if the columns have colours.

Method `remove_column_colours()`: Removes the colours from all columns in the specified data table.

Usage:

```
DataBook$remove_column_colours(data_name)
```

Arguments:

`data_name` The name of the data table.

Returns: None

Method `set_column_colours_by_metadata()`: Sets the colours for the specified columns based on metadata in the given data table.

Usage:

```
DataBook$set_column_colours_by_metadata(data_name, columns, property)
```

Arguments:

`data_name` The name of the data table.

`columns` A vector of column names to set colours for.

`property` The metadata property to use for setting colours.

Returns: None

Method `graph_one_variable()`: Creates a graph for a single variable in the specified data table.

Usage:

```
DataBook$graph_one_variable(
  data_name,
  columns,
  numeric = "geom_boxplot",
  categorical = "geom_bar",
  character = "geom_bar",
  output = "facets",
  free_scale_axis = FALSE,
  ncol = NULL,
  coord_flip = FALSE,
  ...
)
```

Arguments:

`data_name` The name of the data table.

`columns` A vector of column names to graph.

`numeric` The type of graph for numeric columns. Default is "geom_boxplot".

`categorical` The type of graph for categorical columns. Default is "geom_bar".

`character` The type of graph for character columns. Default is "geom_bar".

`output` The output type for the graph. Default is "facets".

`free_scale_axis` Boolean flag to allow free scaling of axes. Default is FALSE.

`ncol` The number of columns in the output. Default is NULL.

`coord_flip` Boolean flag to flip coordinates. Default is FALSE.

`...` Additional arguments passed to other methods.

Returns: None

Method `make_date_yearmonthday()`**:** Creates a date column from year, month, and day columns in the specified data table.

Usage:

```
DataBook$make_date_yearmonthday(
  data_name,
  year,
  month,
  day,
  f_year,
  f_month,
  f_day,
  year_format = "%Y",
  month_format = "%m"
)
```

Arguments:

`data_name` The name of the data table.

`year` The name of the year column.

`month` The name of the month column.

`day` The name of the day column.

`f_year` The format for the year column.

`f_month` The format for the month column.

`f_day` The format for the day column.

`year_format` The format for the year. Default is "%Y".

`month_format` The format for the month. Default is "%m".

Returns: None

Method `make_date_yeardoy()`**:** Creates a date column from year and day of year columns in the specified data table.

Usage:

```
DataBook$make_date_yeardoy(
  data_name,
  year,
  doy,
  base,
  doy_typical_length = "366"
)
```

Arguments:

`data_name` The name of the data table.

`year` The name of the year column.

`doy` The name of the day of year column.

`base` The base date for the day of year.

`doy_typical_length` The typical length of the day of year. Default is "366".

Returns: None

Method `set_contrasts_of_factor()`**:** Sets the contrasts for a factor column in the specified data table.

Usage:

```
DataBook$set_contrasts_of_factor(
  data_name,
  col_name,
  new_contrasts,
  defined_contr_matrix
)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column.

`new_contrasts` A vector of new contrasts.

`defined_contr_matrix` A defined contrast matrix.

Returns: None

Method `create_factor_data_frame()`**:** Creates a new data frame for a factor column in the specified data table.

Usage:

```
DataBook$create_factor_data_frame(
  data_name,
  factor,
  factor_data_frame_name,
  include_contrasts = FALSE,
  replace = FALSE,
  summary_count = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`factor` The name of the factor column.

`factor_data_frame_name` The name of the new data frame.

`include_contrasts` Boolean flag to include contrasts. Default is FALSE.

`replace` Boolean flag to replace the existing factor data frame. Default is FALSE.

`summary_count` Boolean flag to include summary count. Default is TRUE.

Returns: None

Method `split_date()`**:** Splits a date column into multiple date components in the specified data table.

Usage:

```
DataBook$split_date(
  data_name,
  col_name = "",
  year_val = FALSE,
  year_name = FALSE,
  leap_year = FALSE,
  month_val = FALSE,
  month_abbr = FALSE,
  month_name = FALSE,
  week_val = FALSE,
  week_abbr = FALSE,
  week_name = FALSE,
  weekday_val = FALSE,
  weekday_abbr = FALSE,
  weekday_name = FALSE,
  day = FALSE,
  day_in_month = FALSE,
  day_in_year = FALSE,
  day_in_year_366 = FALSE,
  pentad_val = FALSE,
  pentad_abbr = FALSE,
  dekad_val = FALSE,
  dekad_abbr = FALSE,
  quarter_val = FALSE,
  quarter_abbr = FALSE,
  with_year = FALSE,
  s_start_month = 1,
  s_start_day_in_month = 1,
  days_in_month = FALSE
)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the date column.

`year_val` Boolean flag to include year value. Default is FALSE.

`year_name` Boolean flag to include year name. Default is FALSE.

`leap_year` Boolean flag to include leap year. Default is FALSE.

`month_val` Boolean flag to include month value. Default is FALSE.

`month_abbr` Boolean flag to include month abbreviation. Default is FALSE.

`month_name` Boolean flag to include month name. Default is FALSE.

`week_val` Boolean flag to include week value. Default is FALSE.

`week_abbr` Boolean flag to include week abbreviation. Default is FALSE.

`week_name` Boolean flag to include week name. Default is FALSE.

`weekday_val` Boolean flag to include weekday value. Default is FALSE.

`weekday_abbr` Boolean flag to include weekday abbreviation. Default is FALSE.

`weekday_name` Boolean flag to include weekday name. Default is FALSE.

`day` Boolean flag to include day value. Default is FALSE.

`day_in_month` Boolean flag to include day in month. Default is FALSE.

`day_in_year` Boolean flag to include day in year. Default is FALSE.

`day_in_year_366` Boolean flag to include day in year (366). Default is FALSE.

`pentad_val` Boolean flag to include pentad value. Default is FALSE.

`pentad_abbr` Boolean flag to include pentad abbreviation. Default is FALSE.

`dekad_val` Boolean flag to include dekad value. Default is FALSE.

`dekad_abbr` Boolean flag to include dekad abbreviation. Default is FALSE.

`quarter_val` Boolean flag to include quarter value. Default is FALSE.

`quarter_abbr` Boolean flag to include quarter abbreviation. Default is FALSE.

`with_year` Boolean flag to include with year. Default is FALSE.

`s_start_month` The start month. Default is 1.

`s_start_day_in_month` The start day in month. Default is 1.

`days_in_month` Boolean flag to include days in month. Default is FALSE.

Returns: None

Method `make_inventory_plot()`**:** Create an inventory plot based on the provided data.

Usage:

```
DataBook$make_inventory_plot(
  data_name,
  date_col,
  station_col = NULL,
  year_col = NULL,
  doy_col = NULL,
  element_cols = NULL,
  add_to_data = FALSE,
  year_doy_plot = FALSE,
  coord_flip = FALSE,
  facet_by = NULL,
  graph_title = "Inventory Plot",
  graph_subtitle = NULL,
  graph_caption = NULL,
  title_size = NULL,
  subtitle_size = NULL,
  caption_size = NULL,
  labelXAxis,
  labelYAxis,
  xSize = NULL,
  ySize = NULL,
  Xangle = NULL,
  Yangle = NULL,
  scale_xdate,
  fromXAxis = NULL,
  toXAxis = NULL,
  byXaxis = NULL,
  date_ylabels,
  legend_position = NULL,
  xlabelsize = NULL,
  ylabelsize = NULL,
  scale = NULL,
  dir = "",
  row_col_number,
  nrow = NULL,
  ncol = NULL,
  key_colours = c("red", "grey"),
  display_rain_days = FALSE,
  facet_xsize = 9,
  facet_ysize = 9,
  facet_xangle = 90,
  facet_yangle = 90,
  scale_ydate = FALSE,
  date_ybreaks,
  step = 1,
 rain_cats = list(breaks = c(0, 0.85, Inf), labels = c("Dry", "Rain"), key_colours =
    c("tan3", "blue"))
)
```

Arguments:

`data_name` The name of the data table containing inventory data.

`date_col` The name of the column representing dates.

`station_col` Optional; the name of the column representing station identifiers.

`year_col` Optional; the name of the column representing years.

`doy_col` Optional; the name of the column representing day of the year.

`element_cols` Optional; a vector of column names representing different elements.

`add_to_data` Optional; a boolean indicating whether to add the plot to the existing data.

`year_doy_plot` Optional; a boolean indicating whether to plot by year and day of year.

`coord_flip` Optional; a boolean indicating whether to flip coordinates.

`facet_by` Optional; the variable to facet the plot by.

`graph_title` The title of the plot.

`graph_subtitle` Optional; the subtitle of the plot.

`graph_caption` Optional; the caption of the plot.

`title_size` Optional; the size of the title text.

`subtitle_size` Optional; the size of the subtitle text.

`caption_size` Optional; the size of the caption text.

`labelXAxis` Optional; the label for the x-axis.

`labelYAxis` Optional; the label for the y-axis.

`xSize` Optional; the size of the x-axis text.

`ySize` Optional; the size of the y-axis text.

`Xangle` Optional; the angle of the x-axis labels.

`Yangle` Optional; the angle of the y-axis labels.

`scale_xdate` Optional; a boolean indicating whether to scale x-axis by date.

`fromXAxis` Optional; the starting point for the x-axis.

`toXAxis` Optional; the ending point for the x-axis.

`byXaxis` Optional; the interval for the x-axis.

`date_ylabels` Optional; the labels for the y-axis dates.

`legend_position` Optional; the position of the legend.

`xlabelsize` Optional; the size of the x-axis label text.

`ylabelsize` Optional; the size of the y-axis label text.

`scale` Optional; scaling factor for the plot.

`dir` Optional; the directory to save the plot.

`row_col_number` The number of rows and columns for the plot layout.

`nrow` Optional; the number of rows for the plot layout.

`ncol` Optional; the number of columns for the plot layout.

`key_colours` A vector of colors for the keys in the plot.

`display_rain_days` Optional; a boolean indicating whether to display rain days.

`facet_xsize` Optional; the size of the x-axis facets.

`facet_ysize` Optional; the size of the y-axis facets.

`facet_xangle` Optional; the angle of the x-axis facet labels.

`facet_yangle` Optional; the angle of the y-axis facet labels.

`scale_ydate` Optional; a boolean indicating whether to scale y-axis by date.

`date_ybreaks` Optional; the breaks for y-axis dates.

`step` Optional; the step for the y-axis.

`rain_cats` A list defining categories for rainfall.

Method `import_NetCDF()`**:** Import NetCDF data and convert it into a data frame.

Usage:

```
DataBook$import_NetCDF(
  nc,
  path,
  name,
  only_data_vars = TRUE,
  keep_raw_time = TRUE,
  include_metadata = TRUE,
  boundary,
  lon_points = NULL,
  lat_points = NULL,
  id_points = NULL,
  show_requested_points = TRUE,
  great_circle_dist = FALSE
)
```

Arguments:

`nc` The NetCDF file object.

`path` Optional; the path to the NetCDF file.

`name` The name for the imported data table.

`only_data_vars` A boolean indicating whether to only include data variables.

`keep_raw_time` A boolean indicating whether to keep raw time data.

`include_metadata` A boolean indicating whether to include metadata.

`boundary` Optional; the boundary for the data.

`lon_points` Optional; specific longitude points to include.

`lat_points` Optional; specific latitude points to include.

`id_points` Optional; identifiers for the points to include.

`show_requested_points` A boolean indicating whether to show requested points.

`great_circle_dist` A boolean indicating whether to calculate great circle distances.

Method `infill_missing_dates()`**:** Infill missing dates in the specified data.

Usage:

```
DataBook$infill_missing_dates(
  data_name,
  date_name,
  factors,
  start_month,
  start_date,
  end_date,
  resort = TRUE
)
```

Arguments:

`data_name` The name of the data table to process.

`date_name` The name of the column representing dates.

`factors` A vector of factors to use for infilling dates.

`start_month` The starting month for date infilling.

`start_date` The starting date for date infilling.

`end_date` The ending date for date infilling.

`resort` A boolean indicating whether to resort the data after infilling.

Method `get_key_names()`**:** Retrieve key names from a specified data table.

Usage:

```
DataBook$get_key_names(
  data_name,
  include_overall = TRUE,
  include,
  exclude,
  include_empty = FALSE,
  as_list = FALSE,
  excluded_items = c()
)
```

Arguments:

`data_name` The name of the data table.

`include_overall` A boolean indicating whether to include overall keys.

`include` A vector of specific keys to include.

`exclude` A vector of specific keys to exclude.

`include_empty` A boolean indicating whether to include empty keys.

`as_list` A boolean indicating whether to return the keys as a list.

`excluded_items` A vector of items to exclude from the results.

Method `remove_key()`**:** Remove a specified key from a data table.

Usage:

```
DataBook$remove_key(data_name, key_name)
```

Arguments:

`data_name` The name of the data table.

`key_name` The name of the key to remove.

Method `add_climdex_indices()`**:** Add climdex indices to a specified data table.

Usage:

```
DataBook$add_climdex_indices(
  data_name,
  climdex_output,
  freq = "annual",
  station,
  year,
  month
)
```

Arguments:

`data_name` The name of the data table.

`climdex_output` The output from climdex calculations.

`freq` A string indicating the frequency of the data ('annual' or 'monthly').

`station` The name of the station column (optional).

`year` The name of the year column.

`month` The name of the month column (optional for monthly frequency).

Method `is_metadata()`**:** Check if a specified string is part of the metadata for a data table.

Usage:

```
DataBook$is_metadata(data_name, str)
```

Arguments:

`data_name` The name of the data table.

`str` The string to check in the metadata.

Method `get_climatic_column_name()`**:** Get the climatic column name from the specified data table.

Usage:

```
DataBook$get_climatic_column_name(data_name, col_name)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the climatic column to retrieve.

Method `merge_data()`**:** Merge new data into the specified data table.

Usage:

```
DataBook$merge_data(
  data_name,
  new_data,
  by = NULL,
  type = "left",
  match = "all"
)
```

Arguments:

`data_name` The name of the existing data table.

`new_data` The new data to be merged.

`by` The column(s) to merge by.

`type` The type of merge (e.g., "left", "right", "inner").

`match` How to handle matches (e.g., "all" or "first").

Method `get_corruption_data_names()`**:** Retrieve names of data tables containing corruption data.

Usage:

```
DataBook$get_corruption_data_names()
```

Returns: A vector of names of corruption data tables.

Method `get_corruption_contract_data_names()`**:** Retrieve names of data tables containing corruption contract level data.

Usage:

```
DataBook$get_corruption_contract_data_names()
```

Returns: A vector of names of corruption contract data tables.

Method `get_database_variable_names()`**:** Get variable names from a database based on a query.

Usage:

```
DataBook$get_database_variable_names(
  query,
  data_name,
  include_overall = TRUE,
  include,
  exclude,
  include_empty = FALSE,
  as_list = FALSE,
  excluded_items = c()
)
```

Arguments:

`query` The SQL query to execute.

`data_name` The name of the data table.

`include_overall` A boolean indicating whether to include overall data.

`include` Additional items to include.

`exclude` Additional items to exclude.

`include_empty` A boolean indicating whether to include empty values.

`as_list` A boolean indicating whether to return results as a list.

`excluded_items` A vector of items to exclude from results.

Returns: A list or vector of variable names from the database.

Method `get_nc_variable_names()`**:** Get variable names from a NetCDF file.

Usage:

```
DataBook$get_nc_variable_names(file = "", as_list = FALSE, ...)
```

Arguments:

`file` The path to the NetCDF file.

`as_list` A boolean indicating whether to return results as a list.

`...` Additional arguments passed to other methods.

Returns: A list or vector of variable names from the NetCDF file.

Method `has_database_connection()`**:** Check if there is an active database connection.

Usage:

```
DataBook$has_database_connection()
```

Returns: A boolean indicating whether a database connection exists.

Method `database_connect()`**:** Establish a connection to a database.

Usage:

```
DataBook$database_connect(dbname, user, host, port, drv = RMySQL::MySQL())
```

Arguments:

`dbname` The name of the database.

`user` The username for database access.

`host` The host address of the database.

`port` The port number for database connection.

`drv` The database driver to use (default is MySQL).

Method `get_database_connection()`: Retrieve the current database connection.

Usage:

```
DataBook$get_database_connection()
```

Returns: The active database connection object.

Method `set_database_connection()`: Set the database connection.

Usage:

```
DataBook$set_database_connection(dbi_connection)
```

Arguments:

`dbi_connection` The database connection object to set.

Method `database_disconnect()`: Disconnect from the database if a connection exists.

Usage:

```
DataBook$database_disconnect()
```

Method `get_db_table_row_count()`: Get the row count of a specified table in the database.

Usage:

```
DataBook$get_db_table_row_count(tableName, query_condition = NULL)
```

Arguments:

`tableName` The name of the table to count rows in.

`query_condition` An optional SQL condition to filter the rows.

Returns: The count of rows in the table.

Method `import_climsoft_metadata()`: Import Climsoft metadata, including stations, elements, and flags.

Usage:

```
DataBook$import_climsoft_metadata(
  import_stations = FALSE,
  import_elements = FALSE,
  import_flags = FALSE
)
```

Arguments:

`import_stations` A boolean indicating whether to import station metadata.

`import_elements` A boolean indicating whether to import element metadata.

`import_flags` A boolean indicating whether to import flag metadata.

Method `import_climsoft_data()`: Imports data from Climsoft observation tables, either initial or final. This function also imports selected stations and elements metadata.

Usage:

```
DataBook$import_climsoft_data(
  tableName,
  station_filter_column,
  stations = c(),
  element_filter_column,
  elements = c(),
  qc_status = -1,
  start_date = NULL,
  end_date = NULL,
  unstack_data = FALSE,
  include_element_id = FALSE,
  include_element_name = FALSE,
  include_acquisition_type = FALSE,
  include_level = FALSE,
  include_entry_form = FALSE,
  include_captured_by = FALSE,
  include_qc_status = FALSE,
  include_qc_log = FALSE,
  include_flag = FALSE,
  import_selected_stations_metadata = FALSE,
  import_selected_elements_metadata = FALSE
)
```

Arguments:

`tableName` The name of the Climsoft observation table.

`station_filter_column` The column name used to filter stations.

`stations` A vector of station identifiers to filter the data.

`element_filter_column` The column name used to filter elements.

`elements` A vector of element identifiers to filter the data.

`qc_status` A numeric status for quality control filtering; default is -1 (no filter).

`start_date` Optional; start date for filtering observations.

`end_date` Optional; end date for filtering observations.

`unstack_data` A boolean indicating whether to unstack the data.

`include_element_id` A boolean indicating whether to include element IDs in the output.

`include_element_name` A boolean indicating whether to include element names in the output.

`include_acquisition_type` A boolean indicating whether to include acquisition type in the output.

`include_level` A boolean indicating whether to include observation level in the output.

`include_entry_form` A boolean indicating whether to include entry form in the output.

`include_captured_by` A boolean indicating whether to include the name of the person who captured the data.

`include_qc_status` A boolean indicating whether to include quality control status in the output.

`include_qc_log` A boolean indicating whether to include the quality control log in the output.

`include_flag` A boolean indicating whether to include flags in the output.

`import_selected_stations_metadata` A boolean indicating whether to import metadata for selected stations.

`import_selected_elements_metadata` A boolean indicating whether to import metadata for selected elements.

Method `import_from_iri()`**:** Import data from an IRI source and process it.

Usage:

```
DataBook$import_from_iri(
  download_from,
  data_file,
  data_frame_name,
  location_data_name,
  path,
  X1,
  X2 = NA,
  Y1,
  Y2 = NA,
  get_area_point = "area"
)
```

Arguments:

`download_from` Source to download data from.

`data_file` Name of the data file to import.

`data_frame_name` Name for the data frame created from the imported data.

`location_data_name` Name for the location data frame created from the imported data.

`path` Path to save the imported data.

`X1` The starting coordinate for the x-axis.

`X2` The ending coordinate for the x-axis (optional).

`Y1` The starting coordinate for the y-axis.

`Y2` The ending coordinate for the y-axis (optional).

`get_area_point` Method to determine area point (default is "area").

Method `export_workspace()`**:** Export the current workspace to a file, including optional components.

Usage:

```
DataBook$export_workspace(
  data_names,
  file,
  include_graphs = TRUE,
  include_models = TRUE,
  include_metadata = TRUE
)
```

Arguments:

`data_names` Names of the data frames to export.

`file` Destination file to save the workspace.

`include_graphs` Whether to include graphs (default is TRUE).

`include_models` Whether to include models (default is TRUE).

`include_metadata` Whether to include metadata (default is TRUE).

Method `set_links()`**:** Set new links in the data structure.

Usage:

```
DataBook$set_links(new_links)
```

Arguments:

`new_links` A list of new links to be set.

Method `display_daily_graph()`**:** Display a daily graph for specified climatic elements.

Usage:

```
DataBook$display_daily_graph(
  data_name,
  date_col = NULL,
  station_col = NULL,
  year_col = NULL,
  doy_col = NULL,
  climatic_element = NULL,
  upper_limit = 100,
  bar_colour = "blue",
  rug_colour = "red"
)
```

Arguments:

`data_name` Name of the data frame containing the data.

`date_col` Column name for the date.

`station_col` Column name for the station (optional).

`year_col` Column name for the year (optional).

`doy_col` Column name for the day of the year (optional).

`climatic_element` Name of the climatic element to display (optional).

`upper_limit` Maximum value for the graph (default is 100).

`bar_colour` Color for the bars in the graph (default is "blue").

`rug_colour` Color for the rug plot (default is "red").

Method `create_variable_set()`**:** Create a set of variables from specified columns in the data frame.

Usage:

```
DataBook$create_variable_set(data_name, set_name, columns)
```

Arguments:

`data_name` Name of the data frame.

`set_name` Name for the new variable set.

`columns` Vector of column names to include in the set.

Method `update_variable_set()`**:** Update an existing variable set with new columns.

Usage:

```
DataBook$update_variable_set(data_name, set_name, columns, new_set_name)
```

Arguments:

`data_name` Name of the data frame.

`set_name` Name of the variable set to update.

`columns` Vector of new column names to include.

`new_set_name` New name for the updated variable set.

Method `delete_variable_sets()`**:** Delete specified variable sets from the data frame.

Usage:

```
DataBook$delete_variable_sets(data_name, set_names)
```

Arguments:

`data_name` Name of the data frame.

`set_names` Vector of names of variable sets to delete.

Method `get_variable_sets_names()`**:** Retrieve the names of variable sets in the data frame.

Usage:

```
DataBook$get_variable_sets_names(
  data_name,
  include_overall = TRUE,
  include,
  exclude,
  include_empty = FALSE,
  as_list = FALSE,
  excluded_items = c()
)
```

Arguments:

`data_name` Name of the data frame.

`include_overall` Whether to include overall set (default is TRUE).

`include` Additional filters for inclusion.

`exclude` Exclusion filters.

`include_empty` Whether to include empty sets (default is FALSE).

`as_list` Whether to return as a list (default is FALSE).

`excluded_items` Items to exclude from the results.

Method `get_variable_sets()`**:** Get specific variable sets from the data frame.

Usage:

```
DataBook$get_variable_sets(data_name, set_names, force_as_list = FALSE)
```

Arguments:

`data_name` Name of the data frame.

`set_names` Names of the variable sets to retrieve.

`force_as_list` Whether to force the result as a list (default is FALSE).

Method `crops_definitions()`**:** Define crop conditions and create a new crop definition data frame.

Usage:

```
DataBook$crops_definitions(
  data_name,
  year,
  station,
  rain,
  day,
  rain_totals,
  plant_days,
  plant_lengths,
  start_check = c("both", "yes", "no"),
  season_data_name,
  start_day,
  end_day,
  return_crops_table = TRUE,
  definition_props = TRUE
)
```

Arguments:

`data_name` Name of the data frame containing the data.

`year` Name of the column representing the year.

`station` Name of the column for the station (optional).

`rain` Name of the column containing rainfall data.

`day` Name of the column containing day data.

`rain_totals` Column name for rain totals.

`plant_days` Column name for planting days.

`plant_lengths` Column name for planting lengths.

`start_check` Boolean indicating whether to check start day (default is TRUE).

`season_data_name` Name of the season data frame (optional).

`start_day` Column name for the start day.

`end_day` Column name for the end day.

`return_crops_table` Boolean indicating whether to return the full crops table (default is TRUE).

`definition_props` Boolean indicating whether to calculate properties (default is TRUE).

`print_table` Boolean indicating whether to print the table (default is TRUE).

Method `tidy_climatic_data()`: Tidy climatic data into a specified format.

Usage:

```
DataBook$tidy_climatic_data(
  x,
  format,
  stack_cols,
  day,
  month,
  year,
  stack_years,
  station,
  element,
  element_name = "value",
  ignore_invalid = FALSE,
  silent = FALSE,
  unstack_elements = TRUE,
  new_name
)
```

Arguments:

`x` The input data frame containing climatic data.

`format` A character string indicating the format of the data; must be either 'days', 'months', or 'years'.

`stack_cols` A vector of column names to be stacked.

`day` Optional; the name of the day column.

`month` Optional; the name of the month column.

`year` Optional; the name of the year column.

`stack_years` Optional; a vector of years corresponding to the stack columns.

`station` Optional; the name of the station column.

`element` Optional; the name of the element column.

`element_name` The name to assign to the value column for the climatic element.

`ignore_invalid` A boolean indicating whether to ignore invalid dates.

`silent` A boolean indicating whether to suppress output messages.

`unstack_elements` A boolean indicating whether to unstack multiple elements.

`new_name` Optional; the name to assign to the resulting tidy data frame.

Method `get_geometry()`**:** Retrieve the geometry column from a given data object.

Usage:

```
DataBook$get_geometry(data)
```

Arguments:

`data` The data object from which to retrieve the geometry column.

Returns: The name of the geometry column if found, otherwise an empty string.

Method `package_check()`**:** Check the installation status and version of a specified package.

Usage:

```
DataBook$package_check(package)
```

Arguments:

`package` The name of the package to check.

Returns: A list containing the status of the package: 1 if installed and up to date, 2 if installed but outdated, and 0 if not installed or misspelled.

Method `download_from_IRI()`**:** Download data from the IRI database based on specified parameters.

Usage:

```
DataBook$download_from_IRI(
  source,
  data,
  path = tempdir(),
  min_lon,
  max_lon,
  min_lat,
  max_lat,
  min_date,
  max_date,
  name,
  download_type = "Point",
  import = TRUE
)
```

Arguments:

`source` The source from which to download data.

`data` The specific data to download.

`path` The directory path for saving the downloaded file.

`min_lon` Minimum longitude for area selection.

`max_lon` Maximum longitude for area selection.

`min_lat` Minimum latitude for area selection.

`max_lat` Maximum latitude for area selection.

`min_date` Minimum date for the data.

`max_date` Maximum date for the data.

`name` The name to assign to the imported data.

`download_type` The type of download (Point or Area).

`import` Boolean indicating whether to import the downloaded data.

Method `patch_climate_element()`**:** Patch a climate element in the specified data.

Usage:

```
DataBook$patch_climate_element(
  data_name,
  date_col_name = "",
  var = "",
  vars = c(),
  max_mean_bias = NA,
  max_stdev_bias = NA,
  time_interval = "month",
  column_name,
  station_col_name = station_col_name
)
```

Arguments:

`data_name` The name of the data to patch.

`date_col_name` The name of the date column.

`var` The variable to patch.

`vars` A vector of variables to patch.

`max_mean_bias` Maximum allowable mean bias for the patching.

`max_stdev_bias` Maximum allowable standard deviation bias for the patching.

`time_interval` The time interval for the patching (default is "month").

`column_name` The name of the column to patch.

`station_col_name` The name of the station column.

Method `visualize_element_na()`**:** Visualize the missing values in a specified element within a dataset.

Usage:

```
DataBook$visualize_element_na(
  data_name,
  element_col_name,
  element_col_name_imputed,
  station_col_name,
  x_axis_labels_col_name,
  ncol = 2,
  type = "distribution",
  xlab = NULL,
  ylab = NULL,
  legend = TRUE,
  orientation = "horizontal",
  interval_size = interval_size,
  x_with_truth = NULL,
  measure = "percent"
)
```

Arguments:

`data_name` The name of the data table.

`element_col_name` The name of the column containing the element of interest.

`element_col_name_imputed` The name of the column for imputed values of the element.

`station_col_name` The name of the column representing the station.

`x_axis_labels_col_name` The name of the column for x-axis labels.

`ncol` The number of columns for visualization layout.

`type` The type of visualization (e.g., distribution).

`xlab` Label for the x-axis.

`ylab` Label for the y-axis.

`legend` Logical indicating whether to include a legend.

`orientation` Orientation of the plot (e.g., horizontal).

`interval_size` Size of intervals for the visualization.

`x_with_truth` Optional truth values for comparison.

`measure` Measurement type (e.g., percent).

Method `get_data_entry_data()`: Retrieve data entry for specified elements within a date range.

Usage:

```
DataBook$get_data_entry_data(
  data_name,
  station,
  date,
  elements,
  view_variables,
  station_name,
  type,
  start_date,
  end_date
)
```

Arguments:

`data_name` The name of the data table.

`station` The station of interest.

`date` The date of interest.

`elements` The elements to retrieve.

`view_variables` Variables to view in the output.

`station_name` The name of the station.

`type` The type of data to retrieve.

`start_date` The start date of the data range.

`end_date` The end date of the data range.

Method `save_data_entry_data()`: Save new data entries and associated comments to the dataset.

Usage:

```
DataBook$save_data_entry_data(
  data_name,
  new_data,
  rows_changed,
  comments_list = list(),
  add_flags = FALSE,
  ...
)
```

Arguments:

`data_name` The name of the data table.

`new_data` The new data to save.

`rows_changed` The rows that have been modified.

`comments_list` A list of comments for changes made.

`add_flags` Logical indicating whether to add flags.

`...` Additional arguments passed to other methods.

Method `import_from_cds()`: Import data from CDS (Climate Data Store) for specified parameters.

Usage:

```
DataBook$import_from_cds(
  user,
  dataset,
  elements,
  start_date,
  end_date,
  lon,
  lat,
  path,
  import = FALSE,
  new_name
)
```

Arguments:

`user` The user credentials for accessing CDS.

`dataset` The dataset to import.

`elements` The elements to retrieve from the dataset.

`start_date` The starting date for the data.

`end_date` The ending date for the data.

`lon` Longitude for area definition.

`lat` Latitude for area definition.

`path` The path to save the imported data.

`import` Logical indicating whether to import the data.

`new_name` Optional new name for the imported data.

Method `add_flag_fields()`: Add flag fields to a specified dataset.

Usage:

```
DataBook$add_flag_fields(data_name, col_names, key_column_names)
```

Arguments:

`data_name` The name of the data table.

`col_names` The names of the columns to flag.

`key_column_names` The names of key columns.

Method `remove_empty()`: Remove empty rows or columns from a dataset.

Usage:

```
DataBook$remove_empty(data_name, which = c("rows", "cols"))
```

Arguments:

`data_name` The name of the data table.

`which` The option to remove either "rows" or "cols".

Method `replace_values_with_NA()`**:** Replace specified values with NA in a dataset.

Usage:

```
DataBook$replace_values_with_NA(data_name, row_index, column_index)
```

Arguments:

`data_name` The name of the data table.

`row_index` The index of the row to modify.

`column_index` The index of the column to modify.

Method `has_labels()`**:** Check if specified columns in a dataset have labels.

Usage:

```
DataBook$has_labels(data_name, col_names)
```

Arguments:

`data_name` The name of the data table.

`col_names` The names of the columns to check.

Method `wrap_or_unwrap_data()`**:** Wrap or unwrap data in a specified column of a dataset.

Usage:

```
DataBook$wrap_or_unwrap_data(
  data_name,
  col_name,
  column_data,
  width,
  wrap = TRUE
)
```

Arguments:

`data_name` The name of the data table.

`col_name` The name of the column to modify.

`column_data` The data in the column.

`width` The width for wrapping.

`wrap` Logical indicating whether to wrap or unwrap data.

Method `anova_tables2()`**:** Generate an ANOVA table for specified predictor and response variables. Optionally includes totals, significance levels, and means.

Usage:

```
DataBook$anova_tables2(
  data_name,
  x_col_names,
  y_col_name,
  total = TRUE,
  signif.stars = FALSE,
  sign_level = FALSE,
  means = FALSE,
  interaction = FALSE
)
```

Arguments:

`data_name` The name of the data frame to get the columns from.

`x_col_names` Character vector, the names of predictor variables.

`y_col_name` Character, the name of the response variable.

`total` Logical, whether to include a total row in the ANOVA table. Defaults to FALSE.

`signif.stars` Logical, whether to include significance stars. Defaults to FALSE.

`sign_level` Logical, whether to display significance levels. Defaults to FALSE.

`means` Logical, whether to include means or model coefficients. Defaults to FALSE.

`interaction` Logical, whether to include interaction terms for predictors. Defaults to FALSE.

Returns: A formatted ANOVA table with optional additional sections.

Method `display_daily_table()`**:** Display a daily summary table for a specified climatic data element.

Usage:

```
DataBook$display_daily_table(
  data_name,
  climatic_element,
  date_col = date_col,
  year_col = year_col,
  station_col = station_col,
  Misscode,
  Tracecode,
  Zerocode,
  monstats = c("min", "mean", "median", "max", "IQR", "sum")
)
```

Arguments:

`data_name` A character string representing the name of the dataset.

`climatic_element` A vector specifying the climatic elements to be displayed (e.g., temperature, rainfall).

`date_col` The name of the column containing date information. Default is `date_col`.

`year_col` The name of the column containing year information. Default is `year_col`.

`station_col` The name of the column containing station information. If missing, assigns the `Station` column from metadata.

`Misscode` A value representing missing data in the dataset.

`Tracecode` A value representing trace amounts of the climatic element.

`Zerocode` A value representing zero values for the climatic element.

`monstats` A vector of summary statistics to calculate for monthly data. Options include `"min"`, `"mean"`, `"median"`, `"max"`, `"IQR"`, and `"sum"`.

Returns: A data frame displaying the daily summary table for the specified climatic element.

Method `add_comment()`**:** Adds a new `instat_comment` object to the data sheet if the key is defined and valid.

Usage:

```
DataBook$add_comment(new_comment)
```

Arguments:

`new_comment` An `instat_comment` object to be added to the data sheet.

Details: This function first checks if a key is defined and valid for the data sheet. It also verifies that `new_comment` is an `instat_comment` object and that the key columns in `new_comment` are valid keys in the data frame. If the comment ID already exists, a warning is issued and the existing comment is replaced.

Returns: None. This function modifies the data sheet by adding or replacing a comment.

Method `delete_comment()`**:** Deletes a comment from the data sheet based on the comment ID.

Usage:

```
DataBook$delete_comment(comment_id)
```

Arguments:

`comment_id` A character string representing the ID of the comment to be deleted.

Details: If the specified comment ID does not exist in the data sheet, an error is thrown.

Returns: None. This function modifies the data sheet by removing the specified comment.

Method `get_comment_ids()`**:** Retrieves all comment IDs currently stored in the data sheet.

Usage:

```
DataBook$get_comment_ids()
```

Returns: A character vector containing the IDs of all comments in the data sheet.

Method `get_comments_as_data_frame()`**:** Converts all comments in the data sheet to a data frame format for easier inspection and analysis.

Usage:

```
DataBook$get_comments_as_data_frame()
```

Details: This function collects various fields from each comment and returns them in a data frame. The number of replies and attributes for each comment is also included. Currently, nested comments (replies) and additional attributes are not displayed in detail.

Returns: A data frame with columns representing comment ID, key values, column, value, type, comment text, label, calculation, timestamp, number of replies, resolved status, active status, and number of attributes.

Method `define_as_options_by_context()`**:** Define options by context for a specified dataset.

Usage:

```
DataBook$define_as_options_by_context(
  data_name,
  obyc_types = NULL,
  key_columns = NULL
)
```

Arguments:

`data_name` The name of the data table.

`obyc_types` A named list of options by context types.

`key_columns` A vector of key columns relevant to the dataset.

Method `update_links_rename_data_frame()`**:** This function updates all links that reference a data frame with a specified old name, renaming it to a new name.

Usage:

```
DataBook$update_links_rename_data_frame(old_data_name, new_data_name)
```

Arguments:

`old_data_name` The current name of the data frame in links

`new_data_name` The new name to replace the old data frame name in links Update links to rename a column

Method `update_links_rename_column()`**:** This function updates all links referencing a column in a data frame with a specified old column name, renaming it to a new column name.

Usage:

```
DataBook$update_links_rename_column(
  data_name,
  old_column_name,
  new_column_name
)
```

Arguments:

`data_name` The name of the data frame containing the column

`old_column_name` The current name of the column in links

`new_column_name` The new name to replace the old column name in links

Method `add_link()`**:** This function adds a new link between two data frames with the specified link pairs and type. It will check if the link already exists or if the link columns are keys.

Usage:

```
DataBook$add_link(from_data_frame, to_data_frame, link_pairs, type, link_name)
```

Arguments:

`from_data_frame` The name of the originating data frame in the link

`to_data_frame` The name of the target data frame in the link

`link_pairs` A named vector or list representing pairs of columns to link between data frames

`type` The type of the link (e.g., 'one-to-one', 'many-to-one')

`link_name` Optional; a name for the link. If not provided, a default name is assigned

Method `get_link_names()`**:** Retrieves the names of all links involving a specified data frame, with options to include or exclude specific types.

Usage:

```
DataBook$get_link_names(
  data_name,
  include_overall = TRUE,
  include,
  exclude,
  include_empty = FALSE,
  as_list = FALSE
)
```

Arguments:

`data_name` The name of the data frame

`include_overall` Boolean; if TRUE, includes overall links

`include` Optional vector of link names to include

`exclude` Optional vector of link names to exclude

`include_empty` Boolean; if TRUE, includes links with no associated data

`as_list` Boolean; if TRUE, returns a list format

Method `link_exists_from()`**:** Verifies if a link exists from a specific data frame with given link pairs.

Usage:

```
DataBook$link_exists_from(curr_data_frame, link_pairs)
```

Arguments:

`curr_data_frame` The name of the originating data frame

`link_pairs` The link pairs to check for existence

Method `link_exists_between()`**:** This function checks if there is an ordered or unordered link between two specified data frames.

Usage:

```
DataBook$link_exists_between(from_data_frame, to_data_frame, ordered = FALSE)
```

Arguments:

`from_data_frame` The name of the originating data frame

`to_data_frame` The name of the target data frame

`ordered` Boolean; if TRUE, checks for an ordered link

Method `get_link_between()`**:** Retrieves the link definition between two specified data frames.

Usage:

```
DataBook$get_link_between(from_data_frame, to_data_frame, ordered = FALSE)
```

Arguments:

`from_data_frame` The name of the originating data frame

`to_data_frame` The name of the target data frame

`ordered` Boolean; if TRUE, retrieves an ordered link

Method `link_exists_from_by_to()`**:** This function checks if a link exists from `first_data_frame` to `second_data_frame` using the specified `link_pairs` columns.

Usage:

```
DataBook$link_exists_from_by_to(
  first_data_frame,
  link_pairs,
  second_data_frame
)
```

Arguments:

`first_data_frame` Name of the starting data frame.

`link_pairs` Named vector of columns used in the link.

`second_data_frame` Name of the target data frame.

Returns: Boolean indicating whether the specified link exists.

Method `get_linked_to_data_name()`**:** This function returns the names of data frames linked to `from_data_frame`. Optionally, includes `from_data_frame` itself in the output if `include_self` is TRUE. Filters results by `link_cols`, if provided.

Usage:

```
DataBook$get_linked_to_data_name(
  from_data_frame,
  link_cols = c(),
  include_self = FALSE
)
```

Arguments:

`from_data_frame` Name of the source data frame.

`link_cols` Optional column names to filter links.

`include_self` Boolean indicating if `from_data_frame` should be included.

Returns: A character vector of data frame names.

Method `get_linked_to_definition()`: This function returns a list of the target data frame and matched columns.

Usage:

```
DataBook$get_linked_to_definition(from_data_frame, link_pairs)
```

Arguments:

`from_data_frame` Name of the source data frame.

`link_pairs` Named vector of link columns.

Returns: List with the target data frame name and matching column names.

Method `get_possible_linked_to_definition()`: This function attempts to find a linked data frame that matches `link_pairs`. Recursively explores links between multiple data frames.

Usage:

```
DataBook$get_possible_linked_to_definition(from_data_frame, link_pairs)
```

Arguments:

`from_data_frame` Name of the starting data frame.

`link_pairs` Named vector of columns used in the link.

Returns: List with the name and columns of a matching linked data frame, or an empty list.

Method `get_equivalent_columns()`: This function returns columns in `to_data_name` equivalent to `columns` in `from_data_name`. Recursively searches links between multiple data frames.

Usage:

```
DataBook$get_equivalent_columns(from_data_name, columns, to_data_name)
```

Arguments:

`from_data_name` Name of the source data frame.

`columns` Columns to be matched.

`to_data_name` Name of the target data frame.

Returns: Character vector of equivalent column names in `to_data_name`, or an empty vector.

Method `link_between_containing()`: This function returns columns in `to_data_frame` corresponding to `containing_columns` in `from_data_frame` if a link exists between them.

Usage:

```
DataBook$link_between_containing(
  from_data_frame,
  containing_columns,
  to_data_frame
)
```

Arguments:

`from_data_frame` Name of the source data frame.

`containing_columns` Columns to search for in the link.

`to_data_frame` Name of the target data frame.

Returns: Character vector of columns in `to_data_frame` if a matching link is found, otherwise an empty vector.

Method `view_link()`**:** Displays the details of a specified link.

Usage:

```
DataBook$view_link(link_name)
```

Arguments:

`link_name` The name of the link to view

Method `apply_calculation()`**:** This method applies a given calculation to the data stored in the `DataBook` object. It supports various calculation types (e.g., "summary") and includes options for storing and returning results.

Usage:

```
DataBook$apply_calculation(calc)
```

Arguments:

`calc` A calculation object specifying the type of calculation and its parameters. For a "summary" calculation, parameters should include: - `data_name`: The name of the data object to apply the calculation to. - `columns_to_summarise`: Columns to include in the summary. - `summaries`: The summary operations to perform. - `store_results`: Whether to store the results in the `DataBook`. - `return_output`: Whether to return the summary output.

Returns: If `return_output` = `TRUE`, returns the calculation results as a data frame; otherwise, returns `NULL`.

Method `save_calculation()`**:** This method saves a calculation to a specific data frame within the `DataBook` object. The calculation is stored in the designated data frame's calculation registry for future reference and reuse.

Usage:

```
DataBook$save_calculation(end_data_frame, calc)
```

Arguments:

`end_data_frame` A string specifying the name of the data frame where the calculation should be saved.

`calc` A calculation object or list that defines the calculation to be saved. This object should include relevant parameters and metadata for the calculation.

Details:

- This method retrieves the `end_data_frame` from the `DataBook` object and invokes its `save_calculation` method to store the calculation.
- The `calc` object typically includes details such as its `name`, `type`, and any parameters or dependencies required to perform the calculation.
- See also `DataSheet$save_calculation`

Returns: None. The method performs the operation in-place, saving the calculation to the specified data frame.

Method `apply_instat_calculation()`**:** This method performs a calculation or series of calculations (including sub-calculations) on data within the `DataBook` object. It supports recursive calls for managing dependencies between manipulations and sub-calculations. This method is called recursively, and it would not be called by a user, another function would always handle the output and display results to the user (usually only the $data part of the list)

Usage:

```
DataBook$apply_instat_calculation(
  calc,
  curr_data_list,
  previous_manipulations = list(),
  param_list = list()
)
```

Arguments:

`calc` A calculation object

`curr_data_list` A list of data objects currently being used. Optional.

`previous_manipulations` A list of previously applied manipulations, used recursively. Default is `list()`.

`param_list` A list of additional parameters for the calculation. Default is `list()`.

Details:

- **Manipulations**: Applied sequentially, with the output of one manipulation passed to the next.
- **Sub-Calculations**: Performed independently, with their outputs combined or merged as needed.
- **Recursive Behavior**: The method is called recursively for handling dependencies and grouping.

Returns: A list with four elements:

- `$data`: A data frame containing the output from the calculation, usually not just the output but also other columns at the same "level"
- `$link`: A link used to determine which data frame the output should be saved in.
- `$has_summary`: Logical, whether a summary was performed.
- `$has_filter`: Logical, whether a filter was applied.

Method `run_instat_calculation()`**:** This method runs a specified calculation using `apply_instat_calculation` and displays the results if required. It serves as the primary interface for triggering calculations within the `DataBook`.

Usage:

```
DataBook$run_instat_calculation(calc, display = TRUE, param_list = list())
```

Arguments:

`calc` A calculation object to be applied.

`display` Logical, whether to display the calculation output. Default is `TRUE`.

`param_list` A list of parameters to pass to the calculation. Default is `list()`.

Returns: The `data` component of the calculation result if `display = TRUE`, otherwise `NULL`.

Method `get_corresponding_link_columns()`**:** This function identifies corresponding link columns between two data frames within the `DataBook` object. It checks for existing links and maps column names between the two data frames based on their relationship.

Usage:

```
DataBook$get_corresponding_link_columns(
  first_data_frame_name,
  first_data_frame_columns,
  second_data_frame_name
)
```

Arguments:

`first_data_frame_name` A string specifying the name of the first data frame.

`first_data_frame_columns` A vector of column names from the first data frame to be mapped.

`second_data_frame_name` A string specifying the name of the second data frame.

Details:

- If a direct link exists between the two data frames, the corresponding columns are determined based on the `link_columns` of the existing link.
- If no direct link exists, it defaults to a one-to-one mapping of the input columns (`first_data_frame_columns`) in the first data frame.

Returns: A named vector where the names represent the columns in the first data frame and the values represent the corresponding columns in the second data frame.

Method `get_link_columns_from_data_frames()`**:** This function finds a link between two data frames within the `DataBook` object and returns the corresponding columns to use for linking. It ensures the link is valid by checking that the columns exist in both data frames.

Usage:

```
DataBook$get_link_columns_from_data_frames(
  first_data_frame_name,
  first_data_frame_columns,
  second_data_frame_name,
  second_data_frame_columns
)
```

Arguments:

`first_data_frame_name` A string specifying the name of the first data frame.

`first_data_frame_columns` A vector of column names from the first data frame to be linked.

`second_data_frame_name` A string specifying the name of the second data frame.

`second_data_frame_columns` A vector of column names from the second data frame to be linked.

Details:

- If a direct link exists between the data frames, the function checks for valid `link_columns` in the `existing_link` object.
- A link is established if all columns in the specified `link_columns` exist in their respective data frames.
- If no valid link is found, an empty vector is returned.

Returns: A named vector where the names represent the columns in the first data frame and the values represent the corresponding columns in the second data frame.

Method `save_calc_output()`**:** This method saves the output of a calculation to the appropriate data frame within the `DataBook` object. It manages links and metadata associated with the calculation.

Usage:

```
DataBook$save_calc_output(calc, curr_data_list, previous_manipulations)
```

Arguments:

`calc` The calculation object.

`curr_data_list` The list of data objects containing the calculation output.

`previous_manipulations` A list of previous manipulations applied to the data.

Details:

- If the output data has a summary or filter applied, appropriate links are created or updated.

- Metadata is added to indicate that the column or data frame is the result of a calculation.
- Dependencies between columns are updated based on the calculation.

Returns: None.

Method `append_summaries_to_data_object():` This method appends the results of a summary calculation to a data object in the `DataBook`. If a corresponding summary data object exists, the method merges the new summary into it. Otherwise, it creates a new summary data object.

Usage:

```
DataBook$append_summaries_to_data_object(
  out,
  data_name,
  columns_to_summarise,
  summaries,
  factors = c(),
  summary_name,
  calc,
  calc_name = ""
)
```

Arguments:

`out` A data frame containing the summary calculation results.

`data_name` A string specifying the name of the data object to which the summaries relate.

`columns_to_summarise` A character vector of columns included in the summary.

`summaries` A character vector of summary operations performed (e.g., `"mean"`, `"sum"`).

`factors` A character vector of grouping factors used in the summary. Default is `c()`.

`summary_name` A string specifying the name of the summary data object. Default is generated dynamically.

`calc` The calculation object containing metadata about the calculation.

`calc_name` Optional. The name of the calculation. Default is an empty string.

Details:

- If a summary data object with the specified `factors` already exists, this method merges the new summary into it.
- If no such data object exists, it creates a new one and links it to the original data object via the specified `factors`.
- Metadata is updated to track dependencies and indicate calculated columns.

Returns: None. The operation is performed in place.

Method `calculate_summary():` Computes summary statistics for a dataset based on specified columns, summaries, and grouping factors. Supports flexible percentage calculations, handling of missing values, and result storage.

Usage:

```
DataBook$calculate_summary(
  data_name,
  columns_to_summarise = NULL,
  summaries,
  factors = c(),
  store_results = TRUE,
  drop = TRUE,
  return_output = FALSE,
  summary_name = NA,
  result_names = NULL,
  percentage_type = "none",
  perc_total_columns = NULL,
  perc_total_factors = c(),
  perc_total_filter = NULL,
  perc_decimal = FALSE,
  perc_return_all = FALSE,
  include_counts_with_percentage = FALSE,
  silent = FALSE,
  additional_filter,
  original_level = FALSE,
  signif_fig = 2,
  sep = "_",
  ...
)
```

Arguments:

`data_name` A character string representing the name of the dataset to summarize.

`columns_to_summarise` Optional. A character vector of column names to summarize. Defaults to `NULL`.

`summaries` A vector of summary functions to apply to the data.

`factors` A character vector of factor column names for grouping. Defaults to an empty vector.

`store_results` Logical. If `TRUE`, stores intermediate results. Defaults to `TRUE`.

`drop` Logical. If `TRUE`, drops unused factor levels. Defaults to `TRUE`.

`return_output` Logical. If `TRUE`, returns the summary output. Defaults to `FALSE`.

`summary_name` A character string for naming the summary. Defaults to `NA`.

`result_names` Optional. A character vector for naming summary results. Defaults to `NULL`.

`percentage_type` Character. Type of percentages to calculate ("none", "factors", "columns", "filter"). Defaults to `"none"`.

`perc_total_columns` Optional. Columns to use for total percentage calculations. Defaults to `NULL`.

`perc_total_factors` A character vector of factors to use for total percentage calculations. Defaults to an empty vector.

`perc_total_filter` Optional. A filter condition for percentage calculations. Defaults to `NULL`.

`perc_decimal` Logical. If `TRUE`, displays percentages in decimal format. Defaults to `FALSE`.

`perc_return_all` Logical. If `TRUE`, returns all percentage-related columns. Defaults to `FALSE`.

`include_counts_with_percentage` Logical. If `TRUE`, includes counts alongside percentages. Defaults to `FALSE`.

`silent` Logical. If `TRUE`, suppresses warnings. Defaults to `FALSE`.

`additional_filter` Optional. Additional filtering conditions for the calculation.

`original_level` Logical. If `TRUE`, uses the original level for calculations. Defaults to `FALSE`.

`signif_fig` Numeric. Number of significant figures for rounding numeric values. Defaults to `2`.

`sep` Character. Separator used in result names. Defaults to `"_"`.

`...` Additional arguments passed to other methods.

Returns: A data frame containing the calculated summary statistics.

Method `summary()`**:** Computes summary statistics for specified columns in a dataset, optionally grouped by factors. Handles multiple summaries, data types, and error conditions gracefully.

Usage:

```
DataBook$summary(
  data_name,
  columns_to_summarise,
  summaries,
  factors = c(),
  store_results = FALSE,
  drop = FALSE,
  return_output = FALSE,
  summary_name = NA,
  add_cols = c(),
  filter_names = c(),
  ...
)
```

Arguments:

`data_name` A character string representing the name of the dataset to summarize.

`columns_to_summarise` A character vector of column names to summarize.

`summaries` A vector of summary function names to apply to the columns.

`factors` A character vector of factor column names for grouping. Defaults to an empty vector.

`store_results` Logical. If `TRUE`, stores the summary results. Defaults to `FALSE`.

`drop` Logical. If `TRUE`, drops unused factor levels. Defaults to `FALSE`.

`return_output` Logical. If `TRUE`, returns the summary output. Defaults to `FALSE`.

`summary_name` Optional. A character string to name the summary. Defaults to `NA`.

`add_cols` Optional. Additional columns to include in the output. Defaults to an empty vector.

`filter_names` A character vector of filter names to apply during the calculation. Defaults to an empty vector.

`...` Additional arguments passed to other methods or functions.

Returns: A data frame or list containing the computed summary statistics. If no grouping factors are provided, the result is a table with row names corresponding to the summary functions.
Convert Linked Variable to Matching Class
This function converts the variables in the linked "to data frame" to match the class of the corresponding variables in the "from data frame".

Method `convert_linked_variable()`**:**

Usage:

```
DataBook$convert_linked_variable(from_data_frame, link_cols)
```

Arguments:

`from_data_frame` A character string specifying the name of the source data frame.

`link_cols` A character vector specifying the columns that define the link between the data frames.

Returns: No explicit return value. The function modifies the linked data frame in place. Remove Unused Station-Year Combinations
This function removes station-year combinations that are not used in the linked data.

Method `remove_unused_station_year_combinations()`**:**

Usage:

```
DataBook$remove_unused_station_year_combinations(data_name, year, station)
```

Arguments:

`data_name` A character string specifying the name of the data frame.

`year` A character string specifying the column name representing the year.

`station` A character string specifying the column name representing the station.

Returns: No explicit return value. The function modifies the linked data frame in place.

Method `summary_table()`**:** Creates a summary table for a dataset based on specified columns, summaries, and factors. Provides options for margins, percentages, and various customization settings.

Usage:

```
DataBook$summary_table(
  data_name,
  columns_to_summarise = NULL,
  summaries,
  factors = c(),
  store_table = FALSE,
  store_results = FALSE,
  drop = TRUE,
  na.rm = FALSE,
  summary_name = NA,
  include_margins = FALSE,
  margins = "outer",
  return_output = FALSE,
  treat_columns_as_factor = FALSE,
  page_by = NULL,
  signif_fig = 2,
  na_display = "",
  na_level_display = "NA",
  weights = NULL,
  caption = NULL,
  result_names = NULL,
  percentage_type = "none",
  perc_total_columns = NULL,
  perc_total_factors = c(),
  perc_total_filter = NULL,
  perc_decimal = FALSE,
  include_counts_with_percentage = FALSE,
  margin_name = "(All)",
  additional_filter,
  ...
)
```

Arguments:

`data_name` A character string representing the name of the dataset to summarize.

`columns_to_summarise` Optional. A character vector of column names to summarize. Defaults to `NULL`.

`summaries` A vector of summary functions to apply to the data.

`factors` A character vector of factor column names for grouping. Defaults to an empty vector.

`store_table` Logical. If `TRUE`, stores the resulting table in the data book. Defaults to `FALSE`.

`store_results` Logical. If `TRUE`, stores intermediate results. Defaults to `FALSE`.

`drop` Logical. If `TRUE`, drops unused factor levels. Defaults to `TRUE`.

`na.rm` Logical. If `TRUE`, removes missing values. Defaults to `FALSE`.

`summary_name` A character string for naming the summary. Defaults to `NA`.

`include_margins` Logical. If `TRUE`, includes margin summaries. Defaults to `FALSE`.

`margins` Character. Type of margins to include ("outer", "summary"). Defaults to `"outer"`.

`return_output` Logical. If `TRUE`, returns the summary output. Defaults to `FALSE`.

`treat_columns_as_factor` Logical. If `TRUE`, treats columns to summarize as factors. Defaults to `FALSE`.

`page_by` Optional. A character vector for paginating results. Defaults to `NULL`.

`signif_fig` Numeric. Number of significant figures for rounding numeric values. Defaults to `2`.

`na_display` Character. String to represent missing values in the output. Defaults to an empty string.

`na_level_display` Character. String to represent missing factor levels in the output. Must be non-empty.

`weights` Optional. A numeric vector of weights for weighted summaries. Defaults to `NULL`.

`caption` Optional. A character string for table captions. Defaults to `NULL`.

`result_names` Optional. A character vector for naming summary results. Defaults to `NULL`.

`percentage_type` Character. Type of percentages to calculate ("none", "row", "column", etc.). Defaults to `"none"`.

`perc_total_columns` Optional. Columns to use for total percentage calculations. Defaults to `NULL`.

`perc_total_factors` A character vector of factors to use for total percentage calculations. Defaults to an empty vector.

`perc_total_filter` Optional. A filter condition for percentage calculations. Defaults to `NULL`.

`perc_decimal` Logical. If `TRUE`, displays percentages in decimal format. Defaults to `FALSE`.

`include_counts_with_percentage` Logical. If `TRUE`, includes counts alongside percentages. Defaults to `FALSE`.

`margin_name` Character. Name for margin rows/columns in the output. Defaults to `"(All)"`.

`additional_filter` Optional. An additional filter for data summarization.

`...` Additional arguments passed to other methods.

Returns: A `tibble` containing the summarized data table.

Method `import_SST()`**:** Imports SST data and adds keys and links to the specified data tables.

Usage:

```
DataBook$import_SST(dataset, data_from = 5, data_names = c())
```

Arguments:

`dataset` The SST dataset.

`data_from` The source of the data. Default is 5.

`data_names` A vector of data table names.

Returns: None

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

This method delegates the actual saving of the calculation to the respective data frame's `save_calculation` method, ensuring modularity and separation of concerns.

If the two data frames are not directly linked, the function assumes the columns in the first data frame map directly to columns with the same names in the second data frame.

This function ensures that the linking columns are valid by verifying their existence in both data frames.

`DataSheet` *DataSheet Class*

Description

An R6 class to handle and manage a data frame with associated metadata, filters, and various settings.

Usage

```
DataSheet$new(data = data.frame(), data_name = "",
                      variables_metadata = data.frame(), metadata = list(),
                      imported_from = "",
                      messages = TRUE, convert = TRUE, create = TRUE,
              start_point = 1, filters = list(), column_selections = list(), objects = list(),
              calculations = list(), scalars = list(), keys = list(), comments = list(), keep_attrib
              undo_history = list(), redo_undo_history = list(), disable_undo = FALSE)
```

Format

An R6 class object.

Methods

`set_data(new_data, messages, check_names)` Sets the data for the DataSheet object.

`set_changes(new_changes)` Sets the changes for the DataSheet object.

`set_filters(new_filters)` Sets the filters for the DataSheet object.

`set_column_selections(new_column_selections)` Sets the column selections for the DataSheet object.

`set_meta(new_meta)` Sets the metadata for the DataSheet object.

`clear_metadata()` Clears the metadata for the DataSheet object.

`clear_variables_metadata()` Clears the variables metadata for the DataSheet object.

`add_defaults_meta()` Adds default metadata to the DataSheet object.

`add_defaults_variables_metadata(column_names)` Adds default variables metadata to the DataSheet object.

`set_objects(new_objects)` Sets the objects for the DataSheet object.

`set_calculations(new_calculations)` Sets the calculations for the DataSheet object.

`set_keys(new_keys)` Sets the keys for the DataSheet object.

`set_comments(new_comments)` Sets the comments for the DataSheet object.

`append_to_metadata(label, value)` Appends to the metadata of the DataSheet object.

`is_metadata(label)` Checks if a label is in the metadata of the DataSheet object.

`set_data_changed(new_val)` Set the data_changed flag.

`set_variables_metadata_changed(new_val)` Set the variables_metadata_changed flag.

`set_metadata_changed(new_val)` Set the metadata_changed flag.

`set_enable_disable_undo(disable_undo)` Set whether undo functionality is enabled or disabled.

`save_state_to_undo_history()` Save the current state to the undo history.

`is_undo()` Check if undo functionality is currently disabled.

`has_undo_history()` Check if there are any actions available to undo.

`undo_last_action()` Undo the last action by restoring the previous state. Removes the last saved state from the undo history.

`redo_last_action()` Redo the last undone action by restoring the next state.

`set_scalars(new_scalars)` Set the scalars property.

`set_undo_history(new_data, attributes = list())` Set the undo history with memory management.

`get_scalars()` Retrieve the current scalars.

`get_scalar_names(as_list = FALSE, excluded_items = c(), ...)` Get the names of the scalars.

`get_scalar_value(scalar_name)` Retrieve the value of a specific scalar by name.

`add_scalar(scalar_name = "", scalar_value)` Add a scalar to the scalars property. Replaces an existing scalar if one with the same name already exists.

`get_data_frame(convert_to_character, include_hidden_columns, use_current_filter, use_column_selec`
Get the data frame with various options for filtering, column selection, and attribute handling.

`get_variables_metadata(data_type, convert_to_character, property, column, error_if_no_property, di`
Get the metadata for the variables in the data frame.

`get_column_data_types(columns)` Get the data types of the specified columns.

`get_column_labels(columns)` Get the labels of the specified columns.

`get_data_frame_label(use_current_filter)` Get the label of the data frame.

`clear_variables_metadata()` Clear the variables metadata.

`get_metadata(label, include_calculated, excluded_not_for_display)` Get the metadata for the data frame.

`get_changes()` Get the changes made to the data frame.

`get_calculations()` Get the calculations applied to the data frame.

`get_calculation_names(as_list, excluded_items)` Get the names of the calculations applied to the data frame.

`add_columns_to_data(col_name, col_data, use_col_name_as_prefix, hidden, before, adjacent_column, nu`
Add new columns to the data frame.

`get_columns_from_data(col_names, force_as_data_frame, use_current_filter, use_column_selection, re`
Get the data for the specified columns.

`anova_tables(x_col_names, y_col_name, signif.stars, sign_level, means)` Generate ANOVA tables for the specified columns.

`cor(x_col_names, y_col_name, use, method)` Calculate the correlation between specified columns.

`rename_column_in_data(curr_col_name, new_col_name, label, type, .fn, .cols, new_column_names_df, nev`
Renames a column in the data.

`remove_columns_in_data(cols, allow_delete_all)` Removes specified columns from the data.

`replace_value_in_data(col_names, rows, old_value, old_is_missing, start_value, end_value, new_value`
Replaces values in the specified columns and rows.

`paste_from_clipboard(col_names, start_row_pos, first_clip_row_is_header, clip_board_text)`
Pastes data from the clipboard into the specified columns and rows.

`append_to_metadata(property, new_value)` Appends a new value to the metadata of the data.

`append_to_variables_metadata(col_names, property, new_val)` Appends a new value to the variables metadata.

`append_to_changes(value)` Appends a value to the changes list.

`is_metadata(str)` Checks if a string is in the metadata.

`is_variables_metadata(str, col, return_vector)` Checks if a string is in the variables metadata.

`add_defaults_meta()` Adds default values to the metadata.

`add_defaults_variables_metadata(column_names)` Adds default values to the variables metadata for the specified columns.

`remove_rows_in_data(row_names)` Removes the specified rows from the data.

`get_next_default_column_name(prefix)` Gets the next default column name based on the given prefix.

`reorder_columns_in_data(col_order)` Reorders the columns in the data based on the given order.

`insert_row_in_data(start_row, row_data, number_rows, before)` Inserts new rows into the data at the specified position.

`get_data_frame_length(use_current_filter)` Gets the length of the data frame.

`get_factor_data_frame(col_name, include_levels, include_NA_level)` Gets the data frame for a factor column with optional inclusion of levels and NA level.

`get_column_factor_levels(col_name)` Gets the factor levels for the specified column.

`sort_dataframe(col_names, decreasing, na.last, by_row_names, row_names_as_numeric)`
Sorts the data frame based on the specified columns.

`convert_column_to_type(col_names, to_type, factor_values, set_digits, set_decimals, keep_attr, ignc`
Converts the specified columns to the given type.

`copy_columns(col_names)` Copies the specified columns in the data.

`drop_unused_factor_levels(col_name)` Drops unused factor levels in the specified column.

`set_factor_levels(col_name, new_labels, new_levels, set_new_labels)` Sets the factor levels for the specified column.

`edit_factor_level(col_name, old_level, new_level)` Edits a factor level in the specified column.

`set_factor_reference_level(col_name, new_ref_level)` Sets the reference level for a factor column.

`reorder_factor_levels(col_name, new_level_names)` Reorders the factor levels for the specified column.

`get_column_count(use_column_selection)` Gets the count of columns in the data frame.

`get_column_names(as_list, include, exclude, excluded_items, max_no, use_current_column_selection)`
Gets the names of the columns in the data frame.

`get_data_type(col_name)` Gets the data type of the specified column.

`set_hidden_columns(col_names)` Sets the specified columns as hidden.

`unhide_all_columns()` Unhides all columns.

`set_row_names(row_names)` Sets the row names of the data.

`set_col_names(col_names)` Sets the column names of the data.

`get_row_names()` Gets the row names of the data.

`get_dim_dataframe()` Gets the dimensions of the data frame.

`set_protected_columns(col_names)` Sets the specified columns as protected.

`add_filter(filter, filter_name, replace, set_as_current, na.rm, is_no_filter, and_or, inner_not, out`
Adds a filter to the data.

`add_filter_as_levels(filter_levels, column)` Adds multiple filters based on the levels of a specified column.

`get_current_filter()` Gets the current filter applied to the data.

`set_current_filter(filter_name)` Sets the current filter for the data.

`get_filter_names(as_list, include, exclude, excluded_items)` Gets the names of all filters.

`get_filter(filter_name)` Gets a specific filter by name.

`get_filter_as_logical(filter_name)` Gets the logical vector of a filter.

`get_filter_column_names(filter_name)` Gets the column names used in a filter.

`get_current_filter_column_names()` Gets the column names used in the current filter.

`filter_applied()` Checks if a filter is applied.

`remove_current_filter()` Remove the current filter.

`filter_string(filter_name)` Returns the string representation of a filter.

`get_filter_as_instat_calculation(filter_name)` Returns the filter as an instat calculation object.

`add_column_selection(column_selection, name, replace, set_as_current, is_everything, and_or)`
Adds a column selection to the data.

`get_current_column_selection()` Gets the current column selection applied to the data.

`set_current_column_selection(name)` Sets the current column selection for the data.

`get_column_selection_names(as_list, include, exclude, excluded_items)` Gets the names of all column selections.

`get_column_selection(name)` Gets a specific column selection by name.

`get_column_selection_column_names(name)` Gets the column names used in a column selection.

`get_column_selected_column_names(column_selection_name)` Gets the selected column names for a given column selection name.

`column_selection_applied()` Checks if a column selection is applied.

`remove_current_column_selection()` Removes the current column selection.

`get_variables_metadata_fields(as_list, include, exclude, excluded_items)` Gets the fields of the variables metadata.

`add_object(object_name, object_type_label, object_format, object)` Adds an object with its metadata to the list of objects.

`get_object_names(object_type_label, as_list)` Gets the names of objects of a specified type.

`get_objects(object_type_label)` Gets objects of a specified type.

`get_object(object_name)` Gets a specific object by name.

`rename_object(object_name, new_name, object_type)` Renames an object.

`delete_objects(data_name, object_names, object_type)` Deletes specified objects.

`reorder_objects(new_order)` Reorders the objects.

`data_clone(include_objects, include_metadata, include_logs, include_filters, include_column_select`
Clones the data with specified attributes included or excluded.

`freeze_columns(column)` Freezes the specified columns.

`unfreeze_columns()` Unfreezes all columns.

`add_key(col_names, key_name)` Adds a key with specified columns.

`is_key(col_names)` Checks if specified columns form a key.

`has_key()` Checks if there is a key in the data.

`get_keys(key_name)` Gets the keys of the data.

`remove_key(key_name)` Removes a specified key.

`get_comments(comment_id)` Gets the comments for the data.

`remove_comment(key_name)` Removes a comment.

`set_structure_columns(struc_type_1, struc_type_2, struc_type_3)` Sets the structure columns of the data.

`add_dependent_columns(columns, dependent_cols)` Adds dependent columns to the specified columns.

`set_column_colours(columns, colours)` Sets the colours of the specified columns.

`has_colours(columns)` Checks if the specified columns have colours.

`set_column_colours_by_metadata(data_name, columns, property)` Sets the colours of columns based on metadata property.

`remove_column_colours()` Removes the colours of the columns.

`graph_one_variable(columns, numeric, categorical, output, free_scale_axis, ncol, coord_flip, ...)`
Creates a graph for a single variable.

`make_date_yearmonthday(year, month, day, f_year, f_month, f_day, year_format, month_format)`
Creates a date from year, month, and day columns.

`make_date_yeardoy(year, doy, base, doy_typical_length)` Creates a date from year and day of year columns.

`set_contrasts_of_factor(col_name, new_contrasts, defined_contr_matrix)` Sets contrasts for a factor column in the data.

`split_date(col_name = "", year_val = FALSE, year_name = FALSE, leap_year = FALSE, month_val = FALSE, mo`
Extracts components such as year, month, week, weekday, etc., from a date column and creates respective new columns.

`set_climatic_types(types)` Sets the climatic types for columns in the data.

`append_climatic_types(types)` Appends climatic types to columns in the data.

`make_inventory_plot(date_col, station_col = NULL, year_col = NULL, doy_col = NULL, element_cols = NULL`
Creates an inventory plot for specified date and element columns.

`infill_missing_dates(date_name, factors, start_month, start_date, end_date, resort = TRUE)`
Infills missing dates in the data for a specified date column, with optional factors, start and end dates.

`get_key_names(include_overall = TRUE, include, exclude, include_empty = FALSE, as_list = FALSE, exclu`
: Retrieves key names from the data, with options to include overall, include or exclude specific keys, and return as a list.

`define_corruption_outputs(output_columns = c())` Defines the specified output columns as corruption outputs and updates metadata accordingly.

`define_red_flags(red_flags = c())` Defines the specified columns as red flags and updates metadata accordingly.

`define_as_procurement_country_level_data(types = c(), auto_generate = TRUE)` Defines the data as procurement country-level data with specified types and optionally auto-generates columns.

`is_corruption_type_present(type)` Checks if the specified corruption type is present in the data.

`get_CRI_component_column_names()` Retrieves the column names that are components of the Corruption Risk Index (CRI).

`get_red_flag_column_names()` Retrieves the column names that are defined as red flags.

`get_CRI_column_names()` Retrieves the column names that start with "CRI".

`get_corruption_column_name(type)` Gets the column name associated with the specified corruption type.

`set_procurement_types(primary_types = c(), calculated_types = c(), auto_generate = TRUE)` Sets the specified primary and calculated procurement types, and optionally auto-generates columns.

`generate_award_year()` Generates and appends the award year column to the data.

`generate_procedure_type()` Generates and appends the procedure type column to the data.

`generate_procuring_authority_id()` Generates and appends the procuring authority ID column to the data.

`generate_winner_id()` Generates and appends the winner ID column to the data.

`generate_foreign_winner()` Generates and appends the foreign winner column to the data.

`generate_procurement_type_categories()` Generates and appends the procurement type categories column to the data.

`generate_procurement_type_2()` Generates and appends the procurement type 2 column to the data.

`generate_procurement_type_3()` Generates and appends the procurement type 3 column to the data.

`generate_signature_period()` Generates and appends the signature period column to the data.

`generate_signature_period_corrected()` Generates and appends the corrected signature period column to the data.

`generate_signature_period_5Q()` Generates and appends the signature period 5 quantiles column to the data.

`generate_signature_period_25Q()` Generates and appends the signature period 25 quantiles column to the data.

`generate_rolling_contract_no_winners()` Generates and appends the rolling contract number of winners column to the data.

`generate_rolling_contract_no_issuer()` Generates and appends the rolling contract number of issuers column to the data.

`generate_rolling_contract_value_sum_issuer()` Generates and appends the rolling contract value sum of issuers column to the data.

`generate_rolling_contract_value_sum_winner()` Generates and appends the rolling contract value sum of winners column to the data.

`generate_rolling_contract_value_share_winner()` Generates and appends the rolling contract value share of winners column to the data.

`generate_single_bidder()` Generates and appends the single bidder column to the data.

`generate_contract_value_share_over_threshold()` Generates and appends the contract value share over threshold column to the data.

`generate_all_bids()` Generates and appends the all bids column to the data.

`generate_all_bids_trimmed()` Generates and appends the all bids trimmed column to the data.

`standardise_country_names(country_columns = c())` Standardises the country names in the specified columns.

`get_climatic_column_name(col_name)` Gets the climatic column name from the data.

`is_climatic_data()` Checks if the data is defined as climatic.

`append_column_attributes(col_name, new_attr)` Appends attributes to the specified column.

`display_daily_graph(data_name, date_col = NULL, station_col = NULL, year_col = NULL, doy_col = NULL, c`
Creates and displays daily graphs for the specified climatic element.

`get_variables_metadata_names(columns)` Gets the names of the metadata attributes for the specified columns.

`create_variable_set(set_name, columns)` Creates a variable set with the specified name and columns.

`update_variable_set(set_name, columns, new_set_name)` Updates the variable set with the specified columns and new set name.

`delete_variable_sets(set_names)` Deletes the specified variable sets.

`get_variable_sets_names(include_overall = TRUE, include, exclude, include_empty = FALSE, as_list = FA`
Gets the names of the variable sets.

`get_variable_sets(set_names, force_as_list)` Gets the specified variable sets.

`patch_climate_element(date_col_name = "", var = "", vars = c(), max_mean_bias = NA, max_stdev_bias = NA`
Patches the specified climate element with the given parameters.

`visualize_element_na(element_col_name, element_col_name_imputed, station_col_name, x_axis_labels_`
Visualizes the NA values in the specified element column with the given parameters.

`get_data_entry_data(station, date, elements, view_variables, station_name, type, start_date, end_da`
Gets the data entry data for the specified parameters.

`save_data_entry_data(new_data, rows_changed, add_flags = FALSE, ...)` Saves the data entry data with the specified parameters.

`add_flag_fields(col_names)` Adds flag fields to the specified columns.

`remove_empty(which = c("rows", "cols"))` Removes empty rows or columns from the data.

`replace_values_with_NA(row_index, column_index)` Replaces values with NA in the specified rows and columns.

`set_options_by_context_types(obyc_types = NULL, key_columns = NULL)` Set options by context types for the current data sheet.

`has_labels(col_names)` Checks if the specified columns have labels.

`display_daily_table(data_name, climatic_element, date_col = date_col, year_col = year_col, station_c`
Display a daily summary table for a specified climatic data element.

`add_comment(new_comment)` Adds a new `instat_comment` object to the data sheet if the key is defined and valid.

`delete_comment(comment_id)` Deletes a comment from the data sheet based on the comment ID.

`get_comment_ids()` Retrieves all comment IDs currently stored in the data sheet.

`get_comments_as_data_frame()` Converts all comments in the data sheet to a data frame format for easier inspection and analysis.

`save_calculation(calc)` Save a Calculation to the DataSheet.

`merge_data(new_data, by = NULL, type = "left", match = "all")` Merge New Data with Existing Data

`calculate_summary(calc, ...)` Calculate Summaries for Specified Columns

`get_column_climatic_type(col_name, attr_name)` Retrieve the climatic type attribute for a specific column.

`update_selection(new_values, column_selection_name = NULL)` Update Column Selection.

`anova_tables2(x_col_names, y_col_name, total = FALSE, signif.stars = FALSE, sign_level = FALSE, means` Generate an ANOVA table for specified predictor and response variables. Optionally includes totals, significance levels, and means.

Active bindings

`data_changed` Logical, indicates if the data has changed. If setting a value, `new_value` must be `TRUE` or `FALSE`.

`metadata_changed` Logical, indicates if the metadata has changed. If setting a value, `new_value` must be `TRUE` or `FALSE`.

`variables_metadata_changed` Logical, indicates if the variables metadata has changed. If setting a value, `new_value` must be `TRUE` or `FALSE`.

`current_filter` A list representing the current filter. If setting a value, `filter` must be a list.

`current_column_selection` A list representing the current column selection. If setting a value, `column_selection` must be a list.

Active bindings

`data_changed` Logical, indicates if the data has changed. If setting a value, new_value must be TRUE or FALSE.

`metadata_changed` Logical, indicates if the metadata has changed. If setting a value, new_value must be TRUE or FALSE.

`variables_metadata_changed` Logical, indicates if the variables metadata has changed. If setting a value, new_value must be TRUE or FALSE.

`current_filter` A list representing the current filter. If setting a value, filter must be a list.

`current_column_selection` A list representing the current column selection. If setting a value, column_selection must be a list.

Methods

Public methods:

- `DataSheet$new()`
- `DataSheet$set_data()`
- `DataSheet$set_meta()`
- `DataSheet$clear_metadata()`
- `DataSheet$set_changes()`

- DataSheet$set_filters()
- DataSheet$set_column_selections()
- DataSheet$set_objects()
- DataSheet$set_calculations()
- DataSheet$set_keys()
- DataSheet$set_comments()
- DataSheet$set_data_changed()
- DataSheet$set_variables_metadata_changed()
- DataSheet$set_metadata_changed()
- DataSheet$set_enable_disable_undo()
- DataSheet$save_state_to_undo_history()
- DataSheet$is_undo()
- DataSheet$has_undo_history()
- DataSheet$undo_last_action()
- DataSheet$redo_last_action()
- DataSheet$set_scalars()
- DataSheet$set_undo_history()
- DataSheet$get_scalars()
- DataSheet$get_scalar_names()
- DataSheet$get_scalar_value()
- DataSheet$add_scalar()
- DataSheet$get_data_frame()
- DataSheet$get_variables_metadata()
- DataSheet$get_column_data_types()
- DataSheet$get_column_labels()
- DataSheet$get_data_frame_label()
- DataSheet$clear_variables_metadata()
- DataSheet$get_metadata()
- DataSheet$get_changes()
- DataSheet$get_calculations()
- DataSheet$get_calculation_names()
- DataSheet$add_columns_to_data()
- DataSheet$get_columns_from_data()
- DataSheet$anova_tables()
- DataSheet$cor()
- DataSheet$rename_column_in_data()
- DataSheet$remove_columns_in_data()
- DataSheet$replace_value_in_data()
- DataSheet$paste_from_clipboard()
- DataSheet$append_to_metadata()
- DataSheet$append_to_variables_metadata()
- DataSheet$append_to_changes()
- DataSheet$is_metadata()
- DataSheet$is_variables_metadata()
- DataSheet$add_defaults_meta()

- DataSheet$add_defaults_variables_metadata()
- DataSheet$remove_rows_in_data()
- DataSheet$get_next_default_column_name()
- DataSheet$reorder_columns_in_data()
- DataSheet$insert_row_in_data()
- DataSheet$get_data_frame_length()
- DataSheet$get_factor_data_frame()
- DataSheet$get_column_factor_levels()
- DataSheet$sort_dataframe()
- DataSheet$convert_column_to_type()
- DataSheet$copy_columns()
- DataSheet$drop_unused_factor_levels()
- DataSheet$set_factor_levels()
- DataSheet$edit_factor_level()
- DataSheet$set_factor_reference_level()
- DataSheet$reorder_factor_levels()
- DataSheet$get_column_count()
- DataSheet$get_column_names()
- DataSheet$get_data_type()
- DataSheet$set_hidden_columns()
- DataSheet$unhide_all_columns()
- DataSheet$set_row_names()
- DataSheet$set_col_names()
- DataSheet$get_row_names()
- DataSheet$get_dim_dataframe()
- DataSheet$set_protected_columns()
- DataSheet$add_filter()
- DataSheet$add_filter_as_levels()
- DataSheet$get_current_filter()
- DataSheet$set_current_filter()
- DataSheet$get_filter_names()
- DataSheet$get_filter()
- DataSheet$get_filter_as_logical()
- DataSheet$get_filter_column_names()
- DataSheet$get_current_filter_column_names()
- DataSheet$filter_applied()
- DataSheet$remove_current_filter()
- DataSheet$filter_string()
- DataSheet$get_filter_as_instat_calculation()
- DataSheet$add_column_selection()
- DataSheet$get_current_column_selection()
- DataSheet$set_current_column_selection()
- DataSheet$get_column_selection_names()
- DataSheet$get_column_selection()
- DataSheet$get_column_selection_column_names()

- DataSheet$get_column_selected_column_names()
- DataSheet$column_selection_applied()
- DataSheet$remove_current_column_selection()
- DataSheet$get_variables_metadata_fields()
- DataSheet$add_object()
- DataSheet$get_object_names()
- DataSheet$get_objects()
- DataSheet$get_object()
- DataSheet$rename_object()
- DataSheet$delete_objects()
- DataSheet$reorder_objects()
- DataSheet$data_clone()
- DataSheet$freeze_columns()
- DataSheet$unfreeze_columns()
- DataSheet$add_key()
- DataSheet$is_key()
- DataSheet$has_key()
- DataSheet$get_keys()
- DataSheet$remove_key()
- DataSheet$get_comments()
- DataSheet$remove_comment()
- DataSheet$set_structure_columns()
- DataSheet$add_dependent_columns()
- DataSheet$set_column_colours()
- DataSheet$has_colours()
- DataSheet$set_column_colours_by_metadata()
- DataSheet$remove_column_colours()
- DataSheet$graph_one_variable()
- DataSheet$make_date_yearmonthday()
- DataSheet$make_date_yeardoy()
- DataSheet$set_contrasts_of_factor()
- DataSheet$split_date()
- DataSheet$set_climatic_types()
- DataSheet$append_climatic_types()
- DataSheet$make_inventory_plot()
- DataSheet$infill_missing_dates()
- DataSheet$get_key_names()
- DataSheet$define_corruption_outputs()
- DataSheet$define_red_flags()
- DataSheet$define_as_procurement_country_level_data()
- DataSheet$is_corruption_type_present()
- DataSheet$get_CRI_component_column_names()
- DataSheet$get_red_flag_column_names()
- DataSheet$get_CRI_column_names()
- DataSheet$get_corruption_column_name()

- `DataSheet$set_procurement_types()`
- `DataSheet$generate_award_year()`
- `DataSheet$generate_procedure_type()`
- `DataSheet$generate_procuring_authority_id()`
- `DataSheet$generate_winner_id()`
- `DataSheet$generate_foreign_winner()`
- `DataSheet$generate_procurement_type_categories()`
- `DataSheet$generate_procurement_type_2()`
- `DataSheet$generate_procurement_type_3()`
- `DataSheet$generate_signature_period()`
- `DataSheet$generate_signature_period_corrected()`
- `DataSheet$generate_signature_period_5Q()`
- `DataSheet$generate_signature_period_25Q()`
- `DataSheet$generate_rolling_contract_no_winners()`
- `DataSheet$generate_rolling_contract_no_issuer()`
- `DataSheet$generate_rolling_contract_value_sum_issuer()`
- `DataSheet$generate_rolling_contract_value_sum_winner()`
- `DataSheet$generate_rolling_contract_value_share_winner()`
- `DataSheet$generate_single_bidder()`
- `DataSheet$generate_contract_value_share_over_threshold()`
- `DataSheet$generate_all_bids()`
- `DataSheet$generate_all_bids_trimmed()`
- `DataSheet$standardise_country_names()`
- `DataSheet$get_climatic_column_name()`
- `DataSheet$is_climatic_data()`
- `DataSheet$append_column_attributes()`
- `DataSheet$display_daily_graph()`
- `DataSheet$get_variables_metadata_names()`
- `DataSheet$create_variable_set()`
- `DataSheet$update_variable_set()`
- `DataSheet$delete_variable_sets()`
- `DataSheet$get_variable_sets_names()`
- `DataSheet$get_variable_sets()`
- `DataSheet$patch_climate_element()`
- `DataSheet$visualize_element_na()`
- `DataSheet$get_data_entry_data()`
- `DataSheet$save_data_entry_data()`
- `DataSheet$add_flag_fields()`
- `DataSheet$remove_empty()`
- `DataSheet$replace_values_with_NA()`
- `DataSheet$set_options_by_context_types()`
- `DataSheet$has_labels()`
- `DataSheet$add_comment()`
- `DataSheet$delete_comment()`
- `DataSheet$get_comment_ids()`

- DataSheet$get_comments_as_data_frame()
- DataSheet$save_calculation()
- DataSheet$merge_data()
- DataSheet$calculate_summary()
- DataSheet$get_column_climatic_type()
- DataSheet$update_selection()
- DataSheet$anova_tables2()
- DataSheet$display_daily_table()
- DataSheet$clone()

Method `new()`**:** Create a new DataSheet object.

Usage:

```
DataSheet$new(
  data = data.frame(),
  data_name = "",
  variables_metadata = data.frame(),
  metadata = list(),
  imported_from = "",
  messages = TRUE,
  convert = TRUE,
  create = TRUE,
  start_point = 1,
  filters = list(),
  column_selections = list(),
  objects = list(),
  calculations = list(),
  scalars = list(),
  keys = list(),
  comments = list(),
  keep_attributes = TRUE,
  undo_history = list(),
  redo_undo_history = list(),
  disable_undo = FALSE
)
```

Arguments:

`data` A data frame to be managed by the DataSheet object. Default is an empty data frame.

`data_name` A character string for the name of the data set. Default is an empty string.

`variables_metadata` A data frame containing metadata for the variables. Default is an empty data frame.

`metadata` A list containing additional metadata. Default is an empty list.

`imported_from` A character string indicating the source of the data import. Default is an empty string.

`messages` Logical, if TRUE messages will be shown during the setup. Default is TRUE.

`convert` Logical, if TRUE data will be converted. Default is TRUE.

`create` Logical, if TRUE the data will be created. Default is TRUE.

`start_point` Numeric, the starting point for default naming. Default is 1.

`filters` A list of filters to be applied to the data. Default is an empty list.

`column_selections` A list of column selections. Default is an empty list.

`objects` A list of objects associated with the data. Default is an empty list.

`calculations` A list of calculations to be performed on the data. Default is an empty list.

`scalars` A list of scalars on the data. Default is an empty list.

`keys` A list of keys for the data. Default is an empty list.

`comments` A list of comments associated with the data. Default is an empty list.

`keep_attributes` Logical, if TRUE attributes will be kept. Default is TRUE.

`undo_history` A list containing a history of data frames which will be replaced with the current data frame if the user presses undo.

`redo_undo_history` A list containing the undo history to redo.

`disable_undo` Logical, if TRUE undo option is disabled. Default is FALSE.

Returns: A new `DataSheet` object.

Method `set_data()`**:** Sets the data for the DataSheet object. Accepts various data types and converts them to a data frame.

Usage:

```
DataSheet$set_data(new_data, messages = TRUE, check_names = TRUE)
```

Arguments:

`new_data` The new data to be set. It can be a matrix, tibble, data.table, ts object, array, or vector.

`messages` Logical, if TRUE, messages will be shown during the data setup. Default is TRUE.

`check_names` Logical, if TRUE, column names will be checked and made valid if necessary. Default is TRUE.

Returns: The DataSheet object with the updated data.

Method `set_meta()`**:** Sets the metadata for the DataSheet object.

Usage:

```
DataSheet$set_meta(new_meta)
```

Arguments:

`new_meta` A list containing the new metadata.

Method `clear_metadata()`**:** Clears the metadata for the DataSheet object.

Usage:

```
DataSheet$clear_metadata()
```

Method `set_changes()`**:** Sets the changes for the DataSheet object.

Usage:

```
DataSheet$set_changes(new_changes)
```

Arguments:

`new_changes` A list containing the new changes.

Method `set_filters()`**:** Sets the filters for the DataSheet object.

Usage:

```
DataSheet$set_filters(new_filters)
```

Arguments:

`new_filters` A list containing the new filters.

Method `set_column_selections()`**:** Sets the column selections for the DataSheet object.

Usage:

```
DataSheet$set_column_selections(new_column_selections)
```

Arguments:

`new_column_selections` A list containing the new column selections.

Method `set_objects()`: Sets the objects for the DataSheet object.

Usage:

```
DataSheet$set_objects(new_objects)
```

Arguments:

`new_objects` A list containing the new objects.

Method `set_calculations()`: Sets the calculations for the DataSheet object.

Usage:

```
DataSheet$set_calculations(new_calculations)
```

Arguments:

`new_calculations` A list containing the new calculations.

Method `set_keys()`: Sets the keys for the DataSheet object.

Usage:

```
DataSheet$set_keys(new_keys)
```

Arguments:

`new_keys` A list containing the new keys.

Method `set_comments()`: Sets the comments for the DataSheet object.

Usage:

```
DataSheet$set_comments(new_comments)
```

Arguments:

`new_comments` A list containing the new comments.

Method `set_data_changed()`: Set the data_changed flag.

Usage:

```
DataSheet$set_data_changed(new_val)
```

Arguments:

`new_val` Logical, new value for the data_changed flag.

Method `set_variables_metadata_changed()`: Set the variables_metadata_changed flag.

Usage:

```
DataSheet$set_variables_metadata_changed(new_val)
```

Arguments:

`new_val` Logical, new value for the variables_metadata_changed flag.

Method `set_metadata_changed()`: Set the metadata_changed flag.

Usage:

```
DataSheet$set_metadata_changed(new_val)
```

Arguments:

`new_val` Logical, new value for the metadata_changed flag.

Method `set_enable_disable_undo()`**:** Set whether undo functionality is enabled or disabled.

Usage:

```
DataSheet$set_enable_disable_undo(disable_undo)
```

Arguments:

`disable_undo` Logical, whether to disable the undo functionality.

Method `save_state_to_undo_history()`**:** Save the current state to the undo history.

Usage:

```
DataSheet$save_state_to_undo_history()
```

Method `is_undo()`**:** Check if undo functionality is currently disabled.

Usage:

```
DataSheet$is_undo()
```

Returns: Logical, TRUE if undo functionality is disabled, otherwise FALSE.

Method `has_undo_history()`**:** Check if there are any actions available to undo.

Usage:

```
DataSheet$has_undo_history()
```

Returns: Logical, TRUE if undo history is available, otherwise FALSE.

Method `undo_last_action()`**:** Undo the last action by restoring the previous state. Removes the last saved state from the undo history.

Usage:

```
DataSheet$undo_last_action()
```

Method `redo_last_action()`**:** Redo the last undone action by restoring the next state. Moves the restored state back to the undo history.

Usage:

```
DataSheet$redo_last_action()
```

Method `set_scalars()`**:** Set the scalars property.

Usage:

```
DataSheet$set_scalars(new_scalars)
```

Arguments:

`new_scalars` List, the new scalars to set.

Method `set_undo_history()`**:** Set the undo history with memory management. Ensures undo history size and memory usage are within defined limits.

Usage:

```
DataSheet$set_undo_history(new_data, attributes = list())
```

Arguments:

`new_data` Data frame, the new data to store in undo history.

`attributes` List, attributes associated with the data.

Method `get_scalars()`**:** Retrieve the current scalars.

Usage:

```
DataSheet$get_scalars()
```

Returns: List, the scalars currently stored.

Method `get_scalar_names()`**:** Get the names of the scalars.

Usage:

```
DataSheet$get_scalar_names(as_list = FALSE, excluded_items = c(), ...)
```

Arguments:

`as_list` Logical, whether to return the names as a list. Defaults to FALSE.

`excluded_items` Character vector, items to exclude from the result. Defaults to an empty vector.

`...` Additional arguments for customization.

Returns: Character vector or list, the names of the scalars.

Method `get_scalar_value()`**:** Retrieve the value of a specific scalar by name.

Usage:

```
DataSheet$get_scalar_value(scalar_name)
```

Arguments:

`scalar_name` Character, the name of the scalar to retrieve.

Returns: The value of the specified scalar.

Method `add_scalar()`**:** Add a scalar to the scalars property. Replaces an existing scalar if one with the same name already exists.

Usage:

```
DataSheet$add_scalar(scalar_name = "", scalar_value)
```

Arguments:

`scalar_name` Character, the name of the scalar. Defaults to the next available name.

`scalar_value` The value of the scalar.

Method `get_data_frame()`**:** Get the data frame with various options for filtering, column selection, and attribute handling.

Usage:

```
DataSheet$get_data_frame(
  convert_to_character = FALSE,
  include_hidden_columns = TRUE,
  use_current_filter = TRUE,
  use_column_selection = TRUE,
  filter_name = "",
  column_selection_name = "",
  stack_data = FALSE,
  remove_attr = FALSE,
  retain_attr = FALSE,
  max_cols,
  max_rows,
  drop_unused_filter_levels = FALSE,
  start_row,
  start_col,
  ...
)
```

Arguments:

`convert_to_character` Logical, if TRUE converts the data to character format.

`include_hidden_columns` Logical, if TRUE includes hidden columns in the output.

`use_current_filter` Logical, if TRUE uses the current filter applied to the data.

`use_column_selection` Logical, if TRUE uses the current column selection.

`filter_name` Character, specifies the name of the filter to use.

`column_selection_name` Character, specifies the name of the column selection to use.

`stack_data` Logical, if TRUE stacks the data.

`remove_attr` Logical, if TRUE removes certain attributes from the data.

`retain_attr` Logical, if TRUE retains certain attributes in the data.

`max_cols` Numeric, specifies the maximum number of columns to include in the output.

`max_rows` Numeric, specifies the maximum number of rows to include in the output.

`drop_unused_filter_levels` Logical, if TRUE drops unused levels from factors in the filtered data.

`start_row` Numeric, specifies the starting row for the output.

`start_col` Numeric, specifies the starting column for the output.

`...` Additional arguments passed to internal functions.

Returns: A data frame with the specified options applied.

Method `get_variables_metadata()`**:** Get the metadata for the variables in the data frame.

Usage:

```
DataSheet$get_variables_metadata(
  data_type = "all",
  convert_to_character = FALSE,
  property,
  column,
  error_if_no_property = TRUE,
  direct_from_attributes = FALSE,
  use_column_selection = TRUE
)
```

Arguments:

`data_type` Character, the data type to filter by. Default is "all".

`convert_to_character` Logical, if TRUE converts the metadata to character format.

`property` Character, the property of the metadata to retrieve.

`column` Character, the column to retrieve metadata for.

`error_if_no_property` Logical, if TRUE throws an error if the property is not found.

`direct_from_attributes` Logical, if TRUE retrieves metadata directly from attributes.

`use_column_selection` Logical, if TRUE uses the current column selection.

Returns: A data frame or list of metadata for the variables.

Method `get_column_data_types()`**:** Get the data types of the specified columns.

Usage:

```
DataSheet$get_column_data_types(columns)
```

Arguments:

`columns` Character vector, names of the columns to get data types for.

Returns: A character vector of data types for the specified columns.

Method `get_column_labels()`**:** Get the labels of the specified columns.

Usage:

```
DataSheet$get_column_labels(columns)
```

Arguments:

`columns` Character vector, names of the columns to get labels for.

Returns: A character vector of labels for the specified columns.

Method `get_data_frame_label()`**:** Get the label of the data frame.

Usage:

```
DataSheet$get_data_frame_label(use_current_filter = FALSE)
```

Arguments:

`use_current_filter` Logical, if TRUE uses the current filter applied to the data.

Returns: A character string representing the label of the data frame.

Method `clear_variables_metadata()`**:** Clear the variables metadata.

Usage:

```
DataSheet$clear_variables_metadata()
```

Method `get_metadata()`**:** Get the metadata for the data frame.

Usage:

```
DataSheet$get_metadata(
  label,
  include_calculated = TRUE,
  excluded_not_for_display = TRUE
)
```

Arguments:

`label` Character, specifies the metadata label to retrieve.

`include_calculated` Logical, if TRUE includes calculated metadata.

`excluded_not_for_display` Logical, if TRUE excludes metadata not for display.

Returns: A list of metadata for the data frame.

Method `get_changes()`**:** Get the changes made to the data frame.

Usage:

```
DataSheet$get_changes()
```

Returns: A list of changes made to the data frame.

Method `get_calculations()`**:** Get the calculations applied to the data frame.

Usage:

```
DataSheet$get_calculations()
```

Returns: A list of calculations applied to the data frame.

Method `get_calculation_names()`**:** Get the names of the calculations applied to the data frame.

Usage:

```
DataSheet$get_calculation_names(as_list = FALSE, excluded_items = c())
```

Arguments:

`as_list` Logical, if TRUE returns the names as a list.

`excluded_items` Character vector, names of calculations to exclude.

Returns: A character vector or list of calculation names.

Method `add_columns_to_data()`**:** Add new columns to the data frame.

Usage:

```
DataSheet$add_columns_to_data(
  col_name = "",
  col_data,
  use_col_name_as_prefix = FALSE,
  hidden = FALSE,
  before,
  adjacent_column = "",
  num_cols,
  require_correct_length = TRUE,
  keep_existing_position = TRUE
)
```

Arguments:

`col_name` Character, name of the new column.

`col_data` Data, the data for the new column.

`use_col_name_as_prefix` Logical, if TRUE uses the column name as a prefix.

`hidden` Logical, if TRUE the new column will be hidden.

`before` Logical, if TRUE adds the new column before the specified adjacent column.

`adjacent_column` Character, name of the adjacent column.

`num_cols` Numeric, number of columns to add.

`require_correct_length` Logical, if TRUE requires the new column to have the correct length.

`keep_existing_position` Logical, if TRUE keeps the existing position of the new column.

Returns: The updated data frame with the new columns added.

Method `get_columns_from_data()`**:** Get the data for the specified columns.

Usage:

```
DataSheet$get_columns_from_data(
  col_names,
  force_as_data_frame = FALSE,
  use_current_filter = TRUE,
  use_column_selection = TRUE,
  remove_labels = FALSE,
  drop_unused_filter_levels = FALSE
)
```

Arguments:

`col_names` Character vector, names of the columns to retrieve.

`force_as_data_frame` Logical, if TRUE forces the output to be a data frame.

`use_current_filter` Logical, if TRUE uses the current filter applied to the data.

`use_column_selection` Logical, if TRUE uses the current column selection.

`remove_labels` Logical, if TRUE removes labels from the data.

`drop_unused_filter_levels` Logical, if TRUE drops unused levels from factors in the filtered data.

Returns: A data frame or vector of the specified columns.

Method `anova_tables()`**:** Generate ANOVA tables for the specified columns.

Usage:

```
DataSheet$anova_tables(
  x_col_names,
  y_col_name,
  signif.stars = FALSE,
  sign_level = FALSE,
  means = FALSE
)
```

Arguments:

`x_col_names` Character vector, names of the columns to use as independent variables.

`y_col_name` Character, name of the dependent variable column.

`signif.stars` Logical, if TRUE includes significance stars in the output.

`sign_level` Logical, if TRUE includes significance levels in the output.

`means` Logical, if TRUE includes means in the output.

Method `cor()`**:** Calculate the correlation between specified columns.

Usage:

```
DataSheet$cor(
  x_col_names,
  y_col_name,
  use = "everything",
  method = c("pearson", "kendall", "spearman")
)
```

Arguments:

`x_col_names` Character vector, names of the columns to use as independent variables.

`y_col_name` Character, name of the dependent variable column.

`use` Character, specifies the handling of missing values. Default is "everything".

`method` Character vector, specifies the correlation method to be used. One of "pearson", "kendall", or "spearman". Default is c("pearson", "kendall", "spearman").

Returns: A matrix of correlation coefficients between the specified columns.

Method `rename_column_in_data()`**:** Rename a column in the data.

Usage:

```
DataSheet$rename_column_in_data(
  curr_col_name = "",
  new_col_name = "",
  label = "",
  type = "single",
  .fn,
  .cols = everything(),
  new_column_names_df,
  new_labels_df,
  ...
)
```

Arguments:

`curr_col_name` Character, the current name of the column.

`new_col_name` Character, the new name for the column.

`label` Character, the label for the column.
`type` Character, the type of renaming to perform.
`.fn` Function, the function to use for renaming.
`.cols` Character, the columns to rename.
`new_column_names_df` Data frame, the new column names.
`new_labels_df` Data frame, the new labels for the columns.
`...` Additional arguments passed to the function.

Method `remove_columns_in_data()`**:** Remove specified columns from the data.

Usage:

```
DataSheet$remove_columns_in_data(cols = c(), allow_delete_all = FALSE)
```

Arguments:

`cols` Character vector, the names of the columns to remove.
`allow_delete_all` Logical, if TRUE, allows deleting all columns.

Method `replace_value_in_data()`**:** Replace values in the specified columns and rows.

Usage:

```
DataSheet$replace_value_in_data(
  col_names,
  rows,
  old_value,
  old_is_missing = FALSE,
  start_value = NA,
  end_value = NA,
  new_value,
  new_is_missing = FALSE,
  closed_start_value = TRUE,
  closed_end_value = TRUE,
  locf = FALSE,
  from_last = FALSE
)
```

Arguments:

`col_names` Character vector, the names of the columns.
`rows` Character vector, the names of the rows.
`old_value` The old value to be replaced.
`old_is_missing` Logical, if TRUE, treats old_value as missing.
`start_value` Numeric, the starting value for the range to replace.
`end_value` Numeric, the ending value for the range to replace.
`new_value` The new value to replace with.
`new_is_missing` Logical, if TRUE, treats new_value as missing.
`closed_start_value` Logical, if TRUE, includes the start value in the range.
`closed_end_value` Logical, if TRUE, includes the end value in the range.
`locf` Logical, if TRUE, uses the last observation carried forward method.
`from_last` Logical, if TRUE, uses the last observation from the end.

Method `paste_from_clipboard()`**:** Paste data from the clipboard into the specified columns and rows.

Usage:

```
DataSheet$paste_from_clipboard(
  col_names,
  start_row_pos = 1,
  first_clip_row_is_header = FALSE,
  clip_board_text
)
```

Arguments:

`col_names` Character vector, the names of the columns.

`start_row_pos` Numeric, the starting row position.

`first_clip_row_is_header` Logical, if TRUE, treats the first row of the clipboard data as a header.

`clip_board_text` Character, the clipboard text data.

Method `append_to_metadata()`**:** Append a new value to the metadata of the data.

Usage:

```
DataSheet$append_to_metadata(property, new_value = "")
```

Arguments:

`property` Character, the property to append to.

`new_value` The new value to append.

Method `append_to_variables_metadata()`**:** Append a new value to the variables metadata.

Usage:

```
DataSheet$append_to_variables_metadata(col_names, property, new_val = "")
```

Arguments:

`col_names` Character vector, the names of the columns.

`property` Character, the property to append to.

`new_val` The new value to append.

Method `append_to_changes()`**:** Append a value to the changes list.

Usage:

```
DataSheet$append_to_changes(value)
```

Arguments:

`value` The value to append.

Method `is_metadata()`**:** Check if a string is in the metadata.

Usage:

```
DataSheet$is_metadata(str)
```

Arguments:

`str` Character, the string to check.

Returns: Logical, TRUE if the string is in the metadata, FALSE otherwise.

Method `is_variables_metadata()`**:** Check if a string is in the variables metadata.

Usage:

```
DataSheet$is_variables_metadata(str, col, return_vector = FALSE)
```

Arguments:

`str` Character, the string to check.

`col` Character, the column to check in.

`return_vector` Logical, if TRUE, returns the result as a vector.

Returns: Logical, TRUE if the string is in the variables metadata, FALSE otherwise.

Method `add_defaults_meta()`**:** Adds default values to the metadata.

Usage:

```
DataSheet$add_defaults_meta()
```

Method `add_defaults_variables_metadata()`**:** Adds default values to the variables metadata for the specified columns.

Usage:

```
DataSheet$add_defaults_variables_metadata(column_names)
```

Arguments:

`column_names` Character vector, the names of the columns.

Method `remove_rows_in_data()`**:** Removes the specified rows from the data.

Usage:

```
DataSheet$remove_rows_in_data(row_names)
```

Arguments:

`row_names` Character vector, the names of the rows to remove.

Method `get_next_default_column_name()`**:** Gets the next default column name based on the given prefix.

Usage:

```
DataSheet$get_next_default_column_name(prefix)
```

Arguments:

`prefix` Character, the prefix for the new column name.

Returns: Character, the next default column name.

Method `reorder_columns_in_data()`**:** Reorders the columns in the data based on the given order.

Usage:

```
DataSheet$reorder_columns_in_data(col_order)
```

Arguments:

`col_order` Character vector, the new order of the columns.

Method `insert_row_in_data()`**:** Inserts new rows into the data at the specified position.

Usage:

```
DataSheet$insert_row_in_data(
  start_row,
  row_data = c(),
  number_rows = 1,
  before = FALSE
)
```

Arguments:

`start_row` Character, the starting row for the new rows.

`row_data` Data frame, the data for the new rows.

`number_rows` Numeric, the number of new rows to insert.

`before` Logical, if TRUE, inserts the new rows before the specified row.

Method `get_data_frame_length()`**:** Gets the length of the data frame.

Usage:

```
DataSheet$get_data_frame_length(use_current_filter = FALSE)
```

Arguments:

`use_current_filter` Logical, if TRUE, uses the current filter.

Returns: Numeric, the length of the data frame.

Method `get_factor_data_frame()`**:** Gets the data frame for a factor column with optional inclusion of levels and NA level.

Usage:

```
DataSheet$get_factor_data_frame(
  col_name = "",
  include_levels = TRUE,
  include_NA_level = FALSE
)
```

Arguments:

`col_name` Character, the name of the factor column.

`include_levels` Logical, if TRUE, includes the levels of the factor.

`include_NA_level` Logical, if TRUE, includes the NA level.

Returns: Data frame, the data frame for the factor column.

Method `get_column_factor_levels()`**:** Gets the factor levels for the specified column.

Usage:

```
DataSheet$get_column_factor_levels(col_name = "")
```

Arguments:

`col_name` Character, the name of the column.

Returns: Character vector, the factor levels for the column.

Method `sort_dataframe()`**:** Sorts the data frame based on the specified columns.

Usage:

```
DataSheet$sort_dataframe(
  col_names = c(),
  decreasing = FALSE,
  na.last = TRUE,
  by_row_names = FALSE,
  row_names_as_numeric = TRUE
)
```

Arguments:

`col_names` Character vector, the names of the columns to sort by.

`decreasing` Logical, if TRUE, sorts in decreasing order.

`na.last` Logical, if TRUE, places NA values last.

`by_row_names` Logical, if TRUE, sorts by row names.

`row_names_as_numeric` Logical, if TRUE, treats row names as numeric values.

Method `convert_column_to_type()`**:** Converts the specified columns to the given type.

Usage:

```
DataSheet$convert_column_to_type(
  col_names = c(),
  to_type,
  factor_values = NULL,
  set_digits,
  set_decimals = FALSE,
  keep_attr = TRUE,
  ignore_labels = FALSE,
  keep.labels = TRUE
)
```

Arguments:

`col_names` Character vector, the names of the columns.

`to_type` Character, the type to convert to.

`factor_values` Character, the factor values to use for conversion.

`set_digits` Numeric, the number of digits to use for conversion.

`set_decimals` Logical, if TRUE, sets the number of decimals.

`keep_attr` Logical, if TRUE, keeps the attributes of the columns.

`ignore_labels` Logical, if TRUE, ignores labels during conversion.

`keep.labels` Logical, if TRUE, keeps labels during conversion.

Method `copy_columns()`**:** Copies the specified columns in the data.

Usage:

```
DataSheet$copy_columns(col_names = "")
```

Arguments:

`col_names` Character vector, the names of the columns to copy.

Method `drop_unused_factor_levels()`**:** Drops unused factor levels in the specified column.

Usage:

```
DataSheet$drop_unused_factor_levels(col_name)
```

Arguments:

`col_name` Character, the name of the column.

Method `set_factor_levels()`**:** Sets the factor levels for the specified column.

Usage:

```
DataSheet$set_factor_levels(
  col_name,
  new_labels,
  new_levels,
  set_new_labels = TRUE
)
```

Arguments:

`col_name` Character, the name of the column.

`new_labels` Character vector, the new labels for the factor levels.

`new_levels` Character vector, the new levels for the factor.

`set_new_labels` Logical, if TRUE, sets the new labels.

Method `edit_factor_level()`**:** Edits the factor level in the specified column.

Usage:

```
DataSheet$edit_factor_level(col_name, old_level, new_level)
```

Arguments:

`col_name` Character, the name of the column.

`old_level` Character, the old factor level.

`new_level` Character, the new factor level.

Method `set_factor_reference_level()`**:** Sets the reference level for a factor column.

Usage:

```
DataSheet$set_factor_reference_level(col_name, new_ref_level)
```

Arguments:

`col_name` Character, the name of the column.

`new_ref_level` Character, the new reference level.

Method `reorder_factor_levels()`**:** Reorders the factor levels in the specified column.

Usage:

```
DataSheet$reorder_factor_levels(col_name, new_level_names)
```

Arguments:

`col_name` Character, the name of the column.

`new_level_names` Character vector, the new order of the factor levels.

Method `get_column_count()`**:** Gets the number of columns in the data.

Usage:

```
DataSheet$get_column_count(use_column_selection = FALSE)
```

Arguments:

`use_column_selection` Logical, if TRUE, uses the current column selection.

Returns: Numeric, the number of columns in the data.

Method `get_column_names()`**:** Gets the names of the columns in the data.

Usage:

```
DataSheet$get_column_names(
  as_list = FALSE,
  include = list(),
  exclude = list(),
  excluded_items = c(),
  max_no,
  use_current_column_selection = TRUE
)
```

Arguments:

`as_list` Logical, if TRUE, returns the names as a list.

`include` List, the properties to include.

`exclude` List, the properties to exclude.

`excluded_items` Character vector, the items to exclude.

`max_no` Numeric, the maximum number of columns to return.

`use_current_column_selection` Logical, if TRUE, uses the current column selection.

Returns: Character vector or list, the names of the columns in the data.

Method `get_data_type()`**:** Gets the data type of the specified column.

Usage:

```
DataSheet$get_data_type(col_name = "")
```

Arguments:

`col_name` Character, the name of the column.

Returns: Character, the data type of the column.

Method `set_hidden_columns()`**:** Set the hidden columns in the data.

Usage:

```
DataSheet$set_hidden_columns(col_names = c())
```

Arguments:

`col_names` Character vector, the names of the columns to hide.

Method `unhide_all_columns()`**:** Unhide all columns in the data.

Usage:

```
DataSheet$unhide_all_columns()
```

Method `set_row_names()`**:** Set the row names of the data frame.

Usage:

```
DataSheet$set_row_names(row_names)
```

Arguments:

`row_names` Character vector, the new row names.

Method `set_col_names()`**:** Set the column names of the data frame.

Usage:

```
DataSheet$set_col_names(col_names)
```

Arguments:

`col_names` Character vector, the new column names.

Method `get_row_names()`**:** Get the row names of the data frame.

Usage:

```
DataSheet$get_row_names()
```

Returns: Character vector, the row names of the data frame.

Method `get_dim_dataframe()`**:** Get the dimensions of the data frame.

Usage:

```
DataSheet$get_dim_dataframe()
```

Returns: Numeric vector, the dimensions of the data frame.

Method `set_protected_columns()`**:** Set the protected columns in the data.

Usage:

```
DataSheet$set_protected_columns(col_names)
```

Arguments:

`col_names` Character vector, the names of the columns to protect.

Method `add_filter()`**:** Add a filter to the data.

Usage:

```
DataSheet$add_filter(
  filter,
  filter_name = "",
  replace = TRUE,
  set_as_current = FALSE,
  na.rm = TRUE,
  is_no_filter = FALSE,
  and_or = "&",
  inner_not = FALSE,
  outer_not = FALSE
)
```

Arguments:

`filter` List, the filter conditions.

`filter_name` Character, the name of the filter.

`replace` Logical, if TRUE, replaces an existing filter with the same name.

`set_as_current` Logical, if TRUE, sets the filter as the current filter.

`na.rm` Logical, if TRUE, removes NA values.

`is_no_filter` Logical, if TRUE, specifies that no filter is applied.

`and_or` Character, specifies the logical operator for combining conditions.

`inner_not` Logical, if TRUE, applies negation to the inner condition.

`outer_not` Logical, if TRUE, applies negation to the outer condition.

Method `add_filter_as_levels()`**:** Add filters based on levels of a column.

Usage:

```
DataSheet$add_filter_as_levels(filter_levels, column)
```

Arguments:

`filter_levels` Character vector, the levels to create filters for.

`column` Character, the name of the column.

Method `get_current_filter()`**:** Get the current filter.

Usage:

```
DataSheet$get_current_filter()
```

Returns: List, the current filter.

Method `set_current_filter()`**:** Set the current filter by name.

Usage:

```
DataSheet$set_current_filter(filter_name = "")
```

Arguments:

`filter_name` Character, the name of the filter to set as current.

Method `get_filter_names()`**:** Get the names of all filters.

Usage:

```
DataSheet$get_filter_names(
  as_list = FALSE,
  include = list(),
  exclude = list(),
  excluded_items = c()
)
```

Arguments:

`as_list` Logical, if TRUE, returns the names as a list.

`include` List, the properties to include.

`exclude` List, the properties to exclude.

`excluded_items` Character vector, the items to exclude.

Returns: Character vector or list, the names of the filters.

Method `get_filter()`**:** Get a specific filter by name.

Usage:

```
DataSheet$get_filter(filter_name)
```

Arguments:

`filter_name` Character, the name of the filter.

Returns: List, the specified filter.

Method `get_filter_as_logical()`**:** Get the filter as a logical vector.

Usage:

```
DataSheet$get_filter_as_logical(filter_name)
```

Arguments:

`filter_name` Character, the name of the filter.

Returns: Logical vector, the filter applied as a logical vector.

Method `get_filter_column_names()`**:** Get the column names used in a specific filter.

Usage:

```
DataSheet$get_filter_column_names(filter_name)
```

Arguments:

`filter_name` Character, the name of the filter.

Returns: Character vector, the column names used in the filter.

Method `get_current_filter_column_names()`**:** Get the column names used in the current filter.

Usage:

```
DataSheet$get_current_filter_column_names()
```

Returns: Character vector, the column names used in the current filter.

Method `filter_applied()`**:** Check if a filter is applied.

Usage:

```
DataSheet$filter_applied()
```

Returns: Logical, TRUE if a filter is applied, FALSE otherwise.

Method `remove_current_filter()`**:** Remove the current filter.

Usage:

```
DataSheet$remove_current_filter()
```

Method `filter_string()`**:** Get the filter as a string.

Usage:

```
DataSheet$filter_string(filter_name)
```

Arguments:

`filter_name` Character, the name of the filter.

Returns: Character, the filter as a string.

Method `get_filter_as_instat_calculation()`**:** Get the filter as an instat calculation.

Usage:

```
DataSheet$get_filter_as_instat_calculation(filter_name)
```

Arguments:

`filter_name` Character, the name of the filter.

Returns: Instat calculation, the filter as an instat calculation.

Method `add_column_selection()`**:** Add a column selection to the data.

Usage:

```
DataSheet$add_column_selection(
  column_selection,
  name = "",
  replace = TRUE,
  set_as_current = FALSE,
  is_everything = FALSE,
  and_or = "|"
)
```

Arguments:

`column_selection` List, the column selection conditions.

`name` Character, the name of the column selection.

`replace` Logical, if TRUE, replaces an existing column selection with the same name.

`set_as_current` Logical, if TRUE, sets the column selection as the current selection.

`is_everything` Logical, if TRUE, selects all columns.

`and_or` Character, specifies the logical operator for combining conditions.

Method `get_current_column_selection()`**:** Get the current column selection.

Usage:

```
DataSheet$get_current_column_selection()
```

Returns: List, the current column selection.

Method `set_current_column_selection()`**:** Set the current column selection by name.

Usage:

```
DataSheet$set_current_column_selection(name = "")
```

Arguments:

`name` Character, the name of the column selection to set as current.

Method `get_column_selection_names()`**:** Get the names of all column selections.

Usage:

```
DataSheet$get_column_selection_names(
  as_list = FALSE,
  include = list(),
  exclude = list(),
  excluded_items = c()
)
```

Arguments:

`as_list` Logical, if TRUE, returns the names as a list.

`include` List, the properties to include.

`exclude` List, the properties to exclude.

`excluded_items` Character vector, the items to exclude.

Returns: Character vector or list, the names of the column selections.

Method `get_column_selection()`**:** Get a specific column selection by name.

Usage:

```
DataSheet$get_column_selection(name)
```

Arguments:

`name` Character, the name of the column selection.

Returns: List, the specified column selection.

Method `get_column_selection_column_names()`**:** Get the column names selected by a specific column selection.

Usage:

```
DataSheet$get_column_selection_column_names(name)
```

Arguments:

`name` Character, the name of the column selection.

Returns: Character vector, the column names selected by the column selection.

Method `get_column_selected_column_names()`**:** Get the column names selected by the current column selection.

Usage:

```
DataSheet$get_column_selected_column_names(column_selection_name = "")
```

Arguments:

`column_selection_name` Character, the name of the column selection.

Returns: Character vector, the column names selected by the current column selection.

Method `column_selection_applied()`**:** Check if a column selection is applied.

Usage:

```
DataSheet$column_selection_applied()
```

Returns: Logical, TRUE if a column selection is applied, FALSE otherwise.

Method `remove_current_column_selection()`**:** Remove the current column selection.

Usage:

```
DataSheet$remove_current_column_selection()
```

Method `get_variables_metadata_fields()`**:** Get the fields of the variables metadata.

Usage:

```
DataSheet$get_variables_metadata_fields(
  as_list = FALSE,
  include = c(),
  exclude = c(),
  excluded_items = c()
)
```

Arguments:

`as_list` Logical, if TRUE, returns the fields as a list.

`include` Character vector, the fields to include.

`exclude` Character vector, the fields to exclude.

`excluded_items` Character vector, the items to exclude.

Returns: Character vector or list, the fields of the variables metadata.

Method `add_object()`**:** Add an object to the data.

Usage:

```
DataSheet$add_object(object_name, object_type_label, object_format, object)
```

Arguments:

`object_name` Character, the name of the object.

`object_type_label` Character, the type label of the object.

`object_format` Character, the format of the object.

`object` Any, the object to add.

Method `get_object_names()`**:** Get the names of objects.

Usage:

```
DataSheet$get_object_names(object_type_label = NULL, as_list = FALSE)
```

Arguments:

`object_type_label` Character, the type label of the objects to get names for.

`as_list` Logical, if TRUE, returns the names as a list.

Returns: Character vector or list, the names of the objects.

Method `get_objects()`**:** Get objects by type label.

Usage:

```
DataSheet$get_objects(object_type_label = NULL)
```

Arguments:

`object_type_label` Character, the type label of the objects to get.

Returns: List, the objects with the specified type label.

Method `get_object()`**:** Get a specific object by name.

Usage:

```
DataSheet$get_object(object_name)
```

Arguments:

`object_name` Character, the name of the object.

Returns: Any, the specified object.

Method `rename_object()`**:** Rename an object.

Usage:

```
DataSheet$rename_object(object_name, new_name, object_type = "object")
```

Arguments:

`object_name` Character, the current name of the object.

`new_name` Character, the new name for the object.

`object_type` Character, the type of the object.

Method `delete_objects()`**:** Delete objects.

Usage:

```
DataSheet$delete_objects(data_name, object_names, object_type = "object")
```

Arguments:

`data_name` Character, the name of the data.

`object_names` Character vector, the names of the objects to delete.

`object_type` Character, the type of the objects to delete.

Method `reorder_objects()`**:** Reorder objects.

Usage:

```
DataSheet$reorder_objects(new_order)
```

Arguments:

`new_order` Character vector, the new order of the objects.

Method `data_clone()`**:** Clone the data sheet.

Usage:

```
DataSheet$data_clone(
  include_objects = TRUE,
  include_metadata = TRUE,
  include_logs = TRUE,
  include_filters = TRUE,
  include_column_selections = TRUE,
  include_calculations = TRUE,
  include_comments = TRUE,
  ...
)
```

Arguments:

`include_objects` Logical, if TRUE, includes objects in the clone.

`include_metadata` Logical, if TRUE, includes metadata in the clone.

`include_logs` Logical, if TRUE, includes logs in the clone.

`include_filters` Logical, if TRUE, includes filters in the clone.

`include_column_selections` Logical, if TRUE, includes column selections in the clone.

`include_calculations` Logical, if TRUE, includes calculations in the clone.

`include_comments` Logical, if TRUE, includes comments in the clone.

`...` Additional arguments.

Returns: DataSheet, the cloned data sheet.

Method `freeze_columns()`**:** Freeze columns in the data.

Usage:

```
DataSheet$freeze_columns(column)
```

Arguments:

`column` Character, the name of the column to freeze.

Method `unfreeze_columns()`**:** Unfreeze all columns in the data.

Usage:

```
DataSheet$unfreeze_columns()
```

Method `add_key()`**:** Add a key to the data.

Usage:

```
DataSheet$add_key(col_names, key_name)
```

Arguments:

`col_names` Character vector, the names of the columns to use as the key.

`key_name` Character, the name of the key.

Method `is_key()`**:** Check if columns are a key.

Usage:

```
DataSheet$is_key(col_names)
```

Arguments:

`col_names` Character vector, the names of the columns to check.

Returns: Logical, TRUE if the columns are a key, FALSE otherwise.

Method `has_key()`**:** Check if the data has a key.

Usage:

```
DataSheet$has_key()
```

Returns: Logical, TRUE if the data has a key, FALSE otherwise.

Method `get_keys()`**:** Get the keys in the data.

Usage:

```
DataSheet$get_keys(key_name)
```

Arguments:

`key_name` Character, the name of the key to get.

Returns: List, the keys in the data.

Method `remove_key()`**:** Remove a key from the data.

Usage:

```
DataSheet$remove_key(key_name)
```

Arguments:

`key_name` Character, the name of the key to remove.

Method `get_comments()`**:** Get comments in the data.

Usage:

```
DataSheet$get_comments(comment_id)
```

Arguments:

`comment_id` Character, the ID of the comment to get.

Returns: List, the comments in the data.

Method `remove_comment()`**:** Remove a comment from the data.

Usage:

```
DataSheet$remove_comment(key_name)
```

Arguments:

`key_name` Character, the name of the key to remove the comment from.

Method `set_structure_columns()`: Set the structure columns in the data.

Usage:

```
DataSheet$set_structure_columns(struc_type_1, struc_type_2, struc_type_3)
```

Arguments:

`struc_type_1` Character vector, the names of the columns for structure type 1.

`struc_type_2` Character vector, the names of the columns for structure type 2.

`struc_type_3` Character vector, the names of the columns for structure type 3.

Method `add_dependent_columns()`: Add dependent columns to the data.

Usage:

```
DataSheet$add_dependent_columns(columns, dependent_cols)
```

Arguments:

`columns` Character vector, the names of the columns.

`dependent_cols` List, the dependent columns.

Method `set_column_colours()`: Set the colors of the columns in the data.

Usage:

```
DataSheet$set_column_colours(columns, colours)
```

Arguments:

`columns` Character vector, the names of the columns.

`colours` Character vector, the colors to set.

Method `has_colours()`: Check if columns have colors.

Usage:

```
DataSheet$has_colours(columns)
```

Arguments:

`columns` Character vector, the names of the columns.

Returns: Logical, TRUE if the columns have colors, FALSE otherwise.

Method `set_column_colours_by_metadata()`: Set the colors of the columns based on metadata.

Usage:

```
DataSheet$set_column_colours_by_metadata(data_name, columns, property)
```

Arguments:

`data_name` Character, the name of the data.

`columns` Character vector, the names of the columns.

`property` Character, the property to base the colors on.

Method `remove_column_colours()`: Remove the colors from all columns.

Usage:

```
DataSheet$remove_column_colours()
```

Method `graph_one_variable()`: Create a graph for one variable.

Usage:

```
DataSheet$graph_one_variable(
  columns,
  numeric = "geom_boxplot",
  categorical = "geom_bar",
  output = "facets",
  free_scale_axis = FALSE,
  ncol = NULL,
  coord_flip = FALSE,
  ...
)
```

Arguments:

`columns` Character vector, the names of the columns.

`numeric` Character, the geom for numeric columns.

`categorical` Character, the geom for categorical columns.

`output` Character, the output type ("facets", "combine", "single").

`free_scale_axis` Logical, if TRUE, uses a free scale for the axis.

`ncol` Numeric, the number of columns for facets.

`coord_flip` Logical, if TRUE, flips the coordinates.

`...` Additional arguments for the geom functions.

Returns: ggplot2 object, the graph.

Method `make_date_yearmonthday()`**:** Create a date from year, month, and day columns.

Usage:

```
DataSheet$make_date_yearmonthday(
  year,
  month,
  day,
  f_year,
  f_month,
  f_day,
  year_format = "%Y",
  month_format = "%m"
)
```

Arguments:

`year` Character, the name of the year column.

`month` Character, the name of the month column.

`day` Character, the name of the day column.

`f_year` Numeric vector, the year values.

`f_month` Numeric vector, the month values.

`f_day` Numeric vector, the day values.

`year_format` Character, the format of the year.

`month_format` Character, the format of the month.

Returns: Date, the created date.

Method `make_date_yeardoy()`**:** Create a date from year and day-of-year columns.

Usage:

```
DataSheet$make_date_yeardoy(year, doy, base, doy_typical_length = "366")
```

Arguments:

`year` Character, the name of the year column.

`doy` Character, the name of the day-of-year column.

`base` Numeric, the base year.

`doy_typical_length` Character, the typical length of the day-of-year ("365" or "366").

Returns: Date, the created date.

Method `set_contrasts_of_factor()`**:** Set the contrasts for a specified factor column.

Usage:

```
DataSheet$set_contrasts_of_factor(
  col_name,
  new_contrasts,
  defined_contr_matrix
)
```

Arguments:

`col_name` Character, the name of the factor column.

`new_contrasts` Character or matrix, the type of contrasts to set or a user-defined contrast matrix.

`defined_contr_matrix` Matrix, the user-defined contrast matrix if `new_contrasts` is "user_defined".

Returns: None.

Method `split_date()`**:** Split a date column into various components like year, month, day, etc., and create corresponding new columns.

Usage:

```
DataSheet$split_date(
  col_name = "",
  year_val = FALSE,
  year_name = FALSE,
  leap_year = FALSE,
  month_val = FALSE,
  month_abbr = FALSE,
  month_name = FALSE,
  week_val = FALSE,
  week_abbr = FALSE,
  week_name = FALSE,
  weekday_val = FALSE,
  weekday_abbr = FALSE,
  weekday_name = FALSE,
  day = FALSE,
  day_in_month = FALSE,
  day_in_year = FALSE,
  day_in_year_366 = FALSE,
  pentad_val = FALSE,
  pentad_abbr = FALSE,
  dekad_val = FALSE,
  dekad_abbr = FALSE,
  quarter_val = FALSE,
  quarter_abbr = FALSE,
  with_year = FALSE,
  s_start_month = 1,
  s_start_day_in_month = 1,
  days_in_month = FALSE
)
```

Arguments:

`col_name` Character, the name of the date column.

`year_val` Logical, whether to create a year column.

`year_name` Logical, whether to create a year name column.

`leap_year` Logical, whether to create a leap year column.

`month_val` Logical, whether to create a month value column.

`month_abbr` Logical, whether to create a month abbreviation column.

`month_name` Logical, whether to create a month name column.

`week_val` Logical, whether to create a week value column.

`week_abbr` Logical, whether to create a week abbreviation column.

`week_name` Logical, whether to create a week name column.

`weekday_val` Logical, whether to create a weekday value column.

`weekday_abbr` Logical, whether to create a weekday abbreviation column.

`weekday_name` Logical, whether to create a weekday name column.

`day` Logical, whether to create a day column.

`day_in_month` Logical, whether to create a day in month column.

`day_in_year` Logical, whether to create a day in year column.

`day_in_year_366` Logical, whether to create a day in year (366 days) column.

`pentad_val` Logical, whether to create a pentad value column.

`pentad_abbr` Logical, whether to create a pentad abbreviation column.

`dekad_val` Logical, whether to create a dekad value column.

`dekad_abbr` Logical, whether to create a dekad abbreviation column.

`quarter_val` Logical, whether to create a quarter value column.

`quarter_abbr` Logical, whether to create a quarter abbreviation column.

`with_year` Logical, whether to include the year in quarter calculation.

`s_start_month` Numeric, the starting month for shifted year calculation.

`s_start_day_in_month` Numeric, the starting day in month for shifted year calculation.

`days_in_month` Logical, whether to create a days in month column.

Returns: None.

Method `set_climatic_types()`**:** Set the climatic types for columns in the data.

Usage:

```
DataSheet$set_climatic_types(types)
```

Arguments:

`types` Named character vector, a named vector where names are climatic types and values are the corresponding column names in the dataset.

Returns: None.

Method `append_climatic_types()`**:** Append climatic types to columns in the data.

Usage:

```
DataSheet$append_climatic_types(types)
```

Arguments:

`types` Named character vector, a named vector where names are climatic types and values are the corresponding column names in the dataset.

Returns: None.

Method `make_inventory_plot()`**:** Create an inventory plot for a dataset.

Usage:

```
DataSheet$make_inventory_plot(
  date_col,
  station_col = NULL,
  year_col = NULL,
  doy_col = NULL,
  element_cols = NULL,
  add_to_data = FALSE,
  year_doy_plot = FALSE,
  coord_flip = FALSE,
  facet_by = NULL,
  facet_xsize = 9,
  facet_ysize = 9,
  facet_xangle = 90,
  facet_yangle = 90,
  graph_title = "Inventory Plot",
  graph_subtitle = NULL,
  graph_caption = NULL,
  title_size = NULL,
  subtitle_size = NULL,
  caption_size = NULL,
  labelXAxis,
  labelYAxis,
  xSize = NULL,
  ySize = NULL,
  Xangle = NULL,
  Yangle = NULL,
  scale_xdate,
  fromXAxis = NULL,
  toXAxis = NULL,
  byXaxis = NULL,
  date_ylabels,
  legend_position = NULL,
  xlabelsize = NULL,
  ylabelsize = NULL,
  scale = NULL,
  dir = "",
  row_col_number,
  nrow = NULL,
  ncol = NULL,
  scale_ydate = FALSE,
  date_ybreaks,
  step = 1,
  key_colours = c("red", "grey"),
  display_rain_days = FALSE,
 rain_cats = list(breaks = c(0, 0.85, Inf), labels = c("Dry", "Rain"), key_colours =
    c("tan3", "blue"))
)
```

Arguments:

`date_col` Character, the name of the date column.

`station_col` Character, the name of the station column. Default is NULL.

`year_col` Character, the name of the year column. Default is NULL.

`doy_col` Character, the name of the day of year column. Default is NULL.

`element_cols` Character vector, the names of the element columns.

`add_to_data` Logical, whether to add the plot to the data. Default is FALSE.

`year_doy_plot` Logical, whether to plot year vs. day of year. Default is FALSE.

`coord_flip` Logical, whether to flip coordinates. Default is FALSE.

`facet_by` Character, the faceting method. Default is NULL.

`facet_xsize` Numeric, the size of facet x-axis labels. Default is 9.

`facet_ysize` Numeric, the size of facet y-axis labels. Default is 9.

`facet_xangle` Numeric, the angle of facet x-axis labels. Default is 90.

`facet_yangle` Numeric, the angle of facet y-axis labels. Default is 90.

`graph_title` Character, the title of the plot. Default is "Inventory Plot".

`graph_subtitle` Character, the subtitle of the plot. Default is NULL.

`graph_caption` Character, the caption of the plot. Default is NULL.

`title_size` Numeric, the size of the plot title. Default is NULL.

`subtitle_size` Numeric, the size of the plot subtitle. Default is NULL.

`caption_size` Numeric, the size of the plot caption. Default is NULL.

`labelXAxis` Character, the label for the x-axis.

`labelYAxis` Character, the label for the y-axis.

`xSize` Numeric, the size of the x-axis labels. Default is NULL.

`ySize` Numeric, the size of the y-axis labels. Default is NULL.

`Xangle` Numeric, the angle of the x-axis labels. Default is NULL.

`Yangle` Numeric, the angle of the y-axis labels. Default is NULL.

`scale_xdate` Logical, whether to scale the x-axis as dates. Default is NULL.

`fromXAxis` Date, the starting date for the x-axis scale. Default is NULL.

`toXAxis` Date, the ending date for the x-axis scale. Default is NULL.

`byXaxis` Character, the interval for the x-axis scale. Default is NULL.

`date_ylabels` Character, the labels for the y-axis if scaled as dates. Default is NULL.

`legend_position` Character, the position of the legend. Default is NULL.

`xlabelsize` Numeric, the size of the x-axis label. Default is NULL.

`ylabelsize` Numeric, the size of the y-axis label. Default is NULL.

`scale` Character, the scale for faceting. Default is NULL.

`dir` Character, the direction for faceting. Default is "".

`row_col_number` Numeric, the number of rows or columns for faceting. Default is NULL.

`nrow` Numeric, the number of rows for faceting. Default is NULL.

`ncol` Numeric, the number of columns for faceting. Default is NULL.

`scale_ydate` Logical, whether to scale the y-axis as dates. Default is FALSE.

`date_ybreaks` Character, the breaks for the y-axis if scaled as dates. Default is NULL.

`step` Numeric, the step size for date breaks. Default is 1.

`key_colours` Character vector, the colours for the key. Default is c("red", "grey").

`display_rain_days` Logical, whether to display rain days in the plot. Default is FALSE.

`rain_cats` List, the categories for rain days, including breaks, labels, and key colours. Default is list(breaks = c(0, 0.85, Inf), labels = c("Dry", "Rain"), key_colours = c("tan3", "blue")).

Returns: ggplot object, the inventory plot.

Method `infill_missing_dates()`: Infill missing dates in the specified column.

Usage:

```
DataSheet$infill_missing_dates(
  date_name,
  factors,
  start_month,
  start_date,
  end_date,
  resort = TRUE
)
```

Arguments:

`date_name` Character, the name of the date column.

`factors` Character vector, the names of the factor columns.

`start_month` Numeric, the start month for infilling.

`start_date` Date, the start date for infilling.

`end_date` Date, the end date for infilling.

`resort` Logical, if TRUE, sorts the data frame after infilling.

Returns: None

Method `get_key_names()`: Get the names of the keys in the data.

Usage:

```
DataSheet$get_key_names(
  include_overall = TRUE,
  include,
  exclude,
  include_empty = FALSE,
  as_list = FALSE,
  excluded_items = c()
)
```

Arguments:

`include_overall` Logical, if TRUE, includes the overall keys.

`include` Character vector, the names of the keys to include.

`exclude` Character vector, the names of the keys to exclude.

`include_empty` Logical, if TRUE, includes empty keys.

`as_list` Logical, if TRUE, returns the keys as a list.

`excluded_items` Character vector, the items to exclude from the keys.

Returns: A character vector or list with the names of the keys.

Method `define_corruption_outputs()`: Define corruption outputs for the dataset.

Usage:

```
DataSheet$define_corruption_outputs(output_columns = c())
```

Arguments:

`output_columns` Character vector, the names of the output columns.

Returns: None

Method `define_red_flags()`**:** Define red flags for the dataset.

Usage:

```
DataSheet$define_red_flags(red_flags = c())
```

Arguments:

`red_flags` Character vector, the names of the red flag columns.

Returns: None

Method `define_as_procurement_country_level_data()`**:** Define the dataset as procurement country level data.

Usage:

```
DataSheet$define_as_procurement_country_level_data(
  types = c(),
  auto_generate = TRUE
)
```

Arguments:

`types` Named list, the types of procurement data.

`auto_generate` Logical, if TRUE, automatically generates additional data.

Returns: None

Method `is_corruption_type_present()`**:** Check if a corruption type is present in the dataset.

Usage:

```
DataSheet$is_corruption_type_present(type)
```

Arguments:

`type` Character, the corruption type to check.

Returns: Logical, TRUE if the corruption type is present, FALSE otherwise.

Method `get_CRI_component_column_names()`**:** Get the column names for CRI components.

Usage:

```
DataSheet$get_CRI_component_column_names()
```

Returns: A character vector with the names of the CRI component columns.

Method `get_red_flag_column_names()`**:** Get the column names for red flag components.

Usage:

```
DataSheet$get_red_flag_column_names()
```

Returns: A character vector with the names of the red flag columns.

Method `get_CRI_column_names()`**:** Get the column names for CRI.

Usage:

```
DataSheet$get_CRI_column_names()
```

Returns: A character vector with the names of the CRI columns.

Method `get_corruption_column_name()`**:** Get the column name for a specific corruption type.

Usage:

```
DataSheet$get_corruption_column_name(type)
```

Arguments:

`type` Character, the corruption type to check.

Returns: A character string with the column name of the specified corruption type.

Method `set_procurement_types()`**:** Set procurement types for the dataset.

Usage:

```
DataSheet$set_procurement_types(
  primary_types = c(),
  calculated_types = c(),
  auto_generate = TRUE
)
```

Arguments:

`primary_types` Named list, the primary types of procurement data.

`calculated_types` Named list, the calculated types of procurement data.

`auto_generate` Logical, if TRUE, automatically generates additional data.

Returns: None

Method `generate_award_year()`**:** Generate the award year for the dataset.

Usage:

```
DataSheet$generate_award_year()
```

Returns: None

Method `generate_procedure_type()`**:** Generate the procedure type for the dataset.

Usage:

```
DataSheet$generate_procedure_type()
```

Returns: None

Method `generate_procuring_authority_id()`**:** Generate the procuring authority ID for the dataset.

Usage:

```
DataSheet$generate_procuring_authority_id()
```

Returns: None

Method `generate_winner_id()`**:** Generate the winner ID for the dataset.

Usage:

```
DataSheet$generate_winner_id()
```

Returns: None

Method `generate_foreign_winner()`**:** Generate the foreign winner flag for the dataset.

Usage:

```
DataSheet$generate_foreign_winner()
```

Returns: None

Method `generate_procurement_type_categories()`**:** Generate procurement type categories for the dataset.

Usage:

```
DataSheet$generate_procurement_type_categories()
```

Returns: None

Method `generate_procurement_type_2()`: Generate procurement type categories 2 for the dataset.

Usage:

```
DataSheet$generate_procurement_type_2()
```

Returns: None

Method `generate_procurement_type_3()`: Generate procurement type categories 3 for the dataset.

Usage:

```
DataSheet$generate_procurement_type_3()
```

Returns: None

Method `generate_signature_period()`: Generate the signature period for the dataset.

Usage:

```
DataSheet$generate_signature_period()
```

Returns: None

Method `generate_signature_period_corrected()`: Generate the corrected signature period for the dataset.

Usage:

```
DataSheet$generate_signature_period_corrected()
```

Returns: None

Method `generate_signature_period_5Q()`: Generate the signature period quintiles (5 quantiles) for the dataset.

Usage:

```
DataSheet$generate_signature_period_5Q()
```

Returns: None

Method `generate_signature_period_25Q()`: Generate the signature period 25 quantiles for the dataset.

Usage:

```
DataSheet$generate_signature_period_25Q()
```

Returns: None

Method `generate_rolling_contract_no_winners()`: Generate rolling contract number of winners for the dataset.

Usage:

```
DataSheet$generate_rolling_contract_no_winners()
```

Returns: None

Method `generate_rolling_contract_no_issuer()`: Generate rolling contract number of issuers for the dataset.

Usage:

```
DataSheet$generate_rolling_contract_no_issuer()
```

Returns: None

Method `generate_rolling_contract_value_sum_issuer()`**:** Generate rolling contract value sum of issuers for the dataset.

Usage:

```
DataSheet$generate_rolling_contract_value_sum_issuer()
```

Returns: None

Method `generate_rolling_contract_value_sum_winner()`**:** Generate rolling contract value sum of winners for the dataset.

Usage:

```
DataSheet$generate_rolling_contract_value_sum_winner()
```

Returns: None

Method `generate_rolling_contract_value_share_winner()`**:** Generate rolling contract value share of winners for the dataset.

Usage:

```
DataSheet$generate_rolling_contract_value_share_winner()
```

Returns: None

Method `generate_single_bidder()`**:** Generate the single bidder flag for the dataset.

Usage:

```
DataSheet$generate_single_bidder()
```

Returns: None

Method `generate_contract_value_share_over_threshold()`**:** Generate contract value share over threshold for the dataset.

Usage:

```
DataSheet$generate_contract_value_share_over_threshold()
```

Returns: None

Method `generate_all_bids()`**:** Generate the number of all bids for the dataset.

Usage:

```
DataSheet$generate_all_bids()
```

Returns: None

Method `generate_all_bids_trimmed()`**:** Generate the number of all trimmed bids for the dataset.

Usage:

```
DataSheet$generate_all_bids_trimmed()
```

Returns: None

Method `standardise_country_names()`**:** Standardise country names in the specified columns.

Usage:

```
DataSheet$standardise_country_names(country_columns = c())
```

Arguments:

`country_columns` A vector of column names containing country names to be standardised.

Returns: None

Method `get_climatic_column_name()`**:** Get the column name for a specified climatic type.

Usage:

```
DataSheet$get_climatic_column_name(col_name)
```

Arguments:

`col_name` The climatic type to look for.

Returns: The column name corresponding to the climatic type, or NULL if not found.

Method `is_climatic_data()`**:** Check if the data is defined as climatic.

Usage:

```
DataSheet$is_climatic_data()
```

Returns: TRUE if the data is defined as climatic, FALSE otherwise.

Method `append_column_attributes()`**:** Append new attributes to a column.

Usage:

```
DataSheet$append_column_attributes(col_name, new_attr)
```

Arguments:

`col_name` The name of the column.

`new_attr` A named list of new attributes to append.

Returns: None

Method `display_daily_graph()`**:** Display daily graphs for climatic elements.

Usage:

```
DataSheet$display_daily_graph(
  data_name,
  date_col = NULL,
  station_col = NULL,
  year_col = NULL,
  doy_col = NULL,
  climatic_element = NULL,
  rug_colour = "red",
  bar_colour = "blue",
  upper_limit = 100
)
```

Arguments:

`data_name` The name of the data set.

`date_col` The name of the date column.

`station_col` The name of the station column.

`year_col` The name of the year column.

`doy_col` The name of the day of year column.

`climatic_element` The climatic element to plot.

`rug_colour` The color of the rug plot.

`bar_colour` The color of the bar plot.

`upper_limit` The upper limit for the y-axis.

Returns: A list of ggplot objects or a single ggplot object.

Method `get_variables_metadata_names()`**:** Get the names of all metadata variables for specified columns.

Usage:

```
DataSheet$get_variables_metadata_names(columns)
```

Arguments:

`columns` A vector of column names.

Returns: A vector of unique metadata variable names.

Method `create_variable_set()`**:** Create a variable set with a specified name and columns.

Usage:

```
DataSheet$create_variable_set(set_name, columns)
```

Arguments:

`set_name` The name of the variable set.

`columns` A vector of column names to include in the set.

Returns: None

Method `update_variable_set()`**:** Update an existing variable set with new columns or rename it.

Usage:

```
DataSheet$update_variable_set(set_name, columns, new_set_name)
```

Arguments:

`set_name` The name of the existing variable set.

`columns` A vector of new column names to include in the set.

`new_set_name` An optional new name for the variable set.

Returns: None

Method `delete_variable_sets()`**:** Delete specified variable sets.

Usage:

```
DataSheet$delete_variable_sets(set_names)
```

Arguments:

`set_names` A vector of variable set names to delete.

Returns: None

Method `get_variable_sets_names()`**:** Get the names of all variable sets.

Usage:

```
DataSheet$get_variable_sets_names(
  include_overall = TRUE,
  include,
  exclude,
  include_empty = FALSE,
  as_list = FALSE,
  excluded_items = c()
)
```

Arguments:

`include_overall` A logical value indicating whether to include the overall set.

`include` A vector of set names to include.

`exclude` A vector of set names to exclude.

`include_empty` A logical value indicating whether to include empty sets.

`as_list` A logical value indicating whether to return the result as a list.

`excluded_items` A vector of items to exclude.

Returns: A vector or list of variable set names.

Method `get_variable_sets()`**:** Get the columns belonging to specified variable sets.

Usage:

```
DataSheet$get_variable_sets(set_names, force_as_list)
```

Arguments:

`set_names` A vector of variable set names.

`force_as_list` A logical value indicating whether to force the result as a list.

Returns: A list of column names or a single vector of column names.

Method `patch_climate_element()`**:** Patch daily climatic elements in the dataset.

Usage:

```
DataSheet$patch_climate_element(
  date_col_name = "",
  var = "",
  vars = c(),
  max_mean_bias = NA,
  max_stdev_bias = NA,
  column_name,
  station_col_name,
  time_interval = "month"
)
```

Arguments:

`date_col_name` The name of the date column.

`var` The name of the variable to patch.

`vars` A vector of variables to use for patching.

`max_mean_bias` The maximum mean bias allowed.

`max_stdev_bias` The maximum standard deviation bias allowed.

`column_name` The name of the column to store the patched values.

`station_col_name` The name of the station column.

`time_interval` The time interval for patching.

Returns: None

Method `visualize_element_na()`**:** Visualize missing data for a specified element.

Usage:

```
DataSheet$visualize_element_na(
  element_col_name,
  element_col_name_imputed,
  station_col_name,
  x_axis_labels_col_name,
  ncol = 2,
  type = "distribution",
  xlab = NULL,
  ylab = NULL,
  legend = TRUE,
  orientation = "horizontal",
  interval_size = 1461,
  x_with_truth = NULL,
  measure = "percent"
)
```

Arguments:

`element_col_name` The name of the element column with missing data.

`element_col_name_imputed` The name of the element column with imputed data.

`station_col_name` The name of the station column.

`x_axis_labels_col_name` The name of the column for x-axis labels.

`ncol` The number of columns for the plot layout.

`type` The type of plot ("distribution", "gapsize", "interval", or "imputation").

`xlab` The label for the x-axis.

`ylab` The label for the y-axis.

`legend` A logical value indicating whether to include a legend.

`orientation` The orientation of the plot ("horizontal" or "vertical").

`interval_size` The size of the intervals for "interval" type plots.

`x_with_truth` The column with true values for comparison.

`measure` The measure for "interval" type plots ("percent" or "absolute").

Returns: A ggplot object or a list of ggplot objects.

Method `get_data_entry_data()`**:** Get data entry data for a specified range and type.

Usage:

```
DataSheet$get_data_entry_data(
  station,
  date,
  elements,
  view_variables,
  station_name,
  type,
  start_date,
  end_date
)
```

Arguments:

`station` The name of the station column.

`date` The name of the date column.

`elements` The names of the element columns.

`view_variables` Additional variables to view.

`station_name` The name of the station.

`type` The type of data ("day", "month", or "range").

`start_date` The start date for the range.

`end_date` The end date for the range.

Returns: A data frame containing the specified data.

Method `save_data_entry_data()`**:** Save data entry data after making changes.

Usage:

```
DataSheet$save_data_entry_data(new_data, rows_changed, add_flags = FALSE, ...)
```

Arguments:

`new_data` The new data to save.

`rows_changed` The rows that have changed.

`add_flags` A logical value indicating whether to add flag fields.

`...` Additional arguments.

Returns: None

Method `add_flag_fields()`: Add flag fields to specified columns.

Usage:

```
DataSheet$add_flag_fields(col_names)
```

Arguments:

`col_names` A vector of column names to add flag fields to.

Returns: None

Method `remove_empty()`: Remove empty rows or columns from the dataset.

Usage:

```
DataSheet$remove_empty(which = c("rows", "cols"))
```

Arguments:

`which` A character vector indicating whether to remove empty "rows", "cols", or both.

Returns: None

Method `replace_values_with_NA()`: Replace values with NA at specified row and column indices.

Usage:

```
DataSheet$replace_values_with_NA(row_index, column_index)
```

Arguments:

`row_index` A vector of row indices.

`column_index` A vector of column indices.

Returns: None

Method `set_options_by_context_types()`: Set options by context types for the current data sheet.

Usage:

```
DataSheet$set_options_by_context_types(obyc_types = NULL, key_columns = NULL)
```

Arguments:

`obyc_types` A named list of options by context types.

`key_columns` A vector of key columns relevant to the data sheet.

Method `has_labels()`: Check if specified columns have labels.

Usage:

```
DataSheet$has_labels(col_names)
```

Arguments:

`col_names` A vector of column names.

Returns: A logical vector indicating if each column has labels. Add a Comment to Data Sheet

Method `add_comment()`**:** Adds a new `instat_comment` object to the data sheet if the key is defined and valid.

Usage:

```
DataSheet$add_comment(new_comment)
```

Arguments:

`new_comment` An `instat_comment` object to be added to the data sheet.

Details: This function first checks if a key is defined and valid for the data sheet. It also verifies that `new_comment` is an `instat_comment` object and that the key columns in `new_comment` are valid keys in the data frame. If the comment ID already exists, a warning is issued and the existing comment is replaced.

Returns: None. This function modifies the data sheet by adding or replacing a comment. Delete a Comment from Data Sheet

Method `delete_comment()`**:** Deletes a comment from the data sheet based on the comment ID.

Usage:

```
DataSheet$delete_comment(comment_id)
```

Arguments:

`comment_id` A character string representing the ID of the comment to be deleted.

Details: If the specified comment ID does not exist in the data sheet, an error is thrown.

Returns: None. This function modifies the data sheet by removing the specified comment. Get All Comment IDs

Method `get_comment_ids()`**:** Retrieves all comment IDs currently stored in the data sheet.

Usage:

```
DataSheet$get_comment_ids()
```

Returns: A character vector containing the IDs of all comments in the data sheet. Get Comments as Data Frame

Method `get_comments_as_data_frame()`**:** Converts all comments in the data sheet to a data frame format for easier inspection and analysis.

Usage:

```
DataSheet$get_comments_as_data_frame()
```

Details: This function collects various fields from each comment and returns them in a data frame. The number of replies and attributes for each comment is also included. Currently, nested comments (replies) and additional attributes are not displayed in detail.

Returns: A data frame with columns representing comment ID, key values, column, value, type, comment text, label, calculation, timestamp, number of replies, resolved status, active status, and number of attributes. Save a Calculation to the DataSheet

Method `save_calculation()`**:** This method adds or updates a calculation in the `DataSheet` object. If a calculation with the same name already exists, it will be replaced, with a warning issued to the user.

Usage:

```
DataSheet$save_calculation(calc)
```

Arguments:

`calc` A list or object representing the calculation to be saved. This object must contain a `name` field. If the `name` field is empty, a default name will be generated using the `instatExtras::next_default_item` function.

Details:

- If the `calc$name` field is empty, a default name is generated using the `instatExtras::next_default_item()` function, based on the prefix "calc" and the existing calculation names in the `private$calculations` environment.
- If a calculation with the same name already exists in `private$calculations`, it will be replaced, and a warning will be issued to inform the user.
- The calculation is saved in the `private$calculations` list, keyed by its `name`.

Returns: The name of the saved calculation (a character string).

Method `merge_data()`**:** This method merges a new data frame with the existing data in the `DataSheet` object. It supports multiple types of joins (left, right, inner, full) and ensures that the data types of the columns used for merging are aligned.

Usage:

```
DataSheet$merge_data(new_data, by = NULL, type = "left", match = "all")
```

Arguments:

`new_data` A data frame containing the new data to merge with the existing data.

`by` A character vector specifying the columns to join by. If `NULL`, the function will attempt to join by all columns with matching names.

`type` A string specifying the type of join. Options are: - `"left"`: Keeps all rows from the existing data. - `"right"`: Keeps all rows from the new data. - `"full"`: Keeps all rows from both data frames. - `"inner"`: Keeps only rows that match in both data frames.

`match` Reserved for future use. Currently not implemented.

Details:

- The method ensures that the data types of the columns specified in `by` are aligned (e.g., converting factors to numeric if necessary).
- Metadata from the original data is preserved and updated after the merge.
- Column attributes for the `by` columns are restored after the merge. Calculate Summaries for Specified Columns

Returns: None. The merged data is stored internally in the `DataSheet` object.

Method `calculate_summary()`**:** This method computes summary statistics for specified columns in the data, grouping by optional factors. It supports multiple summary functions (e.g., mean, sum) and can handle missing values through the `na.rm` parameter.

Usage:

```
DataSheet$calculate_summary(calc, ...)
```

Arguments:

`calc` A calculation object containing parameters for the summary. The object should include: - `columns_to_summarise`: Columns to compute the summaries for. - `summaries`: Functions to apply (e.g., `"mean"`, `"sum"`). - `factors`: Grouping factors for the summaries. - `drop`: Whether to drop unused factor levels. Default is `FALSE`. - `add_cols`: Additional columns to include in the output. - `na.rm`: Logical, whether to remove missing values in the summaries. Default is `FALSE`. - `filters`: Filters to apply before performing the summaries.

`...` Additional arguments to pass to the summary functions.

Details:

- The method applies the specified summaries to the columns provided in `columns_to_summarise`, grouping by `factors`.
- Filters can be applied to restrict the data before calculating summaries.
- Multiple summaries and columns can be computed in a single call.

Returns: A data frame containing the computed summaries. The output includes the grouping factors and the computed summary statistics.

Method `get_column_climatic_type()`**:** Retrieve the climatic type attribute for a specific column.

Usage:

```
DataSheet$get_column_climatic_type(col_name, attr_name)
```

Arguments:

`col_name` Character, the name of the column to retrieve the attribute for.

`attr_name` Character, the name of the attribute to retrieve.

Returns: The value of the specified attribute, or NULL if not available. Update Column Selection
This function updates the conditions of a specified column selection with new values.

Method `update_selection()`**:**

Usage:

```
DataSheet$update_selection(new_values, column_selection_name = NULL)
```

Arguments:

`new_values` A vector of new values to update the column selection with.

`column_selection_name` A character string specifying the name of the column selection to update.

Returns: No explicit return value. The function updates the column selection object in place.

Method `anova_tables2()`**:** Generate an ANOVA table for specified predictor and response variables. Optionally includes totals, significance levels, and means.

Usage:

```
DataSheet$anova_tables2(
  x_col_names,
  y_col_name,
  total = FALSE,
  signif.stars = FALSE,
  sign_level = FALSE,
  means = FALSE,
  interaction = FALSE
)
```

Arguments:

`x_col_names` Character vector, the names of predictor variables.

`y_col_name` Character, the name of the response variable.

`total` Logical, whether to include a total row in the ANOVA table. Defaults to FALSE.

`signif.stars` Logical, whether to include significance stars. Defaults to FALSE.

`sign_level` Logical, whether to display significance levels. Defaults to FALSE.

`means` Logical, whether to include means or model coefficients. Defaults to FALSE.

`interaction` Logical, whether to include interaction terms for predictors. Defaults to FALSE.

Returns: A formatted ANOVA table with optional additional sections. Display Daily Summary Table

Method `display_daily_table()`**:** Display a daily summary table for a specified climatic data element.

Usage:

```
DataSheet$display_daily_table(
  data_name,
  climatic_element,
  date_col = date_col,
  year_col = year_col,
  station_col = station_col,
  Misscode,
  Tracecode,
  Zerocode,
  monstats = c("min", "mean", "median", "max", "IQR", "sum")
)
```

Arguments:

`data_name` A character string representing the name of the dataset.

`climatic_element` A vector specifying the climatic elements to be displayed (e.g., temperature, rainfall).

`date_col` The name of the column containing date information. Default is `date_col`.

`year_col` The name of the column containing year information. Default is `year_col`.

`station_col` The name of the column containing station information. If missing, assigns the `Station` column from metadata.

`Misscode` A value representing missing data in the dataset.

`Tracecode` A value representing trace amounts of the climatic element.

`Zerocode` A value representing zero values for the climatic element.

`monstats` A vector of summary statistics to calculate for monthly data. Options include `"min"`, `"mean"`, `"median"`, `"max"`, `"IQR"`, and `"sum"`.

Details: This function retrieves the data frame associated with the specified dataset and renames columns to standardise `Date`, `Year`, and `Station` for ease of processing. It then displays a daily summary table using the specified climatic elements, handling missing codes, trace codes, and zero codes as defined. Monthly statistics are calculated based on the `monstats` argument.

Returns: A data frame displaying the daily summary table for the specified climatic element.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

Be cautious when replacing existing calculations, as the new calculation will overwrite the previous one without confirmation. Merge New Data with Existing Data

See Also

`instatExtras::next_default_item`

DisplayDaily *Display Daily Meteorological Data*

Description

Display daily meteorological data in a readable table format. This function was developed by Helen Greatrex as part of the SSC-RCLIM package. It formats daily meteorological data for easy viewing, displaying values for each day of the month and summarising monthly statistics. The function is similar in display format to the INSTAT "DisplayDaily" feature.

Usage

```
DisplayDaily(
  Datain,
  Stations,
  Variables,
  option = 1,
  Years,
  Misscode,
  Tracecode,
  Zerocode,
  Fileout = NA,
  monstats = c("min", "mean", "median", "max", "IQR", "sum")
)
```

Arguments

`Datain`	A data frame containing meteorological data, with one column for each variable, one row for each date, and the following columns (case-sensitive): • `Station`: names or identifiers for stations • EITHER: a `Date` column (yyyy-mm-dd format, `option = 1`) • OR: separate `Year`, `Month`, and `Day` columns (`option = 2`)
`Stations`	A character vector of station names to view (e.g., `"KUND0002"`, `c("Paris", "London")`). If missing, all stations in `Datain` are selected.
`Variables`	A character vector of column names for the variables to display (e.g., `"Rain"`, `"TMin"`, `"RH"`). If missing, all variables except metadata columns are selected.
`option`	An integer (1 or 2) indicating date format: • `1`: `Date` column format (yyyy-mm-dd) • `2`: separate `Year`, `Month`, and `Day` columns.
`Years`	An integer vector of years to view (default is all years in `Datain`).
`Misscode`	Character string representing the display value for missing data (e.g., "-").
`Tracecode`	Character string for trace amounts of rainfall (default "tr").
`Zerocode`	Character string for zero values (default "–").
`Fileout`	Optional file path for saving the output table. If `NA`, results are printed to the console.
`monstats`	A character vector of monthly statistics to compute, options include `"min"`, `"mean"`, `"median"`, `"max"`, `"IQR"`, and `"sum"`.

Details

This function checks for the presence of required columns and renames variables as needed to ensure compatibility with the `Datain` format. Missing data, trace rainfall, and zero values are replaced with user-defined codes. Monthly statistics are computed for each month. The function fills in any missing dates within the specified range for seamless daily reporting.

Value

This function prints a formatted daily table for each specified station, variable, and year. If a file path is specified in `Fileout`, the output is saved to that file.

`EDI` *Calculate Extremal Dependency Index*

Description

Computes the extremal dependency index (EDI) using the `verification::verify` function.

Usage

```
EDI(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "categorical").
`obs.type`	Character. The type of observation (e.g., "categorical").
`...`	Additional arguments passed to `verification::verify`.

Value

The extremal dependency index.

`EDS` *Calculate Extreme Dependency Score*

Description

Computes the extreme dependency score (EDS) using the `verification::verify` function.

Usage

```
EDS(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "categorical").
`obs.type`	Character. The type of observation (e.g., "categorical").
`...`	Additional arguments passed to `verification::verify`.

Value

The extreme dependency score.

`ETS` *Calculate Equitable Threat Score*

Description

Computes the equitable threat score using the `verification::verify` function.

Usage

```
ETS(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The equitable threat score.

`FAR` *Calculate False Alarm Ratio*

Description

Computes the false alarm ratio using the `verification::verify` function.

Usage

```
FAR(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The false alarm ratio.

`find_df_from_calc_from` *Find data frame from calculation list*

Description

Find data frame from calculation list

Usage

```
find_df_from_calc_from(x, column)
```

Arguments

`x`	A list of calculations.
`column`	The column name to search for.

Value

The name of the data frame associated with the column.

`get_summary_calculation_names` *Get Summary Calculation Names*

Description

Generates a set of unique names for summary calculations, based on provided summaries, columns, and filters.

Usage

```
get_summary_calculation_names(
  calc,
  summaries,
  columns_to_summarise,
  calc_filters
)
```

Arguments

`calc`	A calculation object (unused in the current implementation).
`summaries`	A vector of summary function names.
`columns_to_summarise`	A vector of column names to summarize.
`calc_filters`	A list of filter objects applied to the calculations.

Value

A character vector of unique summary calculation names.

GS *Calculate Gerrity Score*

Description

Computes the Gerrity score using the `verification::verify` function.

Usage

```
GS(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The Gerrity score.

HSS *Calculate Heidke Skill Score (HSS)*

Description

Computes the Heidke skill score using the `verification::verify` function.

Usage

```
HSS(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The Heidke skill score.

`instat_comment` *Clone* `instat_comment` *Object*

Description

The `instat_comment` R6 class represents a comment in a data sheet, with various properties including identifiers, key-value pairs, comment details, timestamps, and status flags for resolution and activity. A comment is metadata for a row or cell of a data frame A DataSheet will contain a list of instat_comment objects as part of the metadata for the data frame

Details

Instat Comment Class

The `data_clone` method duplicates the current `instat_comment` object, ensuring any `instat_comment` instances within the `replies` field are recursively cloned. Non-`instat_comment` replies are directly copied without cloning.

Methods

`data_clone(...)` Creates a deep clone of the current `instat_comment` object, including all of its fields and nested `instat_comment` replies.

Public fields

`id` A numeric/character string representing the unique identifier for the comment. This must be unique within a data frame.

`key_values` A character vector storing key-value pairs associated with the comment. This identifies the row the comment is on.

`column` If the comment is on a cell, this is the name of the column of the cell

`value` If the comment is on a cell, this is the value in the cell at the time the comment was created.

`type` The type of comment (`"critical"`, `"warning"`, `"message"`, or `""`).

`comment` A character string for the comment text or message.

`label` A character variable. A label or grouping for the comment e.g. if comments are produced by an operation they may all have the same label. This then allows similar comments to be identified e.g. for editing/deleting

`calculation` A character variable. If the comment was created through a calculation e.g. filtering the data frame, this shows how the calculation done on the data frame time_stamp The date and time (POSIXct, POSIXt) the comment was created, defaulting to the current system time if empty.

replies A list of replies to the comment. A reply could be a comment itself

resolved Logical value indicating if the comment is marked as resolved (TRUE or FALSE).

active Logical value indicating if the comment is marked as active (TRUE or FALSE).

attributes A named list of additional information about the comment.

Methods

Public methods:

- instat_comment$new()
- instat_comment$data_clone()
- instat_comment$clone()

Method new()**:** Create a new instat_comment object. A comment is metadata for a row or cell of a data frame A DataSheet will contain a list of instat_comment objects as part of the metadata for the data frame

Usage:

```
instat_comment$new(
  id = "",
  key_values = c(),
  column = "",
  value = "",
  type = "",
  comment = "",
  label = "",
  calculation = "",
  time_stamp = "",
  replies = list(),
  resolved = FALSE,
  active = TRUE,
  attributes = list()
)
```

Arguments:

id A numeric/character string representing the unique identifier for the comment. This must be unique within a data frame.

key_values A character vector storing key-value pairs associated with the comment. This identifies the row the comment is on.

column If the comment is on a cell, this is the name of the column of the cell

value If the comment is on a cell, this is the value in the cell at the time the comment was created.

type The type of comment ("critical", "warning", "message", or "").

comment A character string for the comment text or message.

label A character variable. A label or grouping for the comment e.g. if comments are produced by an operation they may all have the same label. This then allows similar comments to be identified e.g. for editing/deleting

calculation A character variable. If the comment was created through a calculation e.g. filtering the data frame, this shows how the calculation done on the data frame `time_stamp` The date and time (`POSIXct`, `POSIXt`) the comment was created, defaulting to the current system time if empty.

`replies` A list of replies to the comment. A reply could be a comment itself

`resolved` Logical value indicating if the comment is marked as resolved (`TRUE` or `FALSE`).

`active` Logical value indicating if the comment is marked as active (`TRUE` or `FALSE`).

`attributes` A named list of additional information about the comment.

Method `data_clone()`**:** Creates a deep clone of the current `instat_comment` object, including all of its fields and nested `instat_comment` replies.

Usage:

```
instat_comment$data_clone(...)
```

Arguments:

`...` Additional parameters to read in

Returns: A new `instat_comment` object with the same field values as the original, including a cloned list of `replies`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`is_climatic_element` *Is Climatic Element*

Description

Check if the column name is a climatic element.

Usage

```
is_climatic_element(x)
```

Arguments

`x`	Character, the name of the column.

Value

Logical, TRUE if the column is a climatic element, FALSE otherwise.

KGE *Calculate Kling-Gupta Efficiency*

Description

Computes the Kling-Gupta efficiency using the `hydroGOF::KGE` function.

Usage

```
KGE(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The Kling-Gupta efficiency.

link *Link Class*

Description

The `link` R6 class represents a relationship between two data frames, defined by link attributes and the columns used to link them.

Methods

`data_clone(...)` Creates a deep clone of the current `link` object, including all its fields.

`rename_data_frame_in_link(old_data_name, new_data_name)` Renames one of the data frames involved in the link.

`rename_column_in_link(data_name, old_column_name, new_column_name)` Renames a column involved in the link between data frames.

Public fields

`from_data_frame` A character string representing the name of the first data frame in the link.

`to_data_frame` A character string representing the name of the second data frame in the link.

`type` A character string representing the type of link, e.g., "keyed".

`link_columns` A list where each element defines a way to link the data frames, with each element as a named character vector.

Methods

Public methods:

- `link$new()`
- `link$data_clone()`
- `link$rename_data_frame_in_link()`
- `link$rename_column_in_link()`
- `link$clone()`

Method `new()`**:** Create a new `link` object. Defines a relationship between two data frames and specifies linking columns.

Usage:

```
link$new(
  from_data_frame = "",
  to_data_frame = "",
  type = "",
  link_columns = list()
)
```

Arguments:

`from_data_frame` A character string representing the name of the first data frame in the link.

`to_data_frame` A character string representing the name of the second data frame in the link.

`type` A character string representing the type of link, e.g., "keyed".

`link_columns` A list where each element defines a way to link the data frames, with each element as a named character vector. The names are columns in `from_data_frame` and the values are corresponding columns in `to_data_frame`. Clone `link` Object.

Method `data_clone()`**:** Creates a deep clone of the current `link` object, including all its fields.

Usage:

```
link$data_clone(...)
```

Arguments:

`...` Additional parameters to read in

Returns: A new `link` object with the same field values as the original. Rename a Data Frame in the Link.

Method `rename_data_frame_in_link()`**:** Renames the specified data frame in the link.

Usage:

```
link$rename_data_frame_in_link(old_data_name, new_data_name)
```

Arguments:

`old_data_name` The current name of the data frame to be renamed.

`new_data_name` The new name for the data frame. Rename a Column in the Link.

Method `rename_column_in_link()`**:** Renames a column involved in the link between data frames.

Usage:

```
link$rename_column_in_link(data_name, old_column_name, new_column_name)
```

Arguments:

`data_name` The name of the data frame where the column is located.

`old_column_name`	The current name of the column to be renamed.
`new_column_name`	The new name for the column.

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

Usage:

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

Arguments:

`deep`	Whether to make a deep clone.

`mae` *Calculate Mean Absolute Error*

Description

Computes the mean absolute error using the `hydroGOF::mae` function.

Usage

```
mae(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The mean absolute error.

`md` *Calculate Modified Index of Agreement*

Description

Computes the modified index of agreement using the `hydroGOF::md` function.

Usage

```
md(x, y, j = 1, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`j`	Numeric. Parameter for the modified index of agreement. Defaults to `1`.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The modified index of agreement.

`me` *Calculate Mean Error*

Description

Computes the mean error using the `hydroGOF::me` function.

Usage

```
me(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The mean error.

`missing_values_check` *Check for Missing Values*

Description

A placeholder function that always returns `FALSE`.

Usage

```
missing_values_check(x)
```

Arguments

`x`	A vector to check for missing values.

Value

Logical. Always returns `FALSE`.

`mNSE` *Calculate Modified Nash-Sutcliffe Efficiency*

Description

Computes the modified Nash-Sutcliffe efficiency using the `hydroGOF::mNSE` function.

Usage

```
mNSE(x, y, j = 1, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`j`	Numeric. Exponent parameter for the modified NSE calculation. Defaults to `1`.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The modified Nash-Sutcliffe efficiency.

mse *Calculate Mean Squared Error*

Description

Computes the mean squared error using the `hydroGOF::mse` function.

Usage

```
mse(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The mean squared error.

na_check *Check Missing Values Based on Conditions*

Description

Evaluates a vector against specified conditions for missing values.

Usage

```
na_check(
  x,
  na_type = c(),
  na_consecutive_n = NULL,
  na_max_n = NULL,
  na_max_prop = NULL,
  na_min_n = NULL,
  na_FUN = NULL,
  ...
)
```

Arguments

`x`	A vector to check for missing values.
`na_type`	A character vector specifying the types of checks to perform. Options include: • `"n"`: Total number of missing values (<= `na_max_n`). • `"prop"`: Proportion of missing values (<= `na_max_prop` in percentage). • `"n_non_miss"`: Minimum number of non-missing values (>= `na_min_n`). • `"FUN"`: A custom function to evaluate missing values. • `"con"`: Maximum consecutive missing values (<= `na_consecutive_n`).
`na_consecutive_n`	Optional. Maximum allowed consecutive missing values.
`na_max_n`	Optional. Maximum allowed missing values.
`na_max_prop`	Optional. Maximum allowed proportion of missing values (in percentage).
`na_min_n`	Optional. Minimum required non-missing values.
`na_FUN`	Optional. A custom function to evaluate missing values.
`...`	Additional arguments passed to the custom function `na_FUN`.

Value

Logical. Returns `TRUE` if all specified checks pass, otherwise `FALSE`.

`nrmse` *Calculate Normalized Root Mean Square Error*

Description

Computes the normalized root mean square error using the `hydroGOF::nrmse` function.

Usage

```
nrmse(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The normalized root mean square error.

NSE *Calculate Nash-Sutcliffe Efficiency*

Description

Computes the Nash-Sutcliffe efficiency using the `hydroGOF::NSeff` function.

Usage

```
NSE(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The Nash-Sutcliffe efficiency.

p10 *Calculate 10th Percentile*

Description

Computes the 10th percentile of a dataset using `summary_quantile`.

Usage

```
p10(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 10th percentile of the dataset.

p20 *Calculate 20th Percentile*

Description

Computes the 20th percentile of a dataset using `summary_quantile`.

Usage

```
p20(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 20th percentile of the dataset.

p25 *Calculate 25th Percentile (First Quartile)*

Description

Computes the 25th percentile of a dataset using `summary_quantile`.

Usage

```
p25(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 25th percentile of the dataset.

p30 *Calculate 30th Percentile*

Description

Computes the 30th percentile of a dataset using `summary_quantile`.

Usage

```
p30(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 30th percentile of the dataset.

p33 *Calculate 33rd Percentile*

Description

Computes the 33rd percentile of a dataset using `summary_quantile`.

Usage

```
p33(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 33rd percentile of the dataset.

p40 *Calculate 40th Percentile*

Description

Computes the 40th percentile of a dataset using `summary_quantile`.

Usage

```
p40(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 40th percentile of the dataset.

p60 *Calculate 60th Percentile*

Description

Computes the 60th percentile of a dataset using `summary_quantile`.

Usage

```
p60(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 60th percentile of the dataset.

p67 *Calculate 67th Percentile*

Description

Computes the 67th percentile of a dataset using `summary_quantile`.

Usage

```
p67(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 67th percentile of the dataset.

p70 *Calculate 70th Percentile*

Description

Computes the 70th percentile of a dataset using `summary_quantile`.

Usage

```
p70(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 70th percentile of the dataset.

p75 *Calculate 75th Percentile (Third Quartile)*

Description

Computes the 75th percentile of a dataset using `summary_quantile`.

Usage

```
p75(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 75th percentile of the dataset.

p80 *Calculate 80th Percentile*

Description

Computes the 80th percentile of a dataset using `summary_quantile`.

Usage

```
p80(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 80th percentile of the dataset.

p90 *Calculate 90th Percentile*

Description

Computes the 90th percentile of a dataset using `summary_quantile`.

Usage

```
p90(x, na.rm = FALSE, na_type = "", weights = NULL, na_max_prop = NULL, ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`...`	Additional arguments passed to `na_check`.

Value

The 90th percentile of the dataset.

PBIAS *Calculate Percent Bias*

Description

Computes the percent bias using the `hydroGOF::pbias` function.

Usage

```
PBIAS(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The percent bias.

PC *Calculate Percent Correct (PC)*

Description

Computes the percent correct using the `verification::verify` function.

Usage

```
PC(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The percent correct.

PODy *Calculate Probability of Detection (PODy)*

Description

Computes the probability of detection (PODy) using the `verification::verify` function.

Usage

```
PODy(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The probability of detection.

`proportion_calc` *Calculate Proportion*

Description

Calculates the proportion of elements in a dataset that satisfy a specified condition.

Usage

```
proportion_calc(
  x,
  prop_test = "==",
  prop_value,
  As_percentage = FALSE,
  na.rm = FALSE,
  na_type = "",
  ...
)
```

Arguments

`x`	A numeric vector.
`prop_test`	Character. The comparison operator (e.g., `"=="`, `">="`).
`prop_value`	Numeric. The value to compare against.
`As_percentage`	Logical. Return the result as a percentage if `TRUE`. Defaults to `FALSE`.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The calculated proportion or percentage.

`pss` *Calculate Pierce Skill Score*

Description

Computes the Pierce skill score using the `verification::verify` function.

Usage

```
pss(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The Pierce skill score.

R2 — *Calculate Coefficient of Determination (R^2)*

Description

Computes the coefficient of determination using the `hydroGOF::br2` function.

Usage

```
R2(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The coefficient of determination (R^2).

rd — *Calculate Relative Index of Agreement*

Description

Computes the relative index of agreement using the `hydroGOF::rd` function.

Usage

```
rd(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The relative index of agreement.

`rmse` *Calculate Root Mean Square Error*

Description

Computes the root mean square error using the `hydroGOF::rmse` function.

Usage

```
rmse(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The root mean square error.

`rNSE` *Calculate Relative Nash-Sutcliffe Efficiency*

Description

Computes the relative Nash-Sutcliffe efficiency using the `hydroGOF::rNSeff` function.

Usage

```
rNSE(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The relative Nash-Sutcliffe efficiency.

`rSD` *Calculate Ratio of Standard Deviations*

Description

Computes the ratio of standard deviations using the `hydroGOF::rSD` function.

Usage

```
rSD(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The ratio of standard deviations.

`rsr` *Calculate Ratio of RMSE*

Description

Computes the ratio of RMSE using the `hydroGOF::rsr` function.

Usage

```
rsr(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The ratio of RMSE.

`SEDI` *Calculate Symmetric Extremal Dependency Index*

Description

Computes the symmetric extremal dependency index (SEDI) using the `verification::verify` function.

Usage

```
SEDI(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "categorical").
`obs.type`	Character. The type of observation (e.g., "categorical").
`...`	Additional arguments passed to `verification::verify`.

Value

The symmetric extremal dependency index.

SEDS *Calculate Symmetric Extreme Dependency Score*

Description

Computes the symmetric extreme dependency score (SEDS) using the `verification::verify` function.

Usage

```
SEDS(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "categorical").
`obs.type`	Character. The type of observation (e.g., "categorical").
`...`	Additional arguments passed to `verification::verify`.

Value

The symmetric extreme dependency score.

ssq *Calculate Sum of Squared Residuals*

Description

Computes the sum of squared residuals using the `hydroGOF::ssq` function.

Usage

```
ssq(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The sum of squared residuals.

standardise_country_names *Standardise Country Names*

Description

Standardise country names in the dataset.

Usage

```
standardise_country_names(country)
```

Arguments

`country`	Name of Country

Value

Name of country

standard_error_mean *Calculate Standard Error of the Mean*

Description

Computes the standard error of the mean for a dataset.

Usage

```
standard_error_mean(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The standard error of the mean.

summary_ang_dev_circular *Calculate Angular Deviation of Circular Data*

Description

Computes the angular deviation of circular data using `circular::angular.deviation`.

Usage

```
summary_ang_dev_circular(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The angular deviation of the circular data.

summary_ang_var_circular *Calculate Angular Variance of Circular Data*

Description

Computes the angular variance of circular data using `circular::angular.variance`.

Usage

```
summary_ang_var_circular(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The angular variance of the circular data.

`summary_coef_var` *Calculate Coefficient of Variation*

Description

Computes the coefficient of variation for a dataset.

Usage

```
summary_coef_var(x, na.rm = FALSE, weights = NULL, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`weights`	Optional weights for the data.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The coefficient of variation.

`summary_cor` *Calculate Correlation*

Description

Computes the correlation or weighted correlation between two datasets.

Usage

```
summary_cor(
  x,
  y,
  na.rm = FALSE,
  na_type = "",
  weights = NULL,
  method = c("pearson", "kendall", "spearman"),
  cor_use = c("everything", "all.obs", "complete.obs", "na.or.complete",
    "pairwise.complete.obs"),
  ...
)
```

Arguments

`x`	A numeric vector.
`y`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`method`	Character. Correlation method ("pearson", "kendall", "spearman"). Defaults to `"pearson"`.
`cor_use`	Character. How missing data is handled ("everything", "all.obs", etc.). Defaults to `"everything"`.
`...`	Additional arguments passed to `na_check`.

Value

The correlation or weighted correlation.

`summary_count` *Count Non-Missing Elements*

Description

Counts the number of non-missing (non-NA) elements in a dataset.

Usage

```
summary_count(x, ...)
```

Arguments

`x`	A vector of data.
`...`	Additional arguments (not used).

Value

The count of non-missing elements in the dataset.

`summary_count_all` *Count Elements in a Dataset*

Description

Counts the number of elements in a dataset.

Usage

```
summary_count_all(x, ...)
```

Arguments

`x`	A vector of data.
`...`	Additional arguments (not used).

Value

The count of elements in the dataset.

`summary_count_miss` *Count Missing Elements in a Dataset*

Description

Counts the number of missing (NA) elements in a dataset.

Usage

```
summary_count_miss(x, ...)
```

Arguments

`x`	A vector of data.
`...`	Additional arguments (not used).

Value

The count of missing elements in the dataset.

summary_cov *Calculate Covariance*

Description

Computes the covariance or weighted covariance between two datasets.

Usage

```
summary_cov(
  x,
  y,
  na.rm = FALSE,
  weights = NULL,
  na_type = "",
  method = c("pearson", "kendall", "spearman"),
  use = c("everything", "all.obs", "complete.obs", "na.or.complete",
    "pairwise.complete.obs"),
  ...
)
```

Arguments

`x`	A numeric vector.
`y`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`weights`	Optional weights for the data.
`na_type`	Character string indicating the type of NA check to perform.
`method`	Character. Covariance method ("pearson", "kendall", "spearman"). Defaults to `"pearson"`.
`use`	Character. How missing data is handled ("everything", "all.obs", etc.). Defaults to `"everything"`.
`...`	Additional arguments passed to `na_check`.

Value

The covariance or weighted covariance.

summary_first *Get First Element*

Description

Returns the first element of a vector, optionally ordered by another vector.

Usage

```
summary_first(x, order_by = NULL, ...)
```

Arguments

`x`	A vector.
`order_by`	Optional vector to order by.
`...`	Additional arguments (not used).

Value

The first element of the vector.

`summary_kurtosis` *Calculate Kurtosis*

Description

Computes the kurtosis or weighted kurtosis of a dataset.

Usage

```
summary_kurtosis(x, na.rm = FALSE, weights = NULL, type = 2, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`weights`	Optional weights for the data.
`type`	Integer. Type of kurtosis calculation. Defaults to `2`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The kurtosis or weighted kurtosis of the dataset.

`summary_last` *Get Last Element*

Description

Returns the last element of a vector, optionally ordered by another vector.

Returns the last element of a vector, optionally ordered by another vector.

Usage

```
summary_last(x, order_by = NULL, ...)
```

```
summary_last(x, order_by = NULL, ...)
```

Arguments

`x`	A vector.
`order_by`	Optional. A vector to order by before selecting the last element.
`...`	Additional arguments (not used).

Value

The last element of the vector.

The last element of the vector.

`summary_max` *Calculate Maximum Value*

Description

Computes the maximum value in a dataset.

Usage

```
summary_max(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The maximum value in the dataset.

`summary_max_circular` *Calculate the Maximum of Circular Data*

Description

Computes the maximum value of circular data using `circular::quantile.circular`.

Usage

```
summary_max_circular(
  x,
  na.rm = FALSE,
  names = FALSE,
  type = 7,
  na_type = "",
  ...
)
```

Arguments

x	A vector of circular data.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
names	Logical. Should the names of the quantiles be returned? Defaults to FALSE.
type	Integer. Type of quantile calculation.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The maximum of the circular data.

summary_mean *Calculate Mean of Data*

Description

Computes the mean or weighted mean of a dataset.

Usage

```
summary_mean(
  x,
  add_cols,
  weights = NULL,
  na.rm = FALSE,
  trim = 0,
  na_type = "",
  ...
)
```

Arguments

x	A numeric vector.
add_cols	Additional columns (not used directly in calculation).
weights	Optional weights for the data.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
trim	Numeric. Fraction of observations to trim from each end before computing the mean.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The mean or weighted mean of the data.

`summary_mean_circular` *Calculate the Mean of Circular Data*

Description

Computes the mean of circular data using `circular::mean.circular`.

Usage

```
summary_mean_circular(
  x,
  na.rm = FALSE,
  control.circular = list(),
  na_type = "",
  ...
)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`control.circular`	List of control parameters for circular objects.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The mean of the circular data.

`summary_median` *Calculate Median*

Description

Computes the median or weighted median of a dataset. Handles ordered factors and dates.

Usage

```
summary_median(x, na.rm = FALSE, weights = NULL, na_type = "", ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`weights`	Optional weights for the data.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The median or weighted median of the dataset.

`summary_medianHL_circular` *Calculate the Hodges-Lehmann Median of Circular Data*

Description

Computes the Hodges-Lehmann median of circular data using `circular::medianHL.circular`.

Usage

```
summary_medianHL_circular(
  x,
  na.rm = FALSE,
  method = c("HL1", "HL2", "HL3"),
  prop = NULL,
  na_type = "",
  ...
)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`method`	Character string specifying the HL method ("HL1", "HL2", or "HL3").
`prop`	Numeric. Proportion of data to consider.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The Hodges-Lehmann median of the circular data.

`summary_median_absolute_deviation` *Calculate Median Absolute Deviation*

Description

Computes the median absolute deviation or weighted absolute deviation.

Usage

```
summary_median_absolute_deviation(
  x,
  constant = 1.4826,
  na.rm = FALSE,
  na_type = "",
  weights = NULL,
  low = FALSE,
  high = FALSE,
  ...
)
```

Arguments

`x`	A numeric vector.
`constant`	Numeric. Scale factor. Defaults to `1.4826`.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`weights`	Optional weights for the data.
`low`	Logical. Use only values below the median. Defaults to `FALSE`.
`high`	Logical. Use only values above the median. Defaults to `FALSE`.
`...`	Additional arguments passed to `na_check`.

Value

The median absolute deviation or weighted absolute deviation.

`summary_median_circular` *Calculate the Median of Circular Data*

Description

Computes the median of circular data using `circular::median.circular`.

Usage

```
summary_median_circular(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The median of the circular data.

summary_min *Calculate Minimum Value*

Description

Computes the minimum value in a dataset.

Usage

```
summary_min(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

x	A numeric vector.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The minimum value in the dataset.

summary_min_circular *Calculate the Minimum of Circular Data*

Description

Computes the minimum value of circular data using circular::quantile.circular.

Usage

```
summary_min_circular(
  x,
  na.rm = FALSE,
  names = FALSE,
  type = 7,
  na_type = "",
  ...
)
```

Arguments

x	A vector of circular data.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
names	Logical. Should the names of the quantiles be returned? Defaults to FALSE.
type	Integer. Type of quantile calculation.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The minimum of the circular data.

`summary_mode` *Calculate Mode*

Description

Determines the mode (most frequent value) of a vector.

Usage

```
summary_mode(x, ...)
```

Arguments

`x`	A vector of data.
`...`	Additional arguments (unused).

Value

The mode of the vector. Returns `NA` if the input is `NULL`.

`summary_nth` *Get nth Element*

Description

Returns the nth element of a vector, optionally ordered by another vector.

Returns the nth element of a vector, optionally ordered by another vector.

Usage

```
summary_nth(x, nth_value, order_by = NULL, ...)
```

```
summary_nth(x, nth_value, order_by = NULL, ...)
```

Arguments

`x`	A vector.
`nth_value`	Integer. The position of the element to return.
`order_by`	Optional. A vector to order by before selecting the nth element.
`...`	Additional arguments (not used).

Value

The nth element of the vector.

The nth element of the vector.

`summary_n_distinct` *Count Distinct Elements*

Description

Counts the number of distinct elements in a vector.

Usage

```
summary_n_distinct(x, na.rm = FALSE, ...)
```

Arguments

`x`	A vector of data.
`na.rm`	Logical. Should missing values (`NA`) be removed? Defaults to `FALSE`.
`...`	Additional arguments (not used).

Value

The count of distinct elements in the vector.

`summary_outlier_limit` *Calculate Outlier Limits*

Description

Computes the upper or lower outlier limits based on skewness and interquartile range.

Usage

```
summary_outlier_limit(
  x,
  coef = 1.5,
  bupperlimit = TRUE,
  bskewedcalc = FALSE,
  skewnessweight = 4,
  na.rm = TRUE,
  na_type = "",
  omit = FALSE,
  value = 0,
  ...
)
```

Arguments

`x`	A numeric vector.
`coef`	Numeric. Coefficient for the IQR. Defaults to `1.5`.
`bupperlimit`	Logical. Calculate upper limit if `TRUE`, lower limit otherwise. Defaults to `TRUE`.
`bskewedcalc`	Logical. Use skewness in the calculation if `TRUE`. Defaults to `FALSE`.
`skewnessweight`	Numeric. Weight for skewness in the calculation. Defaults to `4`.
`na.rm`	Logical. Should missing values be removed? Defaults to `TRUE`.
`na_type`	Character string indicating the type of NA check to perform.
`omit`	Logical. Omit values below a threshold. Defaults to `FALSE`.
`value`	Numeric. Threshold for omission. Defaults to `0`.
`...`	Additional arguments passed to `na_check`.

Value

The calculated outlier limit.

`summary_Q1_circular` *Calculate the First Quartile of Circular Data*

Description

Computes the first quartile (Q1) of circular data using `circular::quantile.circular`.

Usage

```
summary_Q1_circular(
  x,
  na.rm = FALSE,
  names = FALSE,
  type = 7,
  na_type = "",
  ...
)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`names`	Logical. Should the names of the quantiles be returned? Defaults to `FALSE`.
`type`	Integer. Type of quantile calculation.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The first quartile of the circular data.

`summary_Q3_circular` *Calculate the Third Quartile of Circular Data*

Description

Computes the third quartile (Q3) of circular data using `circular::quantile.circular`.

Usage

```
summary_Q3_circular(
  x,
  na.rm = FALSE,
  names = FALSE,
  type = 7,
  na_type = "",
  ...
)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`names`	Logical. Should the names of the quantiles be returned? Defaults to `FALSE`.
`type`	Integer. Type of quantile calculation.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The third quartile of the circular data.

`summary_Qn` *Calculate Qn*

Description

Computes the robust Qn scale estimator using `robustbase::Qn`.

Usage

```
summary_Qn(
  x,
  constant = 2.21914,
  finite.corr = missing(constant),
  na.rm = FALSE,
  na_type = "",
  ...
)
```

Arguments

`x`	A numeric vector.
`constant`	Numeric. Scale factor. Defaults to `2.21914`.
`finite.corr`	Logical. Apply finite sample correction. Defaults to `TRUE`.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The Qn scale estimator.

`summary_quantile` *Calculate Quantile*

Description

Computes the quantile or weighted quantile of a dataset. Handles ordered factors and dates.

Usage

```
summary_quantile(x, na.rm = FALSE, weights = NULL, probs, na_type = "", ...)
```

Arguments

`x`	A numeric vector, ordered factor, or date.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`weights`	Optional weights for the data.
`probs`	Numeric vector of probabilities (e.g., 0.1 for 10th percentile).
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The quantile or weighted quantile of the dataset.

summary_quantile_circular *Calculate Quantiles of Circular Data*

Description

Computes the quantiles of circular data using `circular::quantile.circular`.

Usage

```
summary_quantile_circular(
  x,
  probs = seq(0, 1, 0.25),
  na.rm = FALSE,
  names = FALSE,
  type = 7,
  na_type = "",
  ...
)
```

Arguments

`x`	A vector of circular data.
`probs`	Numeric vector of probabilities.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`names`	Logical. Should the names of the quantiles be returned? Defaults to `FALSE`.
`type`	Integer. Type of quantile calculation.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The quantiles of the circular data.

summary_range *Calculate Range*

Description

Computes the range of a dataset (difference between the maximum and minimum values).

Usage

```
summary_range(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The range of the dataset.

`summary_range_circular` *Calculate Range of Circular Data*

Description

Computes the range of circular data using `circular::range.circular`.

Usage

```
summary_range_circular(
  x,
  test = FALSE,
  na.rm = FALSE,
  finite = FALSE,
  control.circular = list(),
  na_type = "",
  ...
)
```

Arguments

`x`	A vector of circular data.
`test`	Logical. Perform a statistical test on the range. Defaults to `FALSE`.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`finite`	Logical. Should only finite values be considered? Defaults to `FALSE`.
`control.circular`	List of control parameters for circular objects.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The range of the circular data.

`summary_rho_circular` *Calculate Rho of Circular Data*

Description

Computes the Rho (mean resultant length) of circular data using `circular::rho.circular`.

Usage

```
summary_rho_circular(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The Rho of the circular data.

`summary_sample` *Sample a Single Element*

Description

Randomly samples a single element from a vector.

Usage

```
summary_sample(x, replace = FALSE, seed, ...)
```

Arguments

`x`	A vector of data.
`replace`	Logical. Should sampling be with replacement? Defaults to `FALSE`.
`seed`	Optional. A seed for reproducibility.
`...`	Additional arguments (not used).

Value

A randomly sampled element from the vector.

summary_sd *Calculate Standard Deviation*

Description

Computes the standard deviation or weighted standard deviation of a dataset.

Usage

```
summary_sd(x, na.rm = FALSE, weights = NULL, na_type = "", ...)
```

Arguments

x	A numeric vector.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
weights	Optional weights for the data.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The standard deviation or weighted standard deviation of the data.

summary_sd_circular *Calculate the Standard Deviation of Circular Data*

Description

Computes the standard deviation of circular data using circular::sd.circular.

Usage

```
summary_sd_circular(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

x	A vector of circular data.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The standard deviation of the circular data.

summary_skewness *Calculate Skewness*

Description

Computes the skewness or weighted skewness of a dataset.

Usage

```
summary_skewness(x, weights = NULL, na.rm = FALSE, type = 2, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`weights`	Optional weights for the data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`type`	Integer. Type of skewness calculation. Defaults to `2`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The skewness or weighted skewness of the dataset.

summary_skewness_mc *Calculate Medcouple Skewness*

Description

Computes the medcouple skewness using `robustbase::mc`.

Usage

```
summary_skewness_mc(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The medcouple skewness.

`summary_Sn` *Calculate Sn*

Description

Computes the robust Sn scale estimator using `robustbase::Sn`.

Usage

```
summary_Sn(
  x,
  constant = 1.1926,
  finite.corr = missing(constant),
  na.rm = FALSE,
  na_type = "",
  ...
)
```

Arguments

`x`	A numeric vector.
`constant`	Numeric. Scale factor. Defaults to `1.1926`.
`finite.corr`	Logical. Apply finite sample correction. Defaults to `TRUE`.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The Sn scale estimator.

`summary_sum` *Calculate Sum of Data*

Description

Computes the sum or weighted sum of a dataset.

Usage

```
summary_sum(x, weights = NULL, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`weights`	Optional weights for the data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The sum or weighted sum of the data.

`summary_trimmed_mean` *Calculate Trimmed Mean of Data*

Description

Computes the trimmed mean of a dataset.

Usage

```
summary_trimmed_mean(
  x,
  add_cols,
  weights = NULL,
  na.rm = FALSE,
  trimmed = 0,
  na_type = "",
  ...
)
```

Arguments

`x`	A numeric vector.
`add_cols`	Additional columns (not used directly in calculation).
`weights`	Optional weights for the data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`trimmed`	Numeric. Fraction of observations to trim from each end before computing the mean.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The trimmed mean of the data.

summary_var *Calculate Variance*

Description

Computes the variance or weighted variance of a dataset.

Usage

```
summary_var(x, na.rm = FALSE, weights = NULL, na_type = "", ...)
```

Arguments

`x`	A numeric vector.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`weights`	Optional weights for the data.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The variance or weighted variance of the data.

summary_var_circular *Calculate the Variance of Circular Data*

Description

Computes the variance of circular data using `circular::var.circular`.

Usage

```
summary_var_circular(x, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	A vector of circular data.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The variance of the circular data.

summary_where_max *Get Corresponding Value for Maximum*

Description

Returns the value in another vector corresponding to the maximum value in a dataset.

Usage

```
summary_where_max(x, summary_where_y = NULL, na.rm = TRUE, na_type = "", ...)
```

Arguments

x	A numeric vector.
summary_where_y	A vector of values corresponding to x.
na.rm	Logical. Should missing values be removed? Defaults to TRUE.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The value in summary_where_y corresponding to the maximum value in x.

summary_where_min *Get Corresponding Value for Minimum*

Description

Returns the value in another vector corresponding to the minimum value in a dataset.

Usage

```
summary_where_min(x, summary_where_y = NULL, na.rm = TRUE, na_type = "", ...)
```

Arguments

x	A numeric vector.
summary_where_y	A vector of values corresponding to x.
na.rm	Logical. Should missing values be removed? Defaults to TRUE.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The value in summary_where_y corresponding to the minimum value in x.

summary_which_max *Get Indices of Maximum Value*

Description

Finds all indices of the maximum value in a dataset.

Usage

```
summary_which_max(x, na.rm = TRUE, na_type = "", ...)
```

Arguments

x	A numeric vector.
na.rm	Logical. Should missing values be removed? Defaults to TRUE.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

A vector of indices corresponding to the maximum value.

summary_which_min *Get Indices of Minimum Value*

Description

Finds all indices of the minimum value in a dataset.

Usage

```
summary_which_min(x, na.rm = TRUE, na_type = "", ...)
```

Arguments

x	A numeric vector.
na.rm	Logical. Should missing values be removed? Defaults to TRUE.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

A vector of indices corresponding to the minimum value.

TS *Calculate Threat Score*

Description

Computes the threat score using the `verification::verify` function.

Usage

```
TS(x, y, frcst.type, obs.type, ...)
```

Arguments

`x`	Observed values.
`y`	Predicted values.
`frcst.type`	Character. The type of forecast (e.g., "binary").
`obs.type`	Character. The type of observation (e.g., "binary").
`...`	Additional arguments passed to `verification::verify`.

Value

The threat score.

VE *Calculate Volumetric Efficiency*

Description

Computes the volumetric efficiency using the `hydroGOF::VE` function.

Usage

```
VE(x, y, na.rm = FALSE, na_type = "", ...)
```

Arguments

`x`	Observed values.
`y`	Simulated values.
`na.rm`	Logical. Should missing values be removed? Defaults to `FALSE`.
`na_type`	Character string indicating the type of NA check to perform.
`...`	Additional arguments passed to `na_check`.

Value

The volumetric efficiency.

Index