Package 'DataAudit'

August 6, 2026

Type Package

Title Comprehensive Data Quality Auditing and Validation

Version 0.1.0

Description Provides tools for systematic assessment, validation, and reporting of data quality. The package detects common data-quality problems including missing or blank values, duplicate observations, infinite values, constant and near-zero variance variables, outliers, invalid ranges and categories, type and pattern violations, problematic dates and identifiers, sequence errors, dependency violations, and cross-variable inconsistencies. It also supports reusable validation rules, integrated audit reports, and standardized data-quality scoring for reproducible data-quality assessment workflows.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 8.0.0

Suggests knitr, rmarkdown, testthat (>= 3.0.0), tibble

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/vinodhpmd/DataAudit

BugReports https://github.com/vinodhpmd/DataAudit/issues

NeedsCompilation no

Author Vinodh Kumar Obli Rajendran [aut, cre] (ORCID: <https://orcid.org/0000-0002-7232-4122>), Keerthi Aaradhana [aut]

Maintainer Vinodh Kumar Obli Rajendran <vinodhkumar.rajendran@gmail.com>

Repository CRAN

Date/Publication 2026-08-06 12:40:02 UTC

Contents

`apply_audit_rules` *Apply Data Audit Rules*

Description

Applies a set of validation rules created by `audit_rules` to a data frame.

Usage

```
apply_audit_rules(data, rules)
```

Arguments

data	A data.frame.
rules	An object of class DataAuditRules created by audit_rules.

Details

The function evaluates range, category, uniqueness, required-value, and regular-expression rules.

Rules referring to variables that are not present in data produce an error.

Value

An object of class DataAuditRuleResult.

Examples

```
dat <- data.frame(
  Animal_ID = c("A001", "A002", "A002", NA),
  Species = c("Dog", "Cat", "Horse", "Dog"),
  Age = c(5, 3, 40, 2),
  stringsAsFactors = FALSE
)

rules <- audit_rules(
  range = list(
    Age = c(0, 30)
  ),
  category = list(
    Species = c("Dog", "Cat", "Cattle")
  ),
  unique = "Animal_ID",
  required = "Animal_ID",
  pattern = list(
    Animal_ID = "^A[0-9]+$"
  )
)

apply_audit_rules(dat, rules)
```

---

audit_data *Perform a General Data Quality Audit*

Description

Performs a general audit of a data frame and summarizes common data-quality problems including missing values, blank values, infinite values, constant variables, near-zero variance variables, and duplicated rows.

Usage

```
audit_data(data, include_nzv = TRUE, rules = NULL)
```

Arguments

data	A data.frame.
include_nzv	Logical. If TRUE, near-zero variance variables are included in the audit. Default is TRUE.
rules	Optional object of class DataAuditRules created by audit_rules. When supplied, the rules are evaluated and included in the audit report.

Details

audit_data() is intended as a general first-pass audit requiring minimal user configuration.

More specialized checks such as ranges, categories, patterns, identifiers, dates, dependencies, and cross-variable consistency can be performed using the corresponding DataAudit functions.

Custom validation rules created with audit_rules can be supplied through rules. These rules are evaluated using apply_audit_rules.

The returned object contains:

- overview: basic dataset information.
- variables: variable-level audit summary.
- duplicates: duplicated-row diagnostics.
- summary: overall audit statistics.
- rules: custom rule results, or NULL when no rules are supplied.

Value

An object of class DataAuditReport.

Examples

```
dat <- data.frame(
  ID = c(1, 2, 2, 4),
  Species = c("Dog", "", "", "Cat"),
  Value = c(10, NA, Inf, 20),
  Constant = 1
)

audit_data(dat)

rules <- audit_rules(
  range = list(
    Value = c(0, 100)
  ),
  required = "Species"
)
```

```
audit_data(
  dat,
  rules = rules
)
```

`audit_rules` *Create Data Audit Rules*

Description

Creates a structured collection of user-defined data validation rules for use with DataAudit.

Usage

```
audit_rules(
  range = NULL,
  category = NULL,
  unique = NULL,
  required = NULL,
  pattern = NULL
)
```

Arguments

`range`	Named list. Each element specifies the acceptable numeric range for a variable as a numeric vector of length two.
`category`	Named list. Each element contains the allowed values for a variable.
`unique`	Character vector containing variables that should contain unique values.
`required`	Character vector containing variables that should not contain missing values.
`pattern`	Named list. Each element specifies a regular expression that values of a variable should satisfy.

Details

`audit_rules()` defines validation requirements but does not evaluate a dataset. The resulting object can be supplied to functions that apply DataAudit rules.

Variable names are not checked against a dataset when the rule object is created because the same rule specification may be reused across multiple datasets.

Value

An object of class `DataAuditRules`.

Examples

```
rules <- audit_rules(
  range = list(
    Age = c(0, 30),
    Weight = c(0, 100)
  ),
  category = list(
    Species = c("Dog", "Cat", "Cattle")
  ),
  unique = "Animal_ID",
  required = c("Animal_ID", "Species"),
  pattern = list(
    Animal_ID = "^A[0-9]+$"
  )
)

rules
```

---

audit_score *Calculate a Data Quality Score*

Description

Calculates a standardized data-quality score from a `DataAuditReport` produced by `audit_data`.

Usage

```
audit_score(x)
```

Arguments

`x`	An object of class `DataAuditReport`.

Details

The score ranges from 0 to 100, where higher values indicate better data quality.

The score is based on automatic audit issues and custom rule violations relative to the number of observations and variables in the audited dataset.

Value

An object of class `DataAuditScore`.

Examples

```
dat <- data.frame(
  ID = c(1, 2, 2, 4),
  Value = c(10, NA, Inf, 20)
)

report <- audit_data(dat)
audit_score(report)
```

`blank_check` *Check Blank Values*

Description

Identifies blank values in character and factor variables.

Usage

```
blank_check(data, trim = TRUE)
```

Arguments

`data`	A data.frame.
`trim`	Logical. If TRUE, strings containing only whitespace are treated as blank. Default is TRUE.

Details

Blank values are evaluated only for character and factor variables. Missing values (`NA`) are reported separately and are not counted as blank values.

When `trim = TRUE`, leading and trailing whitespace is removed before checking whether a value is blank. Therefore values such as `" "`, `"  "`, and `"\t"` are classified as blank.

Value

A data.frame of class `BlankCheck` containing variable-level blank-value diagnostics.

Examples

```
dat <- data.frame(
  name = c("Dog", "", "Cat", "   ", NA),
  value = 1:5,
  stringsAsFactors = FALSE
)

blank_check(dat)

blank_check(dat, trim = FALSE)
```

case_check *Check Case Consistency*

Description

Detects values in a character or factor variable that differ only in letter case.

Usage

```
case_check(data, variable)
```

Arguments

data	A data.frame.
variable	A single character string specifying the variable to check.

Details

Values are compared after conversion to lower case. Therefore, values such as "Dog", "dog", and "DOG" are treated as belonging to the same normalized category.

A value is flagged when more than one distinct original representation exists for the same normalized value.

Missing values are reported separately and are not classified as case inconsistencies.

The function does not modify the original data.

Value

A data.frame of class CaseCheck containing row-level case-consistency diagnostics.

Examples

```
dat <- data.frame(
  Species = c("Dog", "dog", "DOG", "Cat", "Cat", NA)
)

case_check(
  dat,
  variable = "Species"
)
```

`category_check` *Check Categorical Values*

Description

Checks whether values in a variable belong to a user-defined set of allowed categories.

Usage

```
category_check(data, variable, allowed)
```

Arguments

`data`	A data.frame.
`variable`	A single character string specifying the variable to check.
`allowed`	A non-empty vector containing the allowed values.

Details

Missing values are reported separately and are not classified as invalid categories.

The supplied `allowed` values are compared with the observed values using their underlying values. Character, factor, numeric, integer, and logical variables are supported.

Value

A data.frame of class `CategoryCheck` containing row-level category diagnostics.

Examples

```
dat <- data.frame(
  Species = c("Dog", "Cat", "Horse", NA),
  stringsAsFactors = FALSE
)

category_check(
  dat,
  variable = "Species",
  allowed = c("Dog", "Cat", "Cattle")
)
```

`consistency_check` *Check Consistency Between Two Variables*

Description

Checks whether values in two variables satisfy a specified logical relationship.

Usage

```
consistency_check(data, variable1, variable2, relationship)
```

Arguments

`data`	A data.frame.
`variable1`	A single character string specifying the first variable.
`variable2`	A single character string specifying the second variable.
`relationship`	A character string specifying the expected relationship between the variables. Supported relationships are `"<"`, `"<="`, `">"`, `">="`, `"=="`, and `"!="`.

Details

The function evaluates the relationship as `variable1` `relationship` `variable2`.

For example, when checking whether an end date occurs on or after a start date, use:

`variable1` = `"EndDate"`, `variable2` = `"StartDate"`, and `relationship` = `">="`.

Rows containing a missing value in either variable are reported separately and are not classified as consistent.

The two variables must have compatible types for the requested comparison.

Value

A data.frame of class `ConsistencyCheck` containing row-level consistency diagnostics.

Examples

```
dat <- data.frame(
  StartDate = as.Date(c(
    "2026-01-01",
    "2026-02-01",
    "2026-03-01"
  )),
  EndDate = as.Date(c(
    "2026-01-10",
    "2026-01-20",
    "2026-03-15"
  ))
)
```

```
consistency_check(
  dat,
  variable1 = "EndDate",
  variable2 = "StartDate",
  relationship = ">="
)
```

`constant_check` *Check Constant Variables*

Description

Identifies variables that contain no meaningful variation.

Usage

```
constant_check(data)
```

Arguments

`data`	A data.frame.

Details

A variable is classified as constant when it contains one or fewer unique non-missing values.

Missing values are excluded when counting unique values. Consequently, a variable containing only missing values is also classified as constant.

Value

A data.frame of class `ConstantCheck` containing the number of unique non-missing values, missing values, and constant-variable status for each variable.

Examples

```
dat <- data.frame(
  id = 1:5,
  group = rep("A", 5),
  value = c(10, 10, 10, NA, 10)
)

constant_check(dat)
```

data_overview *Data Overview*

Description

Provides a concise overview of the size, structure, completeness, and variable types of a dataset.

Usage

```
data_overview(data)
```

Arguments

data	A data.frame.

Details

The function reports the number of rows and variables, total number of cells, missing and complete cells, completeness percentage, complete cases, duplicate rows, and counts of common variable types.

Value

An object of class `DataOverview` containing dataset-level summary information.

Examples

```
data_overview(iris)

dat <- iris
dat$Sepal.Length[1:5] <- NA
data_overview(dat)
```

date_check *Check Date Values*

Description

Checks a Date or POSIXct variable for missing values, dates outside user-defined limits, and optionally future dates.

Usage

```
date_check(
  data,
  variable,
  min_date = NULL,
  max_date = NULL,
  allow_future = TRUE
)
```

Arguments

data	A data.frame.
variable	A single character string specifying the Date or POSIXct variable to check.
min_date	Optional minimum acceptable date.
max_date	Optional maximum acceptable date.
allow_future	Logical. If FALSE, dates later than the current date are flagged. Default is TRUE.

Details

The selected variable must inherit from `Date` or `POSIXct`.

Missing values are reported separately and are not classified as out-of-range dates.

When `allow_future = FALSE`, observations later than the current date are flagged in `Future`.

The `min_date` and `max_date` arguments may be supplied as Date, POSIXct, or single character strings in ISO format (`YYYY-MM-DD`).

Value

A data.frame of class `DateCheck` containing row-level date diagnostics.

Examples

```
dat <- data.frame(
  Sample_Date = as.Date(c(
    "2025-01-01",
    "2025-06-01",
    "2026-01-01",
    NA
  ))
)

date_check(
  dat,
  variable = "Sample_Date",
  min_date = "2025-01-01",
  max_date = "2025-12-31"
)
```

dependency_check *Check Conditional Variable Dependency*

Description

Checks whether a target variable satisfies a requirement when a specified condition is met in another variable.

Usage

```
dependency_check(
  data,
  condition_variable,
  condition_value,
  target_variable,
  allow_blank = FALSE
)
```

Arguments

data	A data.frame.
condition_variable	A single character string specifying the variable defining the condition.
condition_value	The value of condition_variable that activates the dependency rule.
target_variable	A single character string specifying the variable that is required when the condition is met.
allow_blank	Logical. If FALSE, blank character values in the target variable are treated as missing. Default is FALSE.

Details

The dependency rule is interpreted as:

If condition_variable == condition_value, then target_variable must contain an acceptable value.

Missing values in the condition variable do not activate the dependency rule.

When allow_blank = FALSE, empty strings and strings containing only whitespace are treated as unavailable target values.

The function does not modify the original data.

Value

A data.frame of class DependencyCheck containing row-level dependency diagnostics.

Examples

```
dat <- data.frame(
  Disease_Status = c(
    "Positive",
    "Negative",
    "Positive",
    "Positive"
  ),
  Test_Result = c(
    "PCR positive",
    NA,
    NA,
    "Detected"
  )
)

dependency_check(
  dat,
  condition_variable = "Disease_Status",
  condition_value = "Positive",
  target_variable = "Test_Result"
)
```

`duplicate_check` *Check Duplicate Rows*

Description

Identifies duplicate rows in a dataset and reports their row positions.

Usage

```
duplicate_check(data)
```

Arguments

`data`	A data.frame.

Details

Duplicate rows are identified using `duplicated()`. The first occurrence of a repeated row is not classified as a duplicate. Only subsequent occurrences are marked as duplicates.

Value

An object of class `DuplicateCheck`. The returned data.frame contains the row number and a logical indicator showing whether each row is a duplicate of an earlier row.

Examples

```
duplicate_check(iris)

dat <- data.frame(
  x = c(1, 2, 2, 3),
  y = c("A", "B", "B", "C")
)

duplicate_check(dat)
```

`group_sequence_check` *Check Sequence Integrity Within Groups*

Description

Checks a numeric or integer sequence independently within groups for missing values, duplicates, ordering problems, gaps, and step mismatches.

Usage

```
group_sequence_check(data, group, variable, step = 1)
```

Arguments

`data`	A data.frame.
`group`	A single character string specifying the grouping variable.
`variable`	A single character string specifying the numeric or integer sequence variable.
`step`	A positive finite numeric value specifying the expected increment. Default is 1.

Details

Sequence integrity is evaluated independently within each group and in the original row order.

Missing values in the grouping variable are retained and treated as a separate missing group for diagnostic purposes.

Missing sequence values are reported separately and are excluded when calculating previous values and differences.

All occurrences of duplicated non-missing sequence values within a group are flagged.

A gap occurs when the difference from the previous non-missing value within the same group is greater than `step`.

Value

A data.frame of class `GroupSequenceCheck` containing row-level sequence diagnostics.

Examples

```
dat <- data.frame(
  Animal_ID = c(
    "A001", "A001", "A001",
    "A002", "A002", "A002"
  ),
  Visit = c(
    1, 2, 4,
    1, 2, 2
  )
)

group_sequence_check(
  dat,
  group = "Animal_ID",
  variable = "Visit"
)
```

`id_check` *Check Identifier Integrity*

Description

Checks an identifier variable for missing values, blank values, duplicates, leading or trailing whitespace, and optionally values that do not match a regular expression.

Usage

```
id_check(data, variable, pattern = NULL)
```

Arguments

`data`	A data.frame.
`variable`	A single character string specifying the identifier variable to check.
`pattern`	Optional regular expression that valid identifiers must match.

Details

Character, factor, numeric, and integer identifier variables are supported.

Duplicate detection excludes missing and blank values.

Whitespace checks are performed on the character representation of the identifier.

If `pattern` is supplied, non-missing and non-blank identifiers that do not match the pattern are flagged.

The function does not modify the original data.

Value

A data.frame of class `IDCheck` containing row-level identifier diagnostics.

Examples

```
dat <- data.frame(
  Animal_ID = c(
    "A001",
    "A002",
    "A002",
    " A003",
    "",
    NA
  )
)

id_check(
  dat,
  variable = "Animal_ID",
  pattern = "^A[0-9]{3}$"
)
```

`infinite_check` *Check Infinite Values*

Description

Identifies positive and negative infinite values in numeric variables of a dataset.

Usage

```
infinite_check(data)
```

Arguments

`data`	A data.frame.

Details

Infinite values are detected only in numeric variables. Positive infinity (`Inf`) and negative infinity (`-Inf`) are reported separately.

Missing values (`NA`) are not classified as infinite values.

Non-numeric variables are retained in the output and reported with zero infinite values.

Value

A data.frame of class `InfiniteCheck` containing variable-level information about infinite values.

Examples

```
dat <- data.frame(
  x = c(1, 2, Inf, 4),
  y = c(1, -Inf, 3, 4),
  group = c("A", "B", "C", "D")
)

infinite_check(dat)
```

`length_check` *Check Character Length*

Description

Checks the number of characters in a character or factor variable and identifies values outside user-defined length limits.

Usage

```
length_check(
  data,
  variable,
  min_length = NULL,
  max_length = NULL,
  exact_length = NULL
)
```

Arguments

`data`	A data.frame.
`variable`	A single character string specifying the variable to check.
`min_length`	Optional minimum acceptable number of characters.
`max_length`	Optional maximum acceptable number of characters.
`exact_length`	Optional exact required number of characters.

Details

The selected variable must be character or factor.

Missing values are reported separately and are not classified as length violations.

If `exact_length` is supplied, `min_length` and `max_length` must both be NULL.

Value

A data.frame of class `LengthCheck` containing row-level length diagnostics.

Examples

```
dat <- data.frame(
  Sample_ID = c("A01", "A002", "B12", NA)
)

length_check(
  dat,
  variable = "Sample_ID",
  exact_length = 3
)
```

`missing_check` *Check Missing Values*

Description

Examines each variable in a dataset and reports the number and percentage of missing and complete values.

Usage

```
missing_check(data)
```

Arguments

`data`	A data.frame.

Details

For each variable, the function reports the total number of observations, number of missing values, number of complete values, and their corresponding percentages.

Missing values are identified using `is.na()`.

Value

A data.frame of class `MissingCheck` containing variable-level missing-value information.

Examples

```
missing_check(iris)

dat <- iris
dat$Sepal.Length[1:10] <- NA
dat$Species[1:5] <- NA
missing_check(dat)
```

nzv_check *Check Near-Zero Variance Variables*

Description

Identifies variables with zero or near-zero variance based on the percentage of unique values and the frequency ratio of the two most common values.

Usage

```
nzv_check(data, unique_cut = 10, freq_cut = 19)
```

Arguments

data	A data.frame.
unique_cut	Numeric. Maximum percentage of unique values used to identify near-zero variance variables. Default is 10.
freq_cut	Numeric. Minimum frequency ratio used to identify near-zero variance variables. Default is 19.

Details

Missing values are excluded when calculating the number of unique values, percentage of unique values, and frequency ratio.

A variable is classified as having zero variance when it contains one or fewer unique non-missing values.

A non-constant variable is classified as near-zero variance when both:

- PercentUnique is less than or equal to unique_cut.
- FrequencyRatio is greater than freq_cut.

Variables with zero variance are also flagged in the NearZeroVariance column.

Value

A data.frame of class NZVCheck containing variable-level near-zero variance diagnostics.

Examples

```
dat <- data.frame(
  x = 1:100,
  y = c(rep(0, 99), 1),
  z = rep(5, 100)
)

nzv_check(dat)
```

```
nzv_check(
  dat,
  unique_cut = 20,
  freq_cut = 10
)
```

`outlier_check` *Check Numeric Outliers*

Description

Identifies potential outliers in numeric variables using the interquartile range (IQR) method.

Usage

```
outlier_check(data, k = 1.5)
```

Arguments

`data`	A data.frame.
`k`	A single positive numeric value controlling the IQR multiplier. Default is 1.5.

Details

For each numeric variable, the lower and upper limits are:

$$Q1 - k * IQR$$

and

$$Q3 + k * IQR$$

Values strictly below the lower limit or strictly above the upper limit are classified as potential outliers.

Missing and infinite values are excluded from the calculation of quartiles and outliers. They are reported separately.

Non-numeric variables are retained in the output but are not evaluated for outliers.

Value

A data.frame of class `OutlierCheck` containing variable-level outlier diagnostics.

Examples

```
dat <- data.frame(
  x = c(10, 11, 12, 13, 100),
  group = c("A", "A", "B", "B", "B")
)

outlier_check(dat)

outlier_check(dat, k = 3)
```

`pattern_check` *Check Character Patterns*

Description

Checks whether values in a character or factor variable match a user-specified regular expression.

Usage

```
pattern_check(data, variable, pattern, ignore_case = FALSE)
```

Arguments

`data`	A data.frame.
`variable`	A single character string specifying the variable to check.
`pattern`	A single non-empty character string containing a regular expression.
`ignore_case`	Logical. If TRUE, matching ignores letter case. Default is FALSE.

Details

Missing values are reported separately and are not classified as matching or invalid.

Factor variables are converted to character internally for pattern matching. The original values are retained in the returned data frame.

Pattern matching is performed using `grepl()`.

Value

A data.frame of class `PatternCheck` containing row-level pattern diagnostics.

Examples

```
dat <- data.frame(
  Sample_ID = c(
    "SMP-001",
    "SMP-002",
    "BAD-003",
    NA
  ),
  stringsAsFactors = FALSE
)

pattern_check(
  dat,
  variable = "Sample_ID",
  pattern = "^SMP-[0-9]{3}$"
)
```

---

print.DataAuditReport *Print a DataAudit Report*

Description

Prints a concise summary of a data audit produced by audit_data.

Usage

```
## S3 method for class 'DataAuditReport'
print(x, ...)
```

Arguments

x	An object of class DataAuditReport.
...	Additional arguments, currently unused.

Value

The input object x, invisibly.

print.DataAuditRuleResult *Print a DataAudit Rule Result*

Description

Prints a concise summary of custom audit-rule results produced by `apply_audit_rules`.

Usage

```
## S3 method for class 'DataAuditRuleResult'
print(x, ...)
```

Arguments

`x`	An object of class `DataAuditRuleResult`.
`...`	Additional arguments, currently unused.

Value

The input object `x`, invisibly.

print.DataAuditScore *Print a DataAudit Score*

Description

Prints a concise summary of a data-quality score generated by `audit_score`.

Usage

```
## S3 method for class 'DataAuditScore'
print(x, ...)
```

Arguments

`x`	An object of class `DataAuditScore`.
`...`	Additional arguments, currently unused.

Value

The input object `x`, invisibly.

range_check *Check Values Against a Valid Range*

Description

Identifies values in a numeric variable that fall below or above user-defined valid limits.

Usage

```
range_check(data, variable, min = NULL, max = NULL)
```

Arguments

data	A data.frame.
variable	Character string giving the name of the numeric variable to check.
min	Optional numeric lower limit. Default is NULL.
max	Optional numeric upper limit. Default is NULL.

Details

At least one of min or max must be supplied.

Missing values are not classified as out-of-range. Infinite values are evaluated against the specified limits and will normally be flagged when an appropriate finite limit is supplied.

Value

A data.frame of class RangeCheck containing row-level range diagnostics.

Examples

```
dat <- data.frame(
  Temperature = c(38.5, 39.2, 44.0, 34.0, NA)
)

range_check(
  dat,
  variable = "Temperature",
  min = 35,
  max = 43
)
```

sequence_check *Check Numeric Sequence Integrity*

Description

Checks a numeric or integer variable for missing values, duplicated values, ordering problems, and gaps in an expected sequence.

Usage

```
sequence_check(data, variable, step = 1)
```

Arguments

data	A data.frame.
variable	A single character string specifying the numeric or integer variable to check.
step	A positive numeric value specifying the expected increment between consecutive values. Default is 1.

Details

The sequence is evaluated in the original row order.

A duplicate is a non-missing value appearing more than once in the variable. All occurrences of duplicated values are flagged.

For each non-missing observation after the first available observation, the difference from the previous non-missing value is calculated.

An ordering problem occurs when this difference is less than or equal to zero. Duplicate values are therefore also ordering problems.

A gap occurs when the difference from the previous non-missing value is greater than `step`.

Differences smaller than `step` but greater than zero are classified as step mismatches.

Missing observations are reported separately.

Value

A data.frame of class `SequenceCheck` containing row-level sequence diagnostics.

Examples

```
dat <- data.frame(
  Visit = c(1, 2, 4, 4, 3, NA, 5)
)

sequence_check(
  dat,
  variable = "Visit"
)
```

```
sequence_check(
  data.frame(Time = c(0, 5, 10, 20)),
  variable = "Time",
  step = 5
)
```

---

summary.DataAuditReport

Summarize a DataAudit Report

Description

Produces a compact summary of a data audit generated by audit_data.

Usage

```
## S3 method for class 'DataAuditReport'
summary(object, ...)
```

Arguments

object	An object of class DataAuditReport.
...	Additional arguments, currently unused.

Value

An object of class summary.DataAuditReport containing dataset-level, variable-level, and optional custom rule summaries.

---

summary.DataAuditRuleResult

Summarize a DataAudit Rule Result

Description

Produces a compact summary of custom validation-rule results generated by apply_audit_rules.

Usage

```
## S3 method for class 'DataAuditRuleResult'
summary(object, ...)
```

Arguments

object	An object of class DataAuditRuleResult.
...	Additional arguments, currently unused.

Value

An object of class summary.DataAuditRuleResult.

---

type_check *Check Variable Types*

Description

Checks whether variables in a dataset match user-defined expected data types.

Usage

```
type_check(data, expected)
```

Arguments

data	A data.frame.
expected	A named character vector specifying the expected type for each variable.

Details

Supported expected types are:

- "numeric"
- "integer"
- "character"
- "factor"
- "logical"
- "Date"
- "POSIXct"

The expected argument must be a named character vector. Names correspond to variables in data.

For example:

```
c(Age = "numeric", Species = "factor")
```

Value

A data.frame of class TypeCheck containing variable-level type diagnostics.

Examples

```
dat <- data.frame(
  Age = c(2, 4, 6),
  Species = factor(c("Dog", "Cat", "Dog")),
  Positive = c(TRUE, FALSE, TRUE)
)

type_check(
  dat,
  c(
    Age = "numeric",
    Species = "factor",
    Positive = "logical"
  )
)
```

---

unique_check *Check Unique Identifier*

Description

Evaluates whether a variable can serve as a unique identifier by checking for missing and duplicated values.

Usage

```
unique_check(data, variable)
```

Arguments

data	A data.frame.
variable	A single character string specifying the variable to check.

Details

Each observation is evaluated for missing and duplicated identifier values.

A value is considered duplicated when it occurs more than once among non-missing observations. All occurrences of a repeated value are flagged.

Missing values are flagged separately and are not treated as duplicated values.

The output contains:

- Row: row number.
- Value: identifier value.
- Missing: whether the value is missing.
- Duplicate: whether the non-missing value occurs more than once.
- ValidUnique: whether the value is both non-missing and unique.

Value

A data.frame of class `UniqueCheck` containing row-level uniqueness diagnostics.

Examples

```
dat <- data.frame(
  Animal_ID = c(101, 102, 102, 103, NA)
)

unique_check(
  dat,
  "Animal_ID"
)
```

`whitespace_check` *Check Leading and Trailing Whitespace*

Description

Detects leading and trailing whitespace in a character or factor variable.

Usage

```
whitespace_check(data, variable)
```

Arguments

`data`	A data.frame.
`variable`	A single character string specifying the variable to check.

Details

Missing values are reported separately and are not classified as whitespace problems.

The function detects leading whitespace, trailing whitespace, and values consisting entirely of whitespace. It does not modify the original data.

Value

A data.frame of class `WhitespaceCheck` containing row-level whitespace diagnostics.

Examples

```
dat <- data.frame(
  Species = c("Dog", " Cat", "Cattle ", " Goat ", NA)
)

whitespace_check(
  dat,
  variable = "Species"
)
```

Index