Package: DQA (via r-universe)

May 20, 2026

Title Data Quality Assessment Tools

Version 0.1.1

Date 2026-04-18

Description In the context of data quality assessment, this package provides a number of functions for evaluating data quality across various dimensions, including completeness, plausibility, concordance, conformance, currency, timeliness, and correctness. It has been developed based on two well-known frameworks—Michael G. Kahn (2016) <doi:10.13063/2327-9214.1244> and Nicole G. Weiskopf (2017) <doi:10.5334/egems.218>—for data quality assessment. Using this package, users can evaluate the quality of their datasets, provided that corresponding metadata are available.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

Imports ggplot2, data.table

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Hamed Tabesh [aut], Elham Hosseinzadeh [aut, cre], Marziyeh Afkanpour [aut]

Maintainer Elham Hosseinzadeh <elham.hh2012@gmail.com>

Repository https://elhamhsnzadeh.r-universe.dev

Date/Publication 2026-04-20 09:21:50 UTC

RemoteUrl https://github.com/cran/DQA

RemoteRef HEAD

RemoteSha 42e4ab7da04fdca84d614dab86fee0367663cb3d

Contents

`check_missing_itemwise` *Check Missing Data Item-wise with Dependency Logic*

Description

Analyzes missing data ('NA' values) for each variable (item-wise) by considering dependencies between variables. This function goes beyond simple NA counts by classifying missingness into different categories based on rules defined in metadata.

Usage

```
check_missing_itemwise(
  S_data,
  M_data,
  var_select = 1:nrow(M_data),
  Show_Plot = FALSE
)
```

Arguments

`S_data`	A data frame containing the source data to be checked.
`M_data`	A metadata data frame containing the validation rules.
`var_select`	A numeric or character vector specifying which variables to process. Can be indices or names from the 'VARIABLE' column of 'M_data'. Defaults to all variables.
`Show_Plot`	A logical value. If 'TRUE', a ggplot bar chart showing the missingness percentage for each variable is displayed.

Details

This function classifies each row for a given variable into one of four states:

- **Completed:** The value is present where it is expected.
- **Missing:** The value is 'NA' where it was expected (based on a parent condition).
- **Jump:** The value is 'NA' because the parent condition was not met (i.e., the question was correctly skipped).
- **Unexpected:** The value is present where it was *not* expected (a data quality issue).

The metadata ('M_data') must contain the following columns to define the rules:

- **VARIABLE:** The name of the variable in the source data ('S_data') to be checked for missingness.
- **VARIABLE_Code:** A unique numeric or character code assigned to each variable for identification and dependency mapping.
- **Dependency:** Specifies the dependency of the variable on another variable. A value of '0' indicates no dependency, while other values indicate the 'VARIABLE_Code' of the parent variable.
- **Dep_Value:** The specific value or condition of the parent variable (as referenced in 'Dependency') that must be met for the current variable to be applicable. Use '"ANY"' if the value of the parent variable can be any non-missing value.

Value

A 'data.table' summarizing the missing data analysis for each variable, with columns such as 'VARIABLE', 'Missing_Count', 'Jump_Count', 'Unexpected_Count', 'Total_Applicable'(the variable's value was expected to be completed based on metadata rules.), 'Percent_Complete', and 'Percent_Missing'.

See Also

Other missing data checks: `check_missing_record()`, `check_missing_segments()`

Examples

```
# 1. Define comprehensive sample data and metadata
Meta_data <- data.frame(
  stringsAsFactors = FALSE,
  VARIABLE = c(
    "ID", "Gender", "Age", "Has_Job", "Job_Title",
    "Job_Satisfaction", "Last_Promotion_Year", "Has_Insurance",
    "Insurance_Provider", "Annual_Checkup"
  ),
  VARIABLE_Code = 1:10,
  Var_order = 1:10,
  Segment_Names = c(
    "Demographic", "Demographic", "Demographic", "Employment", "Employment",
    "Employment", "Employment", "Health", "Health", "Health"
  ),
```

```
  Dependency = c(0, 0, 0, 0, 4, 5, 5, 0, 8, 8),
  Dep_Value = c(
    "0", "0", "0", "0", "Yes", "ANY", "ANY", "0", "Yes", "Yes"
  )
)

Source_data <- data.frame(
  ID = 1:10,
  Gender = c("Male", "Female", "Male", "Female", "Male",
             "Female", "Male", "Female", "Male", "Female"),
  Age = c(25, 42, 31, 55, 29, 38, 45, 22, 60, 33),
  Has_Job = c(NA, "Yes", "No", "Yes", "Yes", "No", "Yes", "Yes", "No", "Yes"),
  Job_Title = c(NA, "Manager", NA, "Analyst", NA, "Student",
                "Director", "Engineer", NA, "Designer"),
  Job_Satisfaction = c(5, 9, NA, 8, 7, NA, 10, 9, NA, 6),
  Last_Promotion_Year = c(2020, 2021, NA, NA, NA, NA, 2024, 2022, NA, 2023),
  Has_Insurance = c("Yes", "No", "Yes", "Yes", "No", "Yes", "Yes", "No", "No", "Yes"),
  Insurance_Provider = c("Provider A", NA, "Provider B", "Provider C",
                         "Provider D", NA, "Provider E", NA, NA, "Provider F"),
  Annual_Checkup = c("Yes", NA, "No", "Yes", NA, "Yes", "Yes", "No", NA, "Yes")
)

# 2. Run the item-wise check with plot
item_report <- check_missing_itemwise(
  S_data = Source_data, M_data = Meta_data, Show_Plot = TRUE
)
print(item_report)
```

`check_missing_record` *Check Missing Data by Record (Unit Check)*

Description

Provides a high-level summary of data completeness across the entire dataset by classifying each row (or "record") as complete, incomplete, or missing.

Usage

```
check_missing_record(
  S_data,
  M_data,
  Show_Plot = FALSE,
  start_var = 1,
  skip_vars = NULL
)
```

Arguments

`S_data`	A data frame containing the source data to be checked.

M_data	A metadata data frame containing the validation rules.
Show_Plot	A logical value. If 'TRUE', a pie chart visualizing the proportions of complete, incomplete, and missing rows is displayed.
start_var	A numeric value indicating the starting variable index (from 'M_data') to include in the analysis. Defaults to 1.
skip_vars	A character or numeric vector of variables to exclude from the analysis. Can be variable names or column indices.

Details

This function evaluates all specified variables for each row and determines its overall status based on the same error logic as 'check_missing_segments'. A row is:

- **Complete:** The row has all values as non-missing for any variable within the each rows.
- **Incomplete:** The row has at least one 'NA' value for variables in the each rows.
- **Fully Missing:** All variables in the record are 'NA' for that row.

The metadata ('M_data') must contain the following columns to define the rules:

- **VARIABLE:** The name of the variable in the source data ('S_data') to be checked for missingness.
- **VARIABLE_Code:** A unique numeric or character code assigned to each variable for identification and dependency mapping.
- **Dependency:** Specifies the dependency of the variable on another variable. A value of '0' indicates no dependency, while other values indicate the 'VARIABLE_Code' of the parent variable.
- **Dep_Value:** The specific value or condition of the parent variable (as referenced in 'Dependency') that must be met for the current variable to be applicable. Use '"ANY"' if the value of the parent variable can be any non-missing value.

The function returns a single-row data frame summarizing the counts and percentages for the entire dataset.

Value

A single-row 'data.frame' with summary counts and percentages: 'Total_Rows', 'Complete_Count', 'Incomplete_Count', 'Missing_Count', 'Percent_Complete', 'Percent_Incomplete', and 'Percent_Missing'.

See Also

Other missing data checks: `check_missing_itemwise()`, `check_missing_segments()`

Examples

```
# 1. Define comprehensive sample data and metadata
Meta_data <- data.frame(
  stringsAsFactors = FALSE,
  VARIABLE = c(
    "ID", "Gender", "Age", "Has_Job", "Job_Title",
```

```
    "Job_Satisfaction", "Last_Promotion_Year", "Has_Insurance",
    "Insurance_Provider", "Annual_Checkup"
  ),
  VARIABLE_Code = 1:10,
  Var_order = 1:10,
  Segment_Names = c(
    "Demographic", "Demographic", "Demographic", "Employment", "Employment",
    "Employment", "Employment", "Health", "Health", "Health"
  ),
  Dependency = c(0, 0, 0, 0, 4, 5, 5, 0, 8, 8),
  Dep_Value = c(
    "0", "0", "0", "0", "Yes", "ANY", "ANY", "0", "Yes", "Yes"
  )
)

Source_data <- data.frame(
  ID = 1:10,
 Gender = c("Male", NA, "Male", "Female", "Male","Female", "Male", "Female", "Male", "Female"),
Age = c(25, NA, 31, 55, 29, 38, 45, 22, 60, 33),
Has_Job = c("Yes", NA, "No", "Yes", "Yes", "No", "Yes", "Yes", "No", "Yes"),
Job_Title = c(NA, NA, NA, "Analyst", NA, "Student","Director", "Engineer", NA, "Designer"),
Job_Satisfaction = c(5, NA, NA, 8, 7, NA, 10, 9, NA, 6),
Last_Promotion_Year = c(2020,NA , 2021, NA, NA, NA, 2024, 2022, NA, 2023),
Has_Insurance = c("Yes", NA, "Yes", "Yes", "No", "Yes", "Yes", "No", "No", "Yes"),
Insurance_Provider = c("Provider A", NA, "Provider B", "Provider C","Provider D", NA, "Provider E",
 NA, NA, "Provider F"),
Annual_Checkup = c("Yes", NA, "No", "Yes", NA, "Yes", "Yes", "No", NA, "Yes")
)
# 4. Run the row-wise check with plot
row_report <- check_missing_record(
  S_data = Source_data, M_data = Meta_data, skip_vars = "ID", Show_Plot = TRUE
)
print(row_report)
```

`check_missing_segments` *Check Missing Data by Segments*

Description

Analyzes data completeness at the segment level. A segment is a group of variables defined in the ‘Segment_Names‘ column of the metadata.

Usage

```
check_missing_segments(S_data, M_data, Show_Plot = FALSE)
```

Arguments

S_data	A data frame containing the source data to be checked.
M_data	A metadata data frame containing the validation rules, including a 'Segment_Names' column.
Show_Plot	A logical value. If 'TRUE', a stacked bar chart visualizing the proportions for each segment is displayed.

Details

For each segment, this function evaluates every row of the source data ('S_data') and classifies it into one of three categories:

- **Complete:** The row has all values as non-missing for any variable within the segment.
- **Incomplete:** The row has at least one 'NA' value for variables in the segment.
- **Fully Missing:** All variables belonging to the segment are 'NA' for that row.

The metadata ('M_data') must contain the following columns to define the rules:

- **VARIABLE:** The name of the variable in the source data ('S_data') to be checked for missingness.
- **VARIABLE_Code:** A unique numeric or character code assigned to each variable for identification and dependency mapping.
- **Dependency:** Specifies the dependency of the variable on another variable. A value of '0' indicates no dependency, while other values indicate the 'VARIABLE_Code' of the parent variable.
- **Dep_Value:** The specific value or condition of the parent variable (as referenced in 'Dependency') that must be met for the current variable to be applicable. Use '"ANY"' if the value of the parent variable can be any non-missing value.

The function returns a summary table with counts and percentages for each category per segment.

Value

A 'data.frame' summarizing the analysis for each segment, with columns: 'SEGMENT', 'Total_Rows', 'Complete_Count', 'Incomplete_Count', 'Missing_Count', 'Percent_Complete', 'Percent_Incomplete', and 'Percent_Missing'.

See Also

Other missing data checks: `check_missing_itemwise()`, `check_missing_record()`

Examples

```
# 1. Define comprehensive sample data and metadata
Meta_data <- data.frame(
  stringsAsFactors = FALSE,
  VARIABLE = c(
    "ID", "Gender", "Age", "Has_Job", "Job_Title",
    "Job_Satisfaction", "Last_Promotion_Year", "Has_Insurance",
```

```
    "Insurance_Provider", "Annual_Checkup"
  ),
  VARIABLE_Code = 1:10,
  Var_order = 1:10,
  Segment_Names = c(
    "Demographic", "Demographic", "Demographic", "Employment", "Employment",
    "Employment", "Employment", "Health", "Health", "Health"
  ),
  Dependency = c(0, 0, 0, 0, 4, 5, 5, 0, 8, 8),
  Dep_Value = c(
    "0", "0", "0", "0", "Yes", "ANY", "ANY", "0", "Yes", "Yes"
  )
)

Source_data <- data.frame(
  ID = 1:10,
Gender = c("Male", NA, "Male", "Female", "Male","Female", "Male", "Female", "Male", "Female"),
Age = c(25, NA, 31, 55, 29, 38, 45, 22, 60, 33),
Has_Job = c("Yes", NA, "No", "Yes", "Yes", "No", "Yes", "Yes", "No", "Yes"),
Job_Title = c(NA, NA, NA, "Analyst", NA, "Student","Director", "Engineer", NA, "Designer"),
Job_Satisfaction = c(5, NA, NA, 8, 7, NA, 10, 9, NA, 6),
Last_Promotion_Year = c(2020,NA , 2021, NA, NA, NA, 2024, 2022, NA, 2023),
Has_Insurance = c("Yes", NA, "Yes", "Yes", "No", "Yes", "Yes", "No", "No", "Yes"),
Insurance_Provider = c("Provider A", NA, "Provider B", "Provider C","Provider D", NA, "Provider E",
 NA, NA, "Provider F"),
Annual_Checkup = c("Yes", NA, "No", "Yes", NA, "Yes", "Yes", "No", NA, "Yes")
)
# 3. Run the segment check with plot
segment_report <- check_missing_segments(
  S_data = Source_data, M_data = Meta_data, Show_Plot = TRUE
)
print(segment_report)
```

---

concordance_check *Perform Concordance Check on Data Based on Defined Rules*

Description

This function evaluates a source dataframe (‘S_data‘) against a set of rules defined in a metadata dataframe (‘M_data‘). It Checks multiple concordance rules (logical and clinical conditions) on columns of a data frame, based on metadata specifications. Supports flexible rule definition, date handling, and customizable output.

Usage

```
concordance_check(
  S_data,
  M_data,
  Result = FALSE,
  show_column = NULL,
```

```
  date_parser_fun = smart_to_gregorian_vec,
  var_select = "all",
  verbose = FALSE
)
```

Arguments

S_data	data.frame. The source data in which rules will be evaluated. Each column may be referenced by the rules.
M_data	data.frame. Metadata describing variables and their concordance rules. Must include at least columns VARIABLE and Concordance_Rule. Optionally includes TYPE and Concordance_Error_Type.
Result	logical (default: FALSE). If TRUE, returns row-by-row evaluation results for each rule. If FALSE, returns a summary table for each rule.
show_column	character vector (default: NULL). Names of columns from S_data to include in the result when Result = TRUE. Ignored otherwise.
date_parser_fun	function (default: smart_to_gregorian_vec). Converting Persian dates to English,Function to convert date values or date literals to Date class. Must accept character vectors and return Date objects.
var_select	character, numeric, or "all" (default: "all"). Subset of variables (rules) to check. Can be a character vector of variable names, numeric vector of row indices in M_data, or "all" to run all rules.
verbose	logical (default: FALSE). If TRUE, prints diagnostic messages during rule processing and evaluation.

Details

The metadata data.frame (M_data) must contain at least the following columns:

- **VARIABLE**: The name of the variable in S_data to which the rule applies.
- **Concordance_Rule**: The logical or clinical rule (as a string) to be evaluated for each row.
- **TYPE**: The expected type of the variable (e.g., "numeric", "date", "character").
- **Concordance_Error_Type**: The error type for each rule will be reported in the summary output.Based on the importance and severity of the rule, it can include two options: "Warning" or "Error".

For each variable described in M_data, the function:

- Replaces any instance of the string "val" in the rule with the actual column name of the variable.
- Parses and detects any date literals in the rule and substitutes them with placeholders; these placeholders are converted to Date class using the provided date_parser_fun.
- Automatically converts any referenced data columns to the appropriate type (numeric, date, or character) based on the TYPE column in the metadata.
- Detects which columns from S_data are referenced in each rule and ensures they are available and correctly typed before evaluation.

- Evaluates the rule for each row of S_data, using vectorized evaluation for performance where possible, and falling back to row-wise evaluation if necessary (e.g., for rules that are not vectorizable, such as those using ifelse with NA logic).

The function supports flexible rule definitions, including conditions involving multiple columns,clinical rules, date comparisons, and custom logic using R expressions.

If Result = FALSE, the function returns a summary table for each rule, including counts and percentages of rows that meet or do not meet the condition, as well as the error type from the metadata.

If Result = TRUE, the function returns a data.frame with one column per rule/variable, each containing logical values (TRUE, FALSE, or NA) for every row, plus any extra columns from S_data listed in show_column.

Value

If Result = FALSE: a data.frame summary with columns:

- VARIABLE: Name of the variable/rule.
- Condition_Met: Number of rows where the rule is TRUE.
- Condition_Not_Met: Number of rows where the rule is FALSE.
- NA_Count: Number of rows with missing/indeterminate result.
- Total_Applicable: Number of non-NA rows.
- Total_Rows: Number of total rows.
- Percent_Met: Percentage of applicable rows meeting the condition.
- Percent_Not_Met: Percentage of applicable rows not meeting the condition.
- Concordance_Error_Type: Error type from metadata (if available).

Examples

```
# build the long rule in multiple short source lines to avoid >100 char Rd lines
rule_bp <- paste0(
  "(ifelse(is.na(val) | is.na(Systolic_BP2), NA, ",
  "(abs(val - Systolic_BP2) >= 15) & (val > 15 & Prescription_drug == '')))"
)

# Source data
S_data <- data.frame(
  National_code = c("123", "1456", "789","545","4454","554"),
  LastName = c("Aliyar","Johnson","Williams","Brown","Jones","Garcia"),
  VisitDate = c("2025-09-23", "2021-01-10", "2021-01-03","1404-06-28","1404-07-28",NA),
  Test_date = c("1404-07-01", "2021-01-09", "2021-01-14","1404-06-29","2025-09-19",NA),
 Certificate_validity = c("2025-09-21", "2025-01-12", "2025-02-11","1403-06-28","2025-09-19",NA),
  Systolic_BP1 = c(110, NA, 145, 125,114,NA),
  Systolic_BP2 = c(125, 150, NA, 110,100,NA),
  Prescription_drug= c("Atorvastatin", "Metformin", "Amlodipine",
    "Omeprazole", "Aspirin","Metoprolol"),
  Blood_type = c("A-","B+","AB","A+","O-","O+"),
  stringsAsFactors = FALSE
)
```

```
# META DATA (use the short-built rule)
M_data <- data.frame(
  VARIABLE = c("National_code", "Certificate_validity", "VisitDate",
               "Test_date","LastName","Systolic_BP1","Systolic_BP2",
               "Prescription_drug","Blood_type"),
  Concordance_Rule = c(
    "", "", "VisitDate<=Test_date", "Test_date-VisitDate < 7", "",
    rule_bp, "", "", ""
  ),
  TYPE=c("numeric","date","date","date","character",
         "numeric","numeric","character","character"),
  Concordance_Error_Type = c("type1",NA,"type2","type3",NA,NA,NA,NA,"type4"),
  stringsAsFactors = FALSE
)

# test call
result <- concordance_check(S_data = S_data, M_data = M_data, Result = TRUE,
show_column = c("National_code"))
print(result)
```

---

conformance_check *Perform Conformance Check on Data Based on Defined Rules*

Description

This function evaluates a source dataframe (‘S_data‘) against a set of rules defined in a metadata dataframe (‘M_data‘). It uses a set of default rule functions but can also use a user-provided file.

Usage

```
conformance_check(
  S_data,
  M_data,
  rule_file = NULL,
  na_as_error = FALSE,
  var_select = "all"
)
```

Arguments

S_data	A dataframe containing the source data to be checked.
M_data	A metadata dataframe that specifies the rules. It must contain the columns ‘VARIABLE‘, ‘Conformance_Rule‘, and ‘Value‘.
rule_file	The path to a custom R file where rule functions are defined. If ‘NULL‘ (default), the standard rule definitions file included with the‘DQA‘ package will be used. Instructions for using this file are available under the name ‘conformance_rules‘.

na_as_error	A logical value. If 'TRUE', 'NA' values in the source data are treated as errors (non-conformant). If 'FALSE' (default), they are ignored.
var_select	Character or integer vector of variables to check. Accepts variable names, column numbers, or a mix. Default is "all" (check all variables in M_data).

Details

The metadata ('M_data') for conformance_check must include:

- **VARIABLE:** The name of the column in 'S_data' to which the rule applies.
- **Conformance_Rule:** The name of the rule function to execute for the VARIABLE (must be defined in the rule file).
- **Value:** Rule parameters such as Allowed length of values,, allowed category values, or column names required for computational checks.

Value

A dataframe containing the results of the conformance check for each rule.

Examples

```
# 1. Create sample source data (S_data)
S_data <- data.frame(
  id = 1:10,
  national_id = c("1234567890", "0987654321", "123", NA, "1112223334",
                  "1234567890", "5556667778", "9998887770", "12345", "4445556667"),
  gender = c(1, 2, 1, 3, 2, 1, NA, 2, 1, 2), # 1=Male, 2=Female, 3=Error
  age = c(25, 40, 150, 33, -5, 65, 45, 29, 70, 55),
  part_a = c(10, 15, 20, 25, 30, 35, 40, 45, 50, 55),
  part_b = c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50),
  total_parts = c(15, 25, 35, 45, 55, 65, 75, 85, 94, 105), # one error in row 9
  stringsAsFactors = FALSE
)

# 2. Create sample metadata (M_data)
M_data <- data.frame(
  VARIABLE = c(
    "national_id",
    "national_id",
    "gender",
    "total_parts"
  ),
  Conformance_Rule = c(
    "length_check",
    "unique_check",
    "category_check",
    "arithmetic_check"
  ),
  Value = c(
    "10",                  # national_id length must be 10
    "",                    # unique
```

```
    "1 | 2",               # Allowed values for gender
    "part_a + part_b"      # Computational rule for total_parts
  ),
  stringsAsFactors = FALSE
)

# 3. Run the conformance check using the package's default rules
# Ensure the 'DQA' package is loaded before running
 conformance_results <- conformance_check(S_data = S_data, M_data = M_data)
 print(conformance_results)
```

`conformance_rules` *Default Conformance Check Rules (Reference)*

Description

The ‘DQA‘ package provides a suite of built-in rule functions for use with ‘conformance_check()‘. You can use these rule names directly in the ‘Conformance_Rule‘ column of your metadata dataframe.

Details

Most rules accept the column vector as ‘x‘, and some accept extra parameters (e.g., ‘val_num‘, ‘val_lit‘) parsed from the ‘Value‘ column of your metadata.

General Purpose Rules

`unique_check(x)` Checks for unique (non-duplicate) values.

`character_check(x)` Checks for the presence of alphabetic characters.

`email_check(x)` Checks if values are valid email addresses.

`numeric_check(x)` Checks if values can be interpreted as numbers.

`integer_check(x)` Checks if values are integers (whole numbers).

`length_check(x, val_num)` Checks if string length matches the given value (from the ‘Value‘ column).

`category_check(x, val_num, val_ops, val_lit)` Checks membership in allowed categories (numbers or literals).

`set_check(x, val_lit)` Checks if value is in a user-defined set (from ‘Value‘).

`not_null_check(x)` Checks if value is not missing (‘NA‘).

Numeric/Date Rules

`date_check(x, val_lit)` Checks if value is a valid date (optionally in specified format).

`regex_check(x, val_lit)` Checks values against a user-supplied regular expression.

Computational

`bmi_check(x, val_lit = NULL, val = NULL)` Checks a BMI column for consistency with the corresponding Weight and Height columns. Put your "Weight and Height" columns in the Value column in the metadata for calculation

The function reads the column names for weight and height dynamically from the `val_lit` argument, which is typically provided by the metadata (`M_data`) in the `Value` column as a comma- or pipe-separated string (for example, Weight,Height or Weight|Height; use the actual column names in your dataset).

The function automatically detects the unit for height (cm or m) based on the data, and allows for custom rounding digits and tolerance via the optional `val` argument (e.g., `val = c(1, 0.01)` for 1 decimal digit and 0.01 tolerance).

Returns a logical vector indicating which rows pass the BMI consistency check.

Notes

See the actual implementation of these functions in the package by obtaining a copy with ‘conformance_scaffold_rules()‘.

---

correctness_check *Validate Data Against Correctness Rules*

Description

This function validates a data frame against a set of correctness rules specified in another data frame. It allows for complex validation operations, comparison with reference data, and detailed reporting.

Usage

```
correctness_check(
  S_data,
  M_data,
  Result = FALSE,
  show_column = NULL,
  date_parser_fun = smart_to_gregorian_vec,
  golden_data = NULL,
  key_column = NULL,
  external_data = NULL,
  var_select = "all",
  batch_size = 1000,
  verbose = FALSE
)
```

Arguments

S_data	A data frame containing the data to be validated.
M_data	A data frame containing the validation rules. Must have at least the following columns: • VARIABLE: The name of the variable to validate (must match column names in S_data) • Correctness_Rule: The validation rule as an R expression (string) • TYPE: The data type of the variable ("date", "numeric", or other) • Correctness_Error_Type: (Optional) Classification of the error type
Result	Logical. If TRUE, returns the detailed results for each row in S_data. If FALSE (default), returns a summary of validation results.
show_column	Character vector. When Result=TRUE, specifies additional columns from S_data to include in the output.
date_parser_fun	Function to convert date strings to Date objects. Default is smart_to_gregorian_vec, which should handle various date formats including Jalali dates.
golden_data	Optional data frame or list containing reference data for validation. Accessible within rules via the GOLDEN variable.
key_column	Character string specifying the column name that links rows in S_data to corresponding rows in golden_data. Required when comparing individual rows with golden_data.
external_data	Optional list or data frame containing additional data for validation rules.
var_select	Character vector or numeric indices specifying which variables from M_data to validate. By default, it validates all variables.
batch_size	integer. Number of rows to process in each batch (for efficiency).
verbose	logical. If TRUE, prints progress messages.

Details

The function evaluates each rule specified in M_data against the corresponding data in S_data. Rules are R expressions written as strings, evaluated in an environment where:

- Variables from S_data are available directly by name
- val refers to the current variable being validated
- GOLDEN provides access to reference data (when golden_data is provided)

Type conversion is applied to variables in S_data based on the TYPE column in M_data:

- "date": Values are converted using the date_parser_fun
- "numeric": Values are converted to numeric
- Other types: No conversion is applied

Special handling for date comparisons is provided, including automatic wrapping of GOLDEN references when comparing dates.

Value

If `Result=FALSE` (default): A data frame with one row per validated variable containing:

- VARIABLE: Variable name
- Condition_Met: Count of rows meeting the condition
- Condition_Not_Met: Count of rows not meeting the condition
- NA_Count: Count of rows where validation produced NA
- Total_Applicable: Count of non-NA validation results
- Total_Rows: Total number of rows in S_data
- Percent_Met: Percentage of applicable rows meeting the condition
- Percent_Not_Met: Percentage of applicable rows not meeting the condition
- Error_Type: Value from Correctness_Error_Type column in M_data

If `Result=TRUE`: A data frame with one row per row in S_data, containing:

- One column per validated variable with logical values (TRUE/FALSE/NA)
- Any additional columns specified in show_column

Examples

```
Authorized_drug<-data.frame(
  Drug_ID = 1:10,
  Drug_Name = c("Atorvastatin", "Metformin", "Amlodipine", "Omeprazole", "Aspirin",
                "Levothyroxine", "Sertraline", "Pantoprazole", "Losartan", "ASA"),
  stringsAsFactors = FALSE
)

golde<-data.frame(
  National_code = c("123", "456", "789","545","4454","554","665"),
  LastName = c("Bahman","Johnson","Williams","Brown","Jones","Garcia","Miller"),
  Certificate_Expiry = c("1404-07-01", "2030-01-12", "2025-01-11",
  "1404-06-28","2025-09-19",NA,NA),
  Blood_type = c("A-","B+","AB","A+","O-","O+","AB-"),
  stringsAsFactors = FALSE
)

S_data <- data.frame(
  National_code = c("123", "1456", "789","545","4454","554"),
  LastName = c("Aliyar","Johnson","Williams","Brown","Jones","Garcia"),
  VisitDate = c("2025-09-23", "2021-01-10", "2021-01-03","1404-06-28","1404-07-28",NA),
  Test_date = c("1404-07-01", "2021-01-09", "2021-01-14","1404-06-29","2025-09-19",NA),
 Certificate_validity = c("2025-09-23", "2025-01-12", "2025-02-11","1403-06-28","2025-09-19",NA),
  Systolic_Reading1 = c(110, NA, 145, 125,114,NA),
  Systolic_Reading2 = c(125, 150, NA, 110,100,NA),
  Prescription_drug= c("Atorvastatin", "Metformin", "Amlodipine",
   "Omeprazole", "Aspirin","Metoprolol"),
  Blood_type = c("A-","B+","AB","A+","O-","O+"),
  Height = c(178,195,165,NA,155,1.80),
  stringsAsFactors = FALSE
```

```
)

M_data <- data.frame(
  VARIABLE = c("National_code", "Certificate_validity", "VisitDate","Test_date",
               "LastName","Systolic_Reading1","Systolic_Reading2",
               "Prescription_drug","Blood_type","Height"),
  Correctness_Rule = c(
    "National_code %in% GOLDEN$National_code",
    "val <= GOLDEN$Certificate_Expiry",
    "((val >= '1404-06-01' & val <= '1404-06-31') | val == as.Date('2021-01-02'))",
    "val != VisitDate",
    "val %in% GOLDEN$LastName",
    "",
    "",
    "val %in% Authorized_drug$Drug_Name",
    "val %in% GOLDEN$Blood_type",
    ""),
  TYPE=c("numeric","date","date","date","character","numeric",
  "numeric","character","character","numeric"),
  Correctness_Error_Type=c("Error",NA,"Warning","Error",NA,NA,NA,NA,"Error","Warning"),
  stringsAsFactors = FALSE
)

result <- correctness_check(
  S_data = S_data,
  M_data = M_data,
  golden_data = golde,
  key_column = c("National_code"),
  Result =FALSE,
  external_data = Authorized_drug
)

print(result)
#
result <- correctness_check(
  S_data = S_data,
  M_data = M_data,
  golden_data = golde,
  #key_column = c("National_code"),#If you do not select a key, you can use Gold Data as a
  #list and your logical rules will be NA.
  Result =TRUE,
  external_data = Authorized_drug
)
print(result)
```

---

currency_check *Perform Currency Check for Data Frame Columns*

Description

This function evaluates a source dataframe (‘S_data‘) against a set of rules defined in a metadata dataframe (‘M_data‘). it Checks currency rules (Temporal conditions) on columns of a data frame, based on metadata specifications. Supports flexible rule definition,date literal handling, and customizable output.

Usage

```
currency_check(
  S_data,
  M_data,
  Result = FALSE,
  show_column = NULL,
  date_parser_fun = smart_to_gregorian_vec,
  var_select = "all",
  verbose = FALSE
)
```

Arguments

S_data	data.frame. The source data in which rules will be evaluated. Each column may be referenced by the rules.
M_data	data.frame. Metadata describing variables and their currency rules. Must include columns VARIABLE, Currency_Rule and TYPE. Optionally includes Currency_Error_Type.
Result	logical (default: FALSE). If TRUE, returns row-by-row evaluation results for each rule. If FALSE, returns a summary table for each rule.
show_column	character vector (default: NULL). Names of columns from S_data to include in the result when Result = TRUE. Ignored otherwise.
date_parser_fun	function (default: smart_to_gregorian_vec). Converting Persian dates to English,Function to convert date values or date literals to Date class. Must accept character vectors and return Date objects.
var_select	character, numeric, or "all" (default: "all"). Subset of variables (rules) to check. Can be a character vector of variable names, numeric vector of row indices in M_data, or "all" to run all rules.
verbose	logical (default: FALSE). If TRUE, prints diagnostic messages during rule processing and evaluation.

Details

The metadata data.frame (M_data) **must** contain the following columns:

- **VARIABLE**: Name of the variable in S_data to which the rule applies.
- **Currency_Rule**: A logical rule provided as a string that defines temporal (date/time) conditions to be evaluated.
- **TYPE**: Specifies the type of the variable (e.g., "numeric", "date", "character").

- **Currency_Error_Type**: The error type for each rule will be reported in the summary output.Based on the importance and severity of the rule, it can include two options: "Warning" or "Error".

For each variable described in `M_data`, the function:

- Preprocesses the rule: replaces 'val' with the variable name, parses date literals and substitutes them with placeholders.
- Converts referenced data columns to appropriate types (numeric, date) based on metadata.
- Evaluates the rule for each row, either vectorized or row-wise if needed.

If `Result = FALSE`, returns a summary table with counts and percentages of rows meeting/not meeting each condition. If `Result = TRUE`, returns a data.frame with boolean results for each rule, optionally including selected columns from the source data.

Value

If `Result = FALSE`: a data.frame summary with columns:

- VARIABLE: Name of the variable/rule.
- Condition_Met: Number of rows where the rule is TRUE.
- Condition_Not_Met: Number of rows where the rule is FALSE.
- NA_Count: Number of rows with missing/indeterminate result.
- Total_Applicable: Number of non-NA rows.
- Total_Rows: Number of total rows.
- Percent_Met: Percentage of applicable rows meeting the condition.
- Percent_Not_Met: Percentage of applicable rows not meeting the condition.
- Currency_Error_Type: Error type from metadata (if available).

If `Result = TRUE`: a data.frame with one column per rule (variable), each containing logical values for every row, plus optional columns from the source data.

Examples

```
# Source data
S_data <- data.frame(
  VisitDate = c("2025-09-23", "2021-01-10", "2021-01-03","1404-06-28","1404-07-28",NA),
  Test_date = c("1404-07-01", "2021-01-09", "2021-01-14","1404-06-29","2025-09-19",NA)
)

# META DATA
M_data <- data.frame(
  VARIABLE = c("VisitDate","Test_date"),
  Currency_Rule = c(
    "",
    "VisitDate<=Test_date",
    " Test_date-VisitDate <10 ",
    ""),
  TYPE=c("date","date"),
```

```
  Currency_Error_Type = c("Error","warning"),
  stringsAsFactors = FALSE
)

result <- currency_check(
  S_data = S_data,
  M_data = M_data,
  Result = TRUE,
  show_column = FALSE
)

print(result)

result <- currency_check(
  S_data = S_data,
  M_data = M_data,
  Result = FALSE,
  var_select = c("VisitDate","Test_date")
)

print(result)
```

`plausible_check` *Perform plausibility Check for Data Frame Columns*

Description

This function evaluates Plausibility Rule Checking for Data Frame Columns It Checks logical and clinical rules on columns of a data frame, based on metadata specifications. Supports flexible rule definition, check logical range of variables, and customizable output.

Usage

```
plausible_check(
  S_data,
  M_data,
  Result = FALSE,
  show_column = NULL,
  date_parser_fun = smart_to_gregorian_vec,
  var_select = "all",
  verbose = FALSE
)
```

Arguments

`S_data`	data.frame. The source data in which rules will be evaluated. Each column may be referenced by the rules.

M_data	data.frame. Metadata describing variables and their plausibility rules. Must include at least columns VARIABLE , Plausible_Rule, TYPE and Plausible_Error_Type.
Result	logical (default: FALSE). If TRUE, returns row-by-row evaluation results for each rule. If FALSE, returns a summary table for each rule.
show_column	character vector (default: NULL). Names of columns from S_data to include in the result when Result = TRUE. Ignored otherwise.
date_parser_fun	function (default: smart_to_gregorian_vec). Converting Persian dates to English,Function to convert date values or date literals to Date class. Must accept character vectors and return Date objects.
var_select	character, numeric, or "all" (default: "all"). Subset of variables (rules) to check. Can be a character vector of variable names, numeric vector of row indices in M_data, or "all" to run all rules.
verbose	logical (default: FALSE). If TRUE, prints diagnostic messages during rule processing and evaluation.

Details

The metadata data.frame (M_data) must contain at least the following columns:

- **VARIABLE**: The name of the variable in S_data to which the rule applies.
- **Plausible_Rule**: The logical rule (as a string) to be evaluated for each row.
- **TYPE**: The expected type of the variable (e.g., "numeric", "date", "character").
- **Plausible_Error_Type**: The error type for each rule will be reported in the summary output.Based on the importance and severity of the rule, it can include two options: "Warning" or "Error".

For each variable described in M_data, the function:

- Replaces any instance of the string "val" in the rule with the actual column name of the variable.
- Parses and detects any date literals in the rule and substitutes them with placeholders; these placeholders are converted to Date class using the provided date_parser_fun.
- Automatically converts any referenced data columns to the appropriate type (numeric, date, or character) based on the TYPE column in the metadata.
- Detects which columns from S_data are referenced in each rule and ensures they are available and correctly typed before evaluation.
- Evaluates the rule for each row of S_data, using vectorized evaluation for performance where possible, and falling back to row-wise evaluation if necessary (e.g., for rules that are not vectorizable, such as those using ifelse with NA logic).

The function supports flexible rule definitions, including conditions involving multiple columns, and custom logic using R expressions.

If Result = FALSE, the function returns a summary table for each rule, including counts and percentages of rows that meet or do not meet the condition, as well as the error type from the metadata if present.

If Result = TRUE, the function returns a data.frame with one column per rule/variable, each containing logical values (TRUE, FALSE, or NA) for every row, plus any extra columns from S_data listed in show_column.

Value

If Result = FALSE: a data.frame summary with columns:

- VARIABLE: Name of the variable/rule.
- Condition_Met: Number of rows where the rule is TRUE.
- Condition_Not_Met: Number of rows where the rule is FALSE.
- NA_Count: Number of rows with missing/indeterminate result.
- Total_Applicable: Number of non-NA rows.
- Total_Rows: Number of total rows.
- Percent_Met: Percentage of applicable rows meeting the condition.
- Percent_Not_Met: Percentage of applicable rows not meeting the condition.
- Plausible_Error_Type: Error type from metadata (if available).

Examples

```
# Source data
S_data <- data.frame(
  National_code = c("123", "1456", "789","545","4454","554"),
  LastName = c("Aliyar","Johnson","Williams","Brown","Jones","Garcia"),
  VisitDate = c("2025-09-23", "2021-01-10", "2021-01-03","1404-06-28","1404-07-28",NA),
  Test_date = c("1404-07-01", "2021-01-09", "2021-01-14","1404-06-29","2025-09-19",NA),
 Certificate_validity = c("2025-09-21", "2025-01-12", "2025-02-11","1403-06-28","2025-09-19",NA),
  DiastolicBP = c(110, NA, 145, 125,114,NA),
  SystolicBP = c(125, 150, NA, 110,100,NA),
  Prescription_drug= c("Atorvastatin", "Metformin", "Amlodipine",
  "Omeprazole", "Aspirin","Metoprolol"),
  Blood_type = c("A-","B+","AB","A+","O-","O+"),
  stringsAsFactors = FALSE
)

# META DATA
M_data <- data.frame(
  VARIABLE = c("National_code", "Certificate_validity", "VisitDate",
               "Test_date","LastName","DiastolicBP","SystolicBP",
               "Prescription_drug","Blood_type"),
  Plausible_Rule = c(
    "val<=123",
    "",
    "",
    "",
    "",
    "val < 40 | val > 145",
    "val < 50 | val > 230",
    "",
```

```
    ""),
  TYPE=c("numeric","date","date","date","character",
         "numeric","numeric","character","character"),
 Plausible_Error_Type = c("warning",NA,"Error","warning",NA,"warning","warning",NA,"Error"),
  stringsAsFactors = FALSE
)

result <- plausible_check(
  S_data = S_data,
  M_data = M_data,
  Result = TRUE,
  show_column = c("National_code")
)

print(result)

result <- plausible_check(
  S_data = S_data,
  M_data = M_data,
  Result = FALSE,
  var_select = c("DiastolicBP","DiastolicBP")
)

print(result)
```

`timeliness_check` *Perform timeliness Check for Data Frame Columns*

Description

This function evaluates a source dataframe (‘S_data‘) against a set of rules defined in a metadata dataframe (‘M_data‘). it Checks timeliness rules (Temporal and availability conditions) on columns of a data frame, based on metadata specifications. Supports flexible rule definition,date literal handling, and customizable output.

Usage

```
timeliness_check(
  S_data,
  M_data,
  Result = FALSE,
  show_column = NULL,
  date_parser_fun = smart_to_gregorian_vec,
  var_select = "all",
  verbose = FALSE
)
```

Arguments

S_data	data.frame. The source data in which rules will be evaluated. Each column may be referenced by the rules.
M_data	data.frame. Metadata describing variables and their timeliness rules. Must include columns VARIABLE, Timeliness_Rule and TYPE. Optionally includes Timeliness_Error_Type.
Result	logical (default: FALSE). If TRUE, returns row-by-row evaluation results for each rule. If FALSE, returns a summary table for each rule.
show_column	character vector (default: NULL). Names of columns from S_data to include in the result when Result = TRUE. Ignored otherwise.
date_parser_fun	function (default: smart_to_gregorian_vec). Converting Persian dates to English,Function to convert date values or date literals to Date class. Must accept character vectors and return Date objects.
var_select	character, numeric, or "all" (default: "all"). Subset of variables (rules) to check. Can be a character vector of variable names, numeric vector of row indices in M_data, or "all" to run all rules.
verbose	logical (default: FALSE). If TRUE, prints diagnostic messages during rule processing and evaluation.

Details

The metadata data.frame (M_data) **must** contain the following columns:

- **VARIABLE**: Name of the variable in S_data to which the rule applies.
- **Timeliness_Rule**: A logical rule provided as a string that defines temporal (date/time) conditions to be evaluated.
- **TYPE**: Specifies the type of the variable (e.g., "numeric", "date", "character").
- **Timeliness_Error_Type**: The error type for each rule will be reported in the summary output.Based on the importance and severity of the rule, it can include two options: "Warning" or "Error".

For each variable described in M_data, the function:

- Preprocesses the rule: replaces 'val' with the variable name, parses date literals and substitutes them with placeholders.
- Converts referenced data columns to appropriate types (numeric, date) based on metadata.
- Evaluates the rule for each row, either vectorized or row-wise if needed.

If Result = FALSE, returns a summary table with counts and percentages of rows meeting/not meeting each condition. If Result = TRUE, returns a data.frame with boolean results for each rule, optionally including selected columns from the source data.

Value

If `Result = FALSE`: a data.frame summary with columns:

- VARIABLE: Name of the variable/rule.
- Condition_Met: Number of rows where the rule is TRUE.
- Condition_Not_Met: Number of rows where the rule is FALSE.
- NA_Count: Number of rows with missing/indeterminate result.
- Total_Applicable: Number of non-NA rows.
- Total_Rows: Number of total rows.
- Percent_Met: Percentage of applicable rows meeting the condition.
- Percent_Not_Met: Percentage of applicable rows not meeting the condition.
- Timeliness_Error_Type: Error type from metadata (if available).

If `Result = TRUE`: a data.frame with one column per rule (variable), each containing logical values for every row, plus optional columns from the source data.

Examples

```
# Source data
S_data <- data.frame(
  VisitDate = c("2025-09-23", "2021-01-10", "2021-01-03","1404-06-28","1404-07-28",NA),
  Test_date = c("1404-07-01", "2021-01-09", "2021-01-14","1404-06-29","2025-09-19",NA)
)

# META DATA
M_data <- data.frame(
  VARIABLE = c("VisitDate","Test_date"),
  Timeliness_Rule = c(
    "",
    "VisitDate<=Test_date",
    " Test_date-VisitDate <10 ",
    ""),
  TYPE=c("date","date"),
  Timeliness_Error_Type = c("Error","warning"),
  stringsAsFactors = FALSE
)

result <- timeliness_check(
  S_data = S_data,
  M_data = M_data,
  Result = TRUE,
  show_column = FALSE
)

print(result)

result <- timeliness_check(
  S_data = S_data,
  M_data = M_data,
```

```
  Result = FALSE,
  var_select = c("VisitDate","Test_date")
)

print(result)
```

Index