Package 'vvauditor'

May 8, 2026

Title Creates Assertion Tests

Version 0.8.0

Description
Offers a comprehensive set of assertion tests to help users validate the integrity of their data. These tests can be used to check for specific conditions or properties within a dataset and help ensure that data is accurate and reliable.
The package is designed to make it easy to add quality control checks to data analysis workflows and to aid in identifying and correcting any errors or inconsistencies in data.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports checkmate, cli, dplyr, findR, janitor, kit, lubridate, magrittr, purrr, readr, stats, stringr, tibble, tidyr

Suggests devtools (>= 2.4.5), knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Tomer Iwan [cph], Hajo Bons [aut, cph, cre]

Maintainer Hajo Bons <h.b.bons@vu.nl>

Repository CRAN

Date/Publication 2026-01-19 14:00:18 UTC

Contents

`assertion_message` *Assert Message Based on Type*

Description

This function asserts a message based on the type specified. It can either push the message to an AssertCollection, print a warning, or stop execution with an error message.

Usage

```
assertion_message(message, assertion_fail = "stop")
```

Arguments

`message`	A character string representing the message to be asserted.
`assertion_fail`	A character string indicating the action to take if the assertion fails. Can be an AssertCollection, "warning", or "stop" (default).

Value

None

`assert_date_named` *Assert Date Value in Column*

Description

This function asserts that the values in a specified column of a data frame are of Date type. It uses the `checkmate::assert_date` function to perform the assertion.

Usage

```
assert_date_named(column, df, prefix_column = NULL, ...)
```

Arguments

`column`	A character vector or string with the column name to be tested.
`df`	The data frame that contains the column.
`prefix_column`	A character string that will be prepended to the column name in the assertion message. Default is NULL.
`...`	Additional parameters are passed to the `checkmate::assert_date` function.

Value

None

`assert_field_consistency`

Check if the fieldnames of the dataset are the same

Description

Assert Field Consistency Between Data and Metadata

Usage

```
assert_field_consistency(new_data, field_info)
```

Arguments

`new_data`	A data frame. The new dataset whose field names need to be checked.
`field_info`	A data frame. Metadata containing a column named `raw_field_name` that lists the expected field names.

Details

This function checks for consistency between the field names in new data and the field names specified in a metadata reference. It warns if there are missing fields in the new data or if new unexpected fields appear in the data that are not defined in the metadata.

Value

No return value. The function issues warnings if there are inconsistencies in field names.

`assert_field_distinctness`

Assert Field Uniqueness Consistency Between Data and Metadata

---

Description

This function checks whether the uniqueness of columns in a new dataset matches the expected uniqueness defined in a metadata reference. It warns if any columns do not conform to the expected uniqueness.

Usage

```
assert_field_distinctness(new_data, metadata)
```

Arguments

new_data	A data frame. The dataset whose column uniqueness needs to be verified.
metadata	A data frame. Metadata containing a column named is_unique_column, indicating whether each field is expected to be unique.

Value

No return value. The function issues warnings if any columns deviate from their expected uniqueness.

---

assert_field_existence

Assert Field Existence in New Data

Description

This function checks whether all fields that existed in a previous dataset are still present in a new dataset, based on a metadata reference. It warns if any fields from the previous dataset are missing in the new dataset.

Usage

```
assert_field_existence(new_data, previous_data, metadata)
```

Arguments

new_data	A data frame. The new dataset whose field names need to be checked.
previous_data	A data frame. The previous dataset used as a reference for expected fields.
metadata	A data frame. Metadata containing a column named raw_field_name, which defines the expected field names.

Value

No return value. The function issues warnings if any expected fields are missing in the new dataset.

assert_logical_named *Assert Logical Value in Column*

Description

This function asserts that the values in a specified column of a data frame are logical. It uses the `checkmate::assert_logical` function to perform the assertion.

Usage

```
assert_logical_named(column, df, prefix_column = NULL, ...)
```

Arguments

`column`	A character vector or string with the column name to be tested.
`df`	The data frame that contains the column.
`prefix_column`	A character string that will be prepended to the column name in the assertion message. Default is NULL.
`...`	Additional parameters are passed to the `checkmate::assert_logical` function.

Value

None

Examples

```
# Create a data frame
df <- data.frame(a = c(TRUE, FALSE, TRUE, FALSE), b = c(1, 2, 3, 4))
# Assert that the values in column "a" are logical
assert_logical_named("a", df)
```

assert_missing_values *Assert Consistency of Missing Values in Data*

Description

This function checks whether the percentage of missing values in a dataset matches the documented percentage in a metadata reference. It warns if there are significant discrepancies.

Usage

```
assert_missing_values(data, metadata)
```

Arguments

data	A data frame. The dataset to check for missing values.
metadata	A data frame. Metadata containing expected missing value percentages and valid value counts. It must include the columns raw_field_name, percentage_of_missing_values, count_of_valid_values, and preferred_field_name.

Value

No return value. The function issues warnings if the actual missing value percentages deviate significantly from the documented values.

---

assert_no_duplicates_in_group

Assert No Duplicates in Group

Description

This function asserts that there are no duplicate rows in the specified columns of a data frame. It groups the data frame by the specified columns, counts the number of unique values for each group, and checks if there are any groups with more than one row. If there are, it prints an error message and stops the execution (unless assertion_fail is set to "warn").

Usage

```
assert_no_duplicates_in_group(df, group_vars, assertion_fail = "stop")
```

Arguments

df	A data frame.
group_vars	A character vector of column names.
assertion_fail	A character string indicating the action to take if the assertion fails. Can be "stop" (default) or "warn".

Value

The input data frame.

assert_range_validation

Assert Range Validation for Data Fields

Description

This function checks whether the values in a dataset fall within the expected minimum and maximum range as specified in the metadata. It warns if any values violate the expected range.

Usage

```
assert_range_validation(data, metadata)
```

Arguments

`data`	A data frame. The dataset containing the fields to validate.
`metadata`	A data frame. Metadata containing expected minimum and maximum values for each field. It must include the columns `raw_field_name`, `min`, `max`, and `preferred_field_name`.

Value

No return value. The function issues warnings if any values fall outside the expected range.

assert_type_consistency

Assert Type Consistency Between Data and Metadata

Description

This function checks whether the data types of fields in a dataset match the expected types specified in the metadata. It warns if any fields have a different type than expected.

Usage

```
assert_type_consistency(data, metadata)
```

Arguments

`data`	A data frame. The dataset containing the fields to validate.
`metadata`	A data frame. Metadata specifying the expected data types for each field. It must include the columns `raw_field_name`, `type_of_variable`, and `preferred_field_name`.

Value

No return value. The function issues warnings if any fields have an unexpected type.

calculate_category_percentages

Calculate the percentage of categories in a data vector

Description

This function calculates the percentage of each category in a given data vector and returns the top 10 categories along with their percentages. If the data vector is of Date class, it is converted to POSIXct. If the sum of the percentages is not 100%, an "Other" category is added to make up the difference, but only if the number of unique values exceeds 10. If the data vector is of POSIXct class and the smallest percentage is less than 1%, the function returns "Not enough occurrences."

Usage

```
calculate_category_percentages(data_vector)
```

Arguments

data_vector	A vector of categorical data.

Value

A character string detailing the top 10 categories and their percentages, or a special message indicating not enough occurrences or unsupported data type.

Examples

```
# Example with a character vector
data_vector <- c("cat", "dog", "bird", "cat", "dog", "cat", "other")
calculate_category_percentages(data_vector)

# Example with a Date vector
data_vector <- as.Date(c("2020-01-01", "2020-01-02", "2020-01-03"))
calculate_category_percentages(data_vector)
```

check_double_columns *check double columns*

Description

Check whether two dataframes have intersecting column names.

Usage

```
check_double_columns(x, y, connector = NULL)
```

Arguments

x	Data frame x.
y	Data frame y.
connector	The connector columns as strings. Also possible as vector.

Value

Message informing about overlap in columns between the dataframes.

See Also

Other tests: check_no_duplicates_in_group(), check_numeric_or_integer_type(), check_posixct_type(), duplicates_in_column(), test_all_equal()

Examples

```
check_double_columns(mtcars, iris)
```

---

check_duplicates *Check for Duplicate Rows in Selected Columns*

Description

This function checks if there are any duplicate rows in the specified columns of a data frame. It prints the unique rows and returns a boolean indicating whether the number of rows in the original data frame is the same as the number of rows in the data frame with duplicate rows removed.

Usage

```
check_duplicates(data, columns)
```

Arguments

data	A data frame.
columns	A character vector of column names.

Value

A logical value indicating whether the number of rows in the original data frame is the same as the number of rows in the data frame with duplicate rows removed.

Examples

```
# Create a data frame
df <- data.frame(a = c(1, 2, 3, 1), b = c(4, 5, 6, 4), c = c(7, 8, 9, 7))
# Check for duplicate rows in the first two columns
check_duplicates(df, c("a", "b"))
```

check_na_columns *Check for columns with only NA values*

Description

This function checks if there are any columns in the provided dataframe that contain only NA values. If such columns exist, their names are added to the provided collection.

Usage

```
check_na_columns(df, collection)
```

Arguments

df	A dataframe.
collection	A list to store the names of the columns with only NA values.

Value

The updated collection.

Examples

```
# Create a dataframe with some columns containing only NA values
df <- data.frame(a = c(1, NA, 3), b = c(NA, NA, NA), c = c(4, 5, 6))
collection <- checkmate::makeAssertCollection()
check_na_columns(df, collection)
```

check_non_zero_rows *Check for Non-Zero Rows*

Description

This function checks if there are more than 0 rows in the provided dataframe. If there are 0 rows, a message is added to the provided collection.

Usage

```
check_non_zero_rows(dataframe, collection)
```

Arguments

dataframe	A dataframe.
collection	A list to store the message if there are 0 rows.

Value

The updated collection.

Examples

```
# Create an empty dataframe
dataframe <- data.frame()
collection <- checkmate::makeAssertCollection()
check_non_zero_rows(dataframe, collection)
```

`check_no_duplicates_in_group` *Check for No Duplicates in Group*

Description

This function checks if there is exactly one row per group in the provided dataframe. If there are multiple rows per group, the assertion fails.

Usage

```
check_no_duplicates_in_group(
  dataframe,
  group_variables = NULL,
  assertion_fail = "stop"
)
```

Arguments

`dataframe`	The dataframe to be checked.
`group_variables`	The group variables as a character vector. The default is NULL.
`assertion_fail`	How the function reacts to a failure. This can be a "warning", where only a warning is given on the failure, or a "stop", where the function execution is stopped and the message is displayed, or an "AssertCollection", where the failure message is added to an assertion collection.

See Also

Other assertions: `check_numeric_or_integer_type()`, `check_posixct_type()`

Other tests: `check_double_columns()`, `check_numeric_or_integer_type()`, `check_posixct_type()`, `duplicates_in_column()`, `test_all_equal()`

Examples

```
# Create a dataframe with some groups having more than one row
dataframe <- data.frame(a = c(1, 1, 2), b = c(2, 2, 3), c = c("x", "x", "y"))
# Check the uniqueness of rows per group
check_no_duplicates_in_group(dataframe)
```

check_no_duplicate_rows *Check for No Duplicate Rows*

Description

This function checks if there are any duplicate rows in the provided dataframe. If there are duplicate rows, a message is added to the provided collection.

Usage

```
check_no_duplicate_rows(dataframe, collection, unique_columns = NULL)
```

Arguments

dataframe	A dataframe.
collection	A list to store the message if there are duplicate rows.
unique_columns	Default is NULL. If provided, these are the columns to check for uniqueness.

Value

The updated collection.

Examples

```
# Create a dataframe with some duplicate rows
dataframe <- data.frame(a = c(1, 1, 2), b = c(2, 2, 3))
collection <- checkmate::makeAssertCollection()
check_no_duplicate_rows(dataframe, collection, c("a", "b"))
```

check_numeric_or_integer_type *Check for Numeric or Integer Type*

Description

This function checks if the specified column in the provided dataframe has a numeric or integer type. It uses the checkmate::assert_numeric or checkmate::assert_integer function to perform the assertion, depending on the value of the field_type parameter.

Usage

```
check_numeric_or_integer_type(
  column_name,
  dataframe,
  column_prefix = NULL,
  field_type = "numeric",
  ...
)
```

Arguments

column_name	A character vector or string with the column name to be tested.
dataframe	The dataframe that contains the column.
column_prefix	Default is NULL. If provided, this text is prepended to the variable name in the assertion message.
field_type	Default is "numeric". Specify "integer" to check if the column has an integer type. This parameter must be either "integer" or "numeric".
...	The remaining parameters are passed to the function assert_numeric or assert_integer.

See Also

Other assertions: check_no_duplicates_in_group(), check_posixct_type()

Other tests: check_double_columns(), check_no_duplicates_in_group(), check_posixct_type(), duplicates_in_column(), test_all_equal()

Examples

```
# Create a dataframe with a numeric column
dataframe <- data.frame(a = c(1, 2, 3))
# Check the numeric type of the 'a' column
check_numeric_or_integer_type("a", dataframe)
```

---

check_posixct_type *Check for POSIXct Type*

Description

This function checks if the specified column in the provided dataframe has a POSIXct type. It uses the checkmate::assert_posixct function to perform the assertion.

Usage

```
check_posixct_type(column_name, dataframe, column_prefix = NULL, ...)
```

Arguments

column_name	A character vector or string with the column name to be tested.
dataframe	The dataframe that contains the column.
column_prefix	Default is NULL. If provided, this text is prepended to the variable name in the assertion message.
...	The remaining parameters are passed to the function assert_posixct.

See Also

Other assertions: check_no_duplicates_in_group(), check_numeric_or_integer_type()

Other tests: check_double_columns(), check_no_duplicates_in_group(), check_numeric_or_integer_type(), duplicates_in_column(), test_all_equal()

Examples

```
# Create a dataframe with a POSIXct column
dataframe <- data.frame(date = as.POSIXct("2023-10-04"))
# Check the POSIXct type of the 'date' column
check_posixct_type("date", dataframe)
```

---

check_rows *Check rows*

Description

This function prints the number of rows of a data frame. This function is used to check that rows are not deleted or doubled unless expected.

Usage

```
check_rows(df, name = NULL)
```

Arguments

df	The data frame whose rows are to be counted
name	The name of the data file (this will be printed)

Value

A message is printed to the console with the number of rows of the data

Examples

```
check_rows(mtcars)
```

check_zero_columns *Check for Columns with Only 0s*

Description

This function checks if there are any columns in the provided dataframe that contain only 0 values. If such columns exist, their names are added to the provided collection.

Usage

```
check_zero_columns(dataframe, collection)
```

Arguments

dataframe	A dataframe.
collection	A list to store the names of the columns with only 0 values.

Value

The updated collection.

Examples

```
# Create a dataframe with some columns containing only 0 values
dataframe <- data.frame(a = c(0, 0, 0), b = c(1, 2, 3), c = c(0, 0, 0))
collection <- checkmate::makeAssertCollection()
check_zero_columns(dataframe, collection)
```

count_more_than_1 *Count more than 1*

Description

Function to count the number of values greater than 1 in a vector This function is used in the function Check_columns_for_double_rows to count duplicate values.

Usage

```
count_more_than_1(x)
```

Arguments

x	The vector to test

Value

Number of values greater than 1.

Examples

```
count_more_than_1(c(1, 1, 4))
```

`create_categorical_details` *Create categorical details csv*

Description

This function returns a categorical details csv. Containing categorical information about the dataset

Usage

```
create_categorical_details(data, mapping)
```

Arguments

`data`	A dataframe for which to create a categorical details csv.
`mapping`	A dataframe containing a mapping named vector, containing preferred fieldnames Example: column_names <- c(mpg = "mpg", cyl = "cyl", disp = "disp", hp = "hp", drat = "drat", wt = "wt", qsec = "qsec", vs = "vs", am = "am", gear = "gear", carb = "carb", spare_tire = "spare_tire"

Value

Dataframe containing categorical details

`create_dataset_summary_table` *Create dataset summary statistics table*

Description

This function creates a summary statistics table for a dataframe, providing insights into the nature of the data contained within. It includes detailed statistics for each column, such as column types, missing value percentages, minimum and maximum values for numeric columns, patterns for character columns, uniqueness of identifiers, and distributions.

Usage

```
create_dataset_summary_table(df_input)
```

Arguments

`df_input`	A dataframe for which to create a summary statistics table.

Value

A tibble with comprehensive summary statistics for each column in the input dataframe.

`create_data_types` *Create data types tibble*

Description

This function returns a data types tibble. Containing type information about the dataset.

Usage

```
create_data_types(data, mapping)
```

Arguments

`data`	A dataframe for which to create a data types csv.
`mapping`	A dataframe containing a mapping named vector, containing preferred field-names Example: column_names <- c(mpg = "mpg", cyl = "cyl", disp = "disp", hp = "hp", drat = "drat", wt = "wt", qsec = "qsec", vs = "vs", am = "am", gear = "gear", carb = "carb", spare_tire = "spare_tire")

Value

Tibble containing data_types

`create_field_info` *Create field info*

Description

This function returns a dataframe containing field info information about the dataset

Usage

```
create_field_info(
  data,
  raw_data_path = NULL,
  broker = NULL,
  product = NULL,
  public_dataset = NULL
)
```

Arguments

data	A dataframe for which to create a field info csv.
raw_data_path	A string containing the original location of the original raw file
broker	The name of of the organisation or person that distributes the dataset
product	The name of the product where this dataset is used in
public_dataset	Boolean containing whether the dataset is publicly available is_primary_key Is_primary_key is variable that can be manually set to TRUE if the dataset contains a primary key.

Value

Dataframe containing subset info

---

create_numeric_details

Create numeric details csv

Description

This function returns a numeric details csv. Containing numeric information about the dataset

Usage

```
create_numeric_details(data, mapping)
```

Arguments

data	A dataframe for which to create a numeric details csv.
mapping	A dataframe containing a mapping named vector, containing preferred fieldnames Example: column_names <- c(mpg = "mpg", cyl = "cyl", disp = "disp", hp = "hp", drat = "drat", wt = "wt", qsec = "qsec", vs = "vs", am = "am", gear = "gear", carb = "carb", spare_tire = "spare_tire")

Value

Dataframe containing numeric details.

`create_subset_fields` *Create subset fields*

Description

This function returns a subsetfields info df. Containing subsetfields information about the dataset

Usage

```
create_subset_fields(data, mapping)
```

Arguments

`data`	A dataframe for which to create a subsetfields csv.
`mapping`	A dataframe containing a mapping named vector, containing preferred field-names Example: column_names <- c(mpg = "mpg", cyl = "cyl", disp = "disp", hp = "hp", drat = "drat", wt = "wt", qsec = "qsec", vs = "vs", am = "am", gear = "gear", carb = "carb", spare_tire = "spare_tire")

Value

Dataframe containing subset info

`drop_na_column_names` *Drop NA column names*

Description

Deletes columns whose name is NA or whose name is empty

Usage

```
drop_na_column_names(x)
```

Arguments

`x`	dataframe

Value

dataframe without columns that are NA

duplicates_in_column *Duplicates in column*

Description

Searches for duplicates in a data frame column.

Usage

```
duplicates_in_column(df, col)
```

Arguments

df	Data frame.
col	Column name.

Value

Rows containing duplicated values.

See Also

Other tests: check_double_columns(), check_no_duplicates_in_group(), check_numeric_or_integer_type(), check_posixct_type(), test_all_equal()

Examples

```
duplicates_in_column(mtcars, "mpg")
```

find_common_columns *Find Common Columns Between Data Frames*

Description

This function identifies common column names between multiple data frames. It takes a variable number of data frames as input and returns a character vector containing the common column names.

Usage

```
find_common_columns(...)
```

Arguments

...	A variable length list of data frames.

Value

A character vector of column names found in common between all data frames.

Examples

```
df1 <- data.frame(a = c(1, 2, 3), b = c(4, 5, 6))
df2 <- data.frame(a = c(7, 8, 9), b = c(10, 11, 12), c = c(13, 14, 15))
common_columns <- find_common_columns(df1, df2)
print(common_columns)
```

`find_maximum_value` *Find the maximum numeric value in a vector, ignoring non-numeric values*

Description

Find the maximum numeric value in a vector, ignoring non-numeric values

Usage

```
find_maximum_value(numeric_vector)
```

Arguments

`numeric_vector` A vector from which to find the maximum numeric value.

Value

The maximum numeric value in the input vector, or NA if none exist.

Examples

```
# Find the maximum of a numeric vector
find_maximum_value(c(3, 1, 4, 1, 5, 9)) # Returns   9

# Find the maximum of a mixed vector with non-numeric values
find_maximum_value(c(3, 1, 4, "two", 5, 9)) # Returns   9

# Attempt to find the maximum of a vector with only non-numeric values
find_maximum_value(c("one", "two", "three")) # Returns NA
```

find_minimum_value *Find the minimum numeric value in a vector, ignoring non-numeric values*

Description

Find the minimum numeric value in a vector, ignoring non-numeric values

Usage

```
find_minimum_value(numeric_vector)
```

Arguments

numeric_vector	A vector from which to find the minimum numeric value.

Value

The minimum numeric value in the input vector, or NA if none exist.

Examples

```
# Find the minimum of a numeric vector
find_minimum_value(c(3, 1, 4, 1, 5, 9)) # Returns   1

# Find the minimum of a mixed vector with non-numeric values
find_minimum_value(c(3, 1, 4, "two", 5, 9)) # Returns   1

# Attempt to find the minimum of a vector with only non-numeric values
find_minimum_value(c("one", "two", "three")) # Returns NA
```

find_pattern_r *Find pattern in R scripts*

Description

Function to search for a pattern in R scripts.

Usage

```
find_pattern_r(pattern, path = ".", case.sensitive = TRUE, comments = FALSE)
```

Arguments

pattern	Pattern to search
path	Directory to search in
case.sensitive	Whether pattern is case sensitive or not
comments	whether to search in commented lines

Value

Dataframe containing R script paths

`get_distribution_statistics` *Compute distribution statistics for a numeric vector*

Description

This function computes summary statistics such as quartiles, mean, and standard deviation for a numeric vector.

Usage

```
get_distribution_statistics(data_vector)
```

Arguments

`data_vector`	A numeric vector for which to compute summary statistics.

Value

A character string describing the summary statistics of the input vector.

Examples

```
# Compute summary statistics for a numeric vector
data_vector <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
get_distribution_statistics(data_vector)
```

`get_first_element_class` *Retrieve the class of the first element of a vector*

Description

Retrieve the class of the first element of a vector

Usage

```
get_first_element_class(input_vector)
```

Arguments

`input_vector`	A vector whose first element's class is to be retrieved.

Value

The class of the first element of the input vector.

Examples

```
# Get the class of the first element in a numeric vector
get_first_element_class(c(1, 2, 3)) # Returns "numeric"

# Get the class of the first element in a character vector
get_first_element_class(c("apple", "banana", "cherry")) # Returns "character"
```

---

get_values *Get values of column*

Description

A function to determine what kind of values are present in columns.

Usage

```
get_values(df, column)
```

Arguments

df	The dataframe
column	Column to get values from.

Value

The class of the column values

Examples

```
get_values(mtcars, "mpg")
```

identify_join_pairs *Identify Possible Join Pairs Between Data Frames*

Description

This function identifies potential join pairs between two data frames based on the overlap between the distinct values in their columns. It returns a data frame showing the possible join pairs.

Usage

```
identify_join_pairs(..., similarity_cutoff = 0.2)
```

Arguments

`...`	A list of two data frames.
`similarity_cutoff`	The minimal percentage of overlap between the distinct values in the columns.

Value

A data frame showing candidate join pairs.

Examples

```
identify_join_pairs(iris, iris3)
```

identify_outliers *Identify Outliers in a Data Frame Column*

Description

This function identifies outliers in a specified column of a data frame. It returns a tibble containing the unique values, tally, and whether it is an outlier or not.

Usage

```
identify_outliers(df, var)
```

Arguments

`df`	The data frame.
`var`	The column to check for outliers.

Value

A tibble containing the unique values, tally, and whether each value is an outlier or not.

Examples

```
df <- data.frame(a = c(1, 2, 3, 100, 101), b = c(4, 5, 6, 7, 8), c = c(7, 8, 9, 100, 101))
outliers <- identify_outliers(df, "a")
print(outliers)
```

`is_unique_column` *Check if a column in a dataframe has unique values*

Description

Check if a column in a dataframe has unique values

Usage

```
is_unique_column(column_name, data_frame)
```

Arguments

`column_name`	The name of the column to check for uniqueness.
`data_frame`	A dataframe containing the column to check.

Value

`TRUE` if the column has unique values, `FALSE` otherwise.

Examples

```
# Create a dataframe with a unique ID column
data_frame <- tibble::tibble(
  id = c(1, 2, 3, 4, 5),
  value = c("a", "b", "c", "d", "e")
)
is_unique_column("id", data_frame) # Returns TRUE

# Create a dataframe with duplicate values in the ID column
data_frame <- tibble::tibble(
  id = c(1, 2, 3, 4, 5, 1),
  value = c("a", "b", "c", "d", "e", "a")
)
is_unique_column("id", data_frame) # Returns FALSE
```

md_complete_cases *MD complete cases*

Description

Print the complete cases of the data.

Usage

```
md_complete_cases(data, digits = 1)
```

Arguments

data	The data frame.
digits	Default: 1. number of digits for rounding.

Value

Message with the number of rows, number of rows with missing values and the percentage of complete rows.

Examples

```
# example code
md_complete_cases(iris)

iris$Sepal.Length[5] <- NA_character_
md_complete_cases(iris)
```

regex_content_parameter *Construct Regex for Matching Function Parameter Content*

Description

This function constructs a regex pattern for matching the content of a parameter in a function. It uses the `base::paste0` function to construct the regex pattern.

Usage

```
regex_content_parameter(parameter)
```

Arguments

parameter	The parameter whose value is to be searched in a function.

Value

A regex pattern as a character string.

Examples

```
# Create a parameter name
parameter <- "my_parameter"
# Construct a regex pattern for matching the content of the parameter
pattern <- regex_content_parameter(parameter)
```

---

regex_time *Generate regular expression of a time.*

Description

This function generates a regular expression for time based on the input format.

Usage

```
regex_time(format = "hh:mm")
```

Arguments

format	The format of the time. Possible values are: • "hh:mm": to generate "09:05". • "h:m": to generate "9:5". • "hh:mm:ss": to generate "09:05:00". • "h:m:s": to generate "9:5:0". • "hh:mm:ss AM/PM": to generate "09:05:00 AM". • "h:m:s AM/PM": to generate "9:5:0 AM".

Value

A regular expression.

Examples

```
regex_time("hh:mm")
regex_time("h:m")
regex_time("hh:mm:ss")
regex_time("h:m:s")
regex_time("hh:mm:ss AM/PM")
regex_time("h:m:s AM/PM")
```

`regex_year_date` *Generate regular expression of a year date.*

Description

This function generates a regular expression for year date based on the input format.

Usage

```
regex_year_date(format = "yyyy")
```

Arguments

`format`	The format of the year date. Possible values are: • "yyyy": to generate "2023". • "yyyy-MM-dd": to generate "2023-09-29". • "yyyy/MM/dd": to generate "2023/09/29". • "yyyy.MM.dd": to generate "2023.09.29". • "yyyy-M-d": to generate "2023-9-29". • "yyyy/M/d": to generate "2023/9/29". • "yyyy.M.d": to generate "2023.9.29". • "yyyy-MM-dd HH:mm:ss": to generate "2023-09-29 12:34:56". • "yyyy/MM/dd HH:mm:ss": to generate "2023/09/29 12:34:56". • "yyyy-MM-dd HH:mm": to generate "2023-09-29 12:34". • "yyyy/MM/dd HH:mm": to generate "2023/09/29 12:34".

Value

A regular expression.

Examples

```
regex_year_date("yyyy")
regex_year_date("yyyy-MM-dd")
regex_year_date("yyyy/MM/dd")
regex_year_date("yyyy.MM.dd")
regex_year_date("yyyy-M-d")
regex_year_date("yyyy/M/d")
regex_year_date("yyyy.M.d")
regex_year_date("yyyy-MM-dd HH:mm:ss")
regex_year_date("yyyy/MM/dd HH:mm:ss")
regex_year_date("yyyy-MM-dd HH:mm")
regex_year_date("yyyy/MM/dd HH:mm")
```

remove_duplicates_and_na

Remove Duplicates and NA Values from Input

Description

This function removes duplicate values and NA values from the input. It first removes NA values from the input using the `na.omit` function from the `stats` package. Then it removes duplicate values from the result using the `unique` function.

Usage

```
remove_duplicates_and_na(input)
```

Arguments

`input`	A vector or data frame.

Value

A vector or data frame with duplicate values and NA values removed.

Examples

```
# Create a vector with duplicate values and NA values
input <- c(1, 2, NA, 2, NA, 3, 4, 4, NA, 5)
# Remove duplicate values and NA values
output <- remove_duplicates_and_na(input)
print(output)
```

retrieve_functions_and_packages

Retrieve functions and packages

Description

Retrieves functions and their corresponding packages used in a given script.

Usage

```
retrieve_functions_and_packages(path)
```

Arguments

`path`	The complete path of the script.

Value

Used_functions

`retrieve_function_calls` *retrieve_function_calls*

Description

retrieve_function_calls

Usage

```
retrieve_function_calls(script_name)
```

Arguments

`script_name`	The script to search functions in

Value

dataframe

`retrieve_package_usage` *Retrieve packages that are loaded in a script*

Description

Retrieve packages that are loaded in a script

Usage

```
retrieve_package_usage(script_name)
```

Arguments

`script_name`	The path to the R script

Value

dataframe

`retrieve_sourced_scripts` *retrieve_sourced_scripts*

Description

retrieve_sourced_scripts

Usage

```
retrieve_sourced_scripts(script_name)
```

Arguments

`script_name`	The main script to search

Value

dataframe

`retrieve_string_assignments` *retrieve_string_assignments*

Description

retrieve_string_assignments

Usage

```
retrieve_string_assignments(script_name)
```

Arguments

`script_name`	The script to search objects in

Value

dataframe

`return_assertions_message` *Return Assertion Messages*

Description

This function returns a message indicating whether an assertion test has passed or failed. An "assertion collection" from the checkmate package must be provided. The message can be returned as an error or a warning. For some assertions, only warnings are allowed, as an error would stop the script from running. This is done for the following assertions: percentage missing values, duplicates, subset, and set_equal.

Usage

```
return_assertions_message(
  collection,
  collection_name,
  fail = "stop",
  silent = FALSE,
  output_map = NULL
)
```

Arguments

`collection`	An object with the class "AssertCollection".
`collection_name`	The name of the collection. This name is mentioned in the messages.
`fail`	"stop" or "warning". If the assertions fail, an error is returned and the script output is stopped. If "warning", only a warning is returned.
`silent`	If FALSE (default), the success message is printed in the console. If TRUE, it is not shown.
`output_map`	A map, like 1. Read data, where the file is stored.

Value

The message indicating whether the assertion test has passed or failed.

return_mtcars_testfile *Read and return the mtcars testfile*

Description

Gets the modified rds dataset for testing assertions.

Usage

```
return_mtcars_testfile()
```

Value

returns mtcars_test dataframe

run_all_assertions *Run All Data Validation Assertions*

Description

This function performs multiple validation checks on a dataset using various assertion functions. It loads metadata from specified CSV files, validates the dataset against expected field properties, and stops execution if any warnings are encountered.

Usage

```
run_all_assertions(new_data, output_dir)
```

Arguments

`new_data`	A data frame. The dataset to validate.
`output_dir`	A character string. The directory containing metadata CSV files (`field_info.csv`, `numeric_details.csv`, `data_types.csv`).

Value

No return value. The function stops execution and displays warnings if any validation checks fail.

str_detect_in_file *Detect string in file*

Description

Detect string in file

Usage

```
str_detect_in_file(file, pattern, only_comments = FALSE, collapse = FALSE)
```

Arguments

file	Path to file.
pattern	Pattern to match.
only_comments	default FALSE. Whether to only search in commented lines.
collapse	default: FALSE: search file line by line. If true, then pattern is search in the entire file at once after collapsing. (only_comments does not work when collapse is set to TRUE)

Value

Boolean whether pattern exists in file.

test_all_equal *Test all equal*

Description

Test whether all values in a vector are equal.

Usage

```
test_all_equal(x, na.rm = FALSE)
```

Arguments

x	Vector to test.
na.rm	default: FALSE. exclude NAs from the test.

Value

Boolean result of the test

See Also

Other tests: check_double_columns(), check_no_duplicates_in_group(), check_numeric_or_integer_type(), check_posixct_type(), duplicates_in_column()

Examples

```
test_all_equal(c(5, 5, 5))

test_all_equal(c(5, 6, 3))
```

---

unique_id *unique id*

Description

Check if parsed variable is a unique identifier. This function was adapted from: Source: https://edwinth.github.io/blog/unique_

Usage

```
unique_id(x, ...)
```

Arguments

x	vector or dataframe.
...	optional variables, e.g. name of column or a vector of names.

Value

Boolean whether variable is a unique identifier.

Examples

```
unique_id(iris, Species)

mtcars$name <- rownames(mtcars)
unique_id(mtcars, name)
```

Index