Package 'tidyaudit'

February 27, 2026

Title Pipeline Audit Trails and Data Diagnostics for 'tidyverse' Workflows

Version 0.1.0

Description Provides pipeline audit trails and data diagnostics for 'tidyverse' workflows. The audit trail system captures lightweight metadata snapshots at each step of a pipeline, building a structured audit report without storing the data itself. Also includes diagnostic functions for interactive data analysis.

License LGPL (>= 3)

URL https://github.com/fpcordeiro/tidyaudit

BugReports https://github.com/fpcordeiro/tidyaudit/issues

Depends R (>= 4.1.0)

Imports cli, dplyr (>= 1.2.0), glue, rlang (>= 1.0.0), stats, utils

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

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Fernando Cordeiro [aut, cre, cph]

Maintainer Fernando Cordeiro <fernandolpcordeiro@gmail.com>

Repository CRAN

Date/Publication 2026-02-27 16:00:02 UTC

Contents

`audit_diff` *Compare Two Audit Trail Snapshots*

Description

Computes detailed differences between any two snapshots in an audit trail, including row/column/NA deltas, columns added/removed, type changes, per-column NA changes, and numeric distribution shifts.

Usage

```
audit_diff(.trail, from, to)

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

Arguments

`.trail`	An `audit_trail()` object.
`from`	Label (character) or index (integer) of the first snapshot.
`to`	Label (character) or index (integer) of the second snapshot.
`x`	An `audit_diff` object to print.
`...`	Additional arguments (currently unused).

Value

An `audit_diff` object (S3 list).

See Also

Other audit trail: audit_report(), audit_tap(), print.audit_snap()

Examples

```
trail <- audit_trail("example")
mtcars |>
  audit_tap(trail, "raw") |>
  dplyr::filter(mpg > 20) |>
  audit_tap(trail, "filtered")
audit_diff(trail, "raw", "filtered")
```

---

audit_report *Generate an Audit Report*

Description

Prints a full audit report for a trail, including the trail summary, all diffs between consecutive snapshots, custom diagnostic results, and a final data profile.

Usage

```
audit_report(.trail, format = c("console", "rmd"), file = NULL)
```

Arguments

.trail	An audit_trail() object.
format	Report format. Currently only "console" is supported. "rmd" is planned for a future version.
file	Output file path (used only with format = "rmd").

Value

.trail, invisibly.

See Also

Other audit trail: audit_diff(), audit_tap(), print.audit_snap()

Examples

```
trail <- audit_trail("example")
mtcars |>
  audit_tap(trail, "raw") |>
  dplyr::filter(mpg > 20) |>
  audit_tap(trail, "filtered")
audit_report(trail)
```

`audit_tap` *Record a Pipeline Snapshot*

Description

Transparent pipe pass-through that captures a metadata snapshot and appends it to an audit trail. Returns `.data` unchanged — the function's only purpose is its side effect on `.trail`.

Usage

```
audit_tap(.data, .trail, label = NULL, .fns = NULL)
```

Arguments

`.data`	A data.frame or tibble flowing through the pipe.
`.trail`	An `audit_trail()` object.
`label`	Optional character label for this snapshot. If `NULL`, an auto-generated label like `"step_1"` is used.
`.fns`	Optional named list of diagnostic functions (or formula lambdas) to run on `.data`. Results are stored in the snapshot.

Value

`.data`, unchanged, returned invisibly. The function is a transparent pass-through; its only effect is the side effect on `.trail`.

See Also

Other audit trail: `audit_diff()`, `audit_report()`, `print.audit_snap()`

Examples

```
trail <- audit_trail("example")
result <- mtcars |>
  audit_tap(trail, "raw") |>
  dplyr::filter(mpg > 20) |>
  audit_tap(trail, "filtered")
print(trail)
```

`audit_transform` *Audit a String Transformation*

Description

Applies a transformation function to a character vector and reports what changed. Provides transparency about the transformation by showing counts and before/after examples.

Usage

```
audit_transform(x, clean_fn, name = NULL)

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

Arguments

`x`	Character vector to transform.
`clean_fn`	A function that takes a character vector and returns a transformed character vector of the same length.
`name`	Optional name for the variable (used in output). If `NULL`, captures the variable name from the call.
`...`	Additional arguments (currently unused).

Value

An S3 object of class `audit_transform` containing:

name Name of the variable

clean_fn_name Name of the transformation function used

n_total Total number of elements

n_changed Count of values that changed

n_unchanged Count of values that stayed the same

n_na Count of NA values

pct_changed Percentage of non-NA values that changed

change_examples Data.frame with before/after pairs

cleaned The transformed character vector

See Also

Other data quality: `diagnose_nas()`, `diagnose_strings()`, `get_summary_table()`, `summarize_column()`

Examples

```
x <- c("  hello ", "WORLD", "  foo  ", NA)
result <- audit_transform(x, trimws)
result$cleaned
```

`compare_tables` *Compare Two Tables*

Description

Compares two data.frames or tibbles by examining column names, row counts, key overlap, and numeric discrepancies. Useful for validating data processing pipelines.

Usage

```
compare_tables(x, y, key_cols = NULL)

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

Arguments

`x`	First data.frame or tibble to compare.
`y`	Second data.frame or tibble to compare.
`key_cols`	Character vector of column names to use as keys for matching rows. If `NULL` (default), automatically detects character, factor, and integer columns as keys.
`...`	Additional arguments (currently unused).

Value

An S3 object of class `compare_tbl` containing:

name1, name2 Names of the compared objects

common_columns Column names present in both tables

only_x Column names only in x

only_y Column names only in y

type_mismatches Data.frame of columns with different types, or NULL

nrow_x Number of rows in x

nrow_y Number of rows in y

key_summary Summary of key overlap, or NULL

numeric_summary Data.frame of numeric discrepancies, or NULL

numeric_method How numeric columns were compared

rows_matched Number of rows matched on keys

See Also

Other join validation: validate_join(), validate_primary_keys(), validate_var_relationship()

Examples

```
x <- data.frame(id = 1:3, value = c(10.0, 20.0, 30.0))
y <- data.frame(id = 1:3, value = c(10.1, 20.0, 30.5))
compare_tables(x, y)
```

---

diagnose_nas *Diagnose Missing Values*

Description

Reports NA counts and percentages for each column in a data.frame, sorted by missing percentage in descending order.

Usage

```
diagnose_nas(.data)

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

Arguments

.data	A data.frame or tibble to diagnose.
x	An object to print.
...	Additional arguments (currently unused).

Value

An S3 object of class diagnose_na containing:

table A data.frame with columns variable, n_na, pct_na, and n_valid, sorted by pct_na descending.

n_cols Total number of columns in the input.

n_with_na Number of columns that have at least one NA.

See Also

Other data quality: audit_transform(), diagnose_strings(), get_summary_table(), summarize_column()

Examples

```
df <- data.frame(
  a = c(1, NA, 3),
  b = c(NA, NA, "x"),
  c = c(TRUE, FALSE, TRUE)
)
diagnose_nas(df)
```

`diagnose_strings` *Diagnose String Column Quality*

Description

Audits a character vector for common data quality issues including missing values, empty strings, whitespace problems, non-ASCII characters, and case inconsistencies. Requires the stringi package (in Suggests).

Usage

```
diagnose_strings(x, name = NULL)

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

Arguments

`x`	Character vector to diagnose.
`name`	Optional name for the variable (used in output). If `NULL`, captures the variable name from the call.
`...`	Additional arguments (currently unused).

Value

An S3 object of class `diagnose_strings` containing:

name Name of the variable

n_total Total number of elements

n_na Count of NA values

n_empty Count of empty strings

n_whitespace_only Count of whitespace-only strings

n_leading_ws Count of strings with leading whitespace

n_trailing_ws Count of strings with trailing whitespace

n_non_ascii Count of strings with non-ASCII characters

n_case_variants Number of unique values with case variants

n_case_variant_groups Number of groups of case-insensitive duplicates

case_variant_examples Data.frame with examples of case variants

See Also

Other data quality: `audit_transform()`, `diagnose_nas()`, `get_summary_table()`, `summarize_column()`

Examples

```
firms <- c("Apple", "APPLE", "apple", "  Microsoft ", "Google", NA, "")
diagnose_strings(firms)
```

`filter_drop` *Filter Data with Diagnostic Statistics (Drop)*

Description

Filters a data.frame or tibble by DROPPING rows where the conditions are TRUE, while reporting statistics about dropped rows and optionally the sum of a statistic column that was dropped.

Usage

```
filter_drop(.data, ...)

## S3 method for class 'data.frame'
filter_drop(.data, ..., .stat = NULL, .quiet = FALSE, .warn_threshold = NULL)
```

Arguments

`.data`	A data.frame, tibble, or other object.
`...`	Filter conditions specifying rows to DROP, evaluated in the context of `.data` using tidy evaluation.
`.stat`	An unquoted column or expression to total, e.g., `amount`, `price * qty`. Reports the amount dropped and its share of the total.
`.quiet`	Logical. If `TRUE`, suppress printing diagnostics.
`.warn_threshold`	Numeric between 0 and 1. If set and the proportion of dropped rows exceeds this threshold, a warning is issued.

Value

The filtered data.frame or tibble.

Methods (by class)

- `filter_drop(data.frame)`: Method for data.frame objects

See Also

Other filter diagnostics: `filter_keep()`

Examples

```
df <- data.frame(
  id = 1:5,
  bad = c(FALSE, TRUE, FALSE, TRUE, FALSE),
  sales = 10:14
)
filter_drop(df, bad == TRUE)
filter_drop(df, bad == TRUE, .stat = sales)
```

`filter_keep` *Filter Data with Diagnostic Statistics (Keep)*

Description

Filters a data.frame or tibble while reporting statistics about dropped rows and optionally the sum of a statistic column that was dropped. Keeps rows where the conditions are TRUE (same as `dplyr::filter()`).

Usage

```
filter_keep(.data, ...)

## S3 method for class 'data.frame'
filter_keep(.data, ..., .stat = NULL, .quiet = FALSE, .warn_threshold = NULL)
```

Arguments

`.data`	A data.frame, tibble, or other object.
`...`	Filter conditions, evaluated in the context of `.data` using tidy evaluation (same as `dplyr::filter()`).
`.stat`	An unquoted column or expression to total, e.g., `amount`, `price * qty`. Reports the amount dropped and its share of the total.
`.quiet`	Logical. If `TRUE`, suppress printing diagnostics.
`.warn_threshold`	Numeric between 0 and 1. If set and the proportion of dropped rows exceeds this threshold, a warning is issued.

Value

The filtered data.frame or tibble.

Methods (by class)

- `filter_keep(data.frame)`: Method for data.frame objects

See Also

Other filter diagnostics: `filter_drop()`

Examples

```
df <- data.frame(
  id = 1:6,
  keep = c(TRUE, FALSE, TRUE, NA, TRUE, FALSE),
  sales = c(100, 50, 200, 25, NA, 75)
)
filter_keep(df, keep == TRUE)
filter_keep(df, keep == TRUE, .stat = sales)
```

---

filter_tap *Operation-Aware Filter Taps*

Description

Performs a diagnostic filter AND records filter diagnostics in an audit trail. `filter_tap()` keeps matching rows (like `dplyr::filter()`), `filter_out_tap()` drops matching rows (the inverse).

Usage

```
filter_tap(
  .data,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .quiet = FALSE,
  .warn_threshold = NULL
)

filter_out_tap(
  .data,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .quiet = FALSE,
  .warn_threshold = NULL
)
```

Arguments

.data	A data.frame or tibble.
...	Filter conditions, evaluated in the context of .data using tidy evaluation (same as dplyr::filter()).
.trail	An audit_trail() object, or NULL (the default). When NULL, behavior depends on diagnostic arguments: if none are provided, a plain dplyr::filter() is performed; if .stat, .warn_threshold, or .quiet = TRUE is provided, delegates to filter_keep() or filter_drop().
.label	Optional character label for this snapshot. If NULL, auto-generated as "filter_1" etc.
.stat	An unquoted column or expression to total, e.g., amount, price * qty. Reports the stat amount dropped and its share of the total.
.quiet	Logical. If TRUE, suppress printing diagnostics (default FALSE).
.warn_threshold	Numeric between 0 and 1. If set and the proportion of dropped rows exceeds this threshold, a warning is issued.

Details

When .trail is NULL:

- No diagnostic args: plain dplyr::filter() / dplyr::filter_out()
- Diagnostic args provided: delegates to filter_keep() / filter_drop() (prints diagnostics but no trail recording)
- .label provided: warns that label is ignored

Value

The filtered data.frame or tibble.

See Also

Other operation taps: join_tap

Examples

```
df <- data.frame(id = 1:10, amount = 1:10 * 100, flag = rep(c(TRUE, FALSE), 5))

# With trail
trail <- audit_trail("filter_example")
result <- df |>
  audit_tap(trail, "raw") |>
  filter_tap(amount > 300, .trail = trail, .label = "big_only")
print(trail)

# Inverse: drop matching rows
trail2 <- audit_trail("filter_out_example")
result2 <- df |>
```

```
  audit_tap(trail2, "raw") |>
  filter_out_tap(flag == FALSE, .trail = trail2, .label = "flagged_only")
print(trail2)

# Without trail (plain filter)
result3 <- filter_tap(df, amount > 300)
```

`get_summary_table` *Generate Summary Table for a Data Frame*

Description

Creates a comprehensive summary of all columns in a data.frame, including type, missing values, descriptive statistics, and example values.

Usage

```
get_summary_table(.data, cols = NULL)
```

Arguments

`.data`	A data.frame or tibble to summarize.
`cols`	Optional character vector of column names to summarize. If `NULL` (the default), all columns are summarized.

Value

A data.frame with one row per column containing summary statistics.

See Also

Other data quality: `audit_transform()`, `diagnose_nas()`, `diagnose_strings()`, `summarize_column()`

Examples

```
df <- data.frame(
  id = 1:100,
  value = rnorm(100),
  category = sample(letters[1:5], 100, replace = TRUE)
)
get_summary_table(df)
```

join_tap *Operation-Aware Join Taps*

Description

Performs a dplyr join AND records enriched diagnostics in an audit trail. These functions replace the pattern of wrapping a join with two `audit_tap()` calls, capturing information that plain taps cannot: match rates, relationship type, duplicate keys, and unmatched row counts.

Usage

```
left_join_tap(.data, y, ..., .trail = NULL, .label = NULL, .stat = NULL)

right_join_tap(.data, y, ..., .trail = NULL, .label = NULL, .stat = NULL)

inner_join_tap(.data, y, ..., .trail = NULL, .label = NULL, .stat = NULL)

full_join_tap(.data, y, ..., .trail = NULL, .label = NULL, .stat = NULL)

anti_join_tap(.data, y, ..., .trail = NULL, .label = NULL, .stat = NULL)

semi_join_tap(.data, y, ..., .trail = NULL, .label = NULL, .stat = NULL)
```

Arguments

`.data`	A data.frame or tibble (left table in the join).
`y`	A data.frame or tibble (right table in the join).
`...`	Arguments passed to the corresponding `dplyr::*_join()` function, including `by`, `suffix`, `keep`, `multiple`, `unmatched`, etc. The `by` argument should be passed by name for enriched diagnostics.
`.trail`	An `audit_trail()` object, or `NULL` (the default). When `NULL`, behavior depends on `.stat`: if `.stat` is also `NULL`, a plain dplyr join is performed; if `.stat` is provided, `validate_join()` diagnostics are printed before the join.
`.label`	Optional character label for this snapshot. If `NULL`, auto-generated as `"left_join_1"` etc.
`.stat`	Optional column name (string) for stat tracking, passed to `validate_join()`.

Details

Enriched diagnostics (match rates, relationship type, duplicate keys) require equality joins — `by` as a character vector, named character vector, or simple equality `join_by()` expression (e.g., `join_by(id)`, `join_by(a == b)`). For non-equi `join_by()` expressions, the tap records a basic snapshot without match-rate diagnostics.

All dplyr join features (`join_by`, `multiple`, `unmatched`, `suffix`, etc.) work unchanged via `...`.

When `.trail` is `NULL`:

- `.stat` also `NULL`: plain dplyr join
- `.stat` provided: prints `validate_join()` diagnostics, then joins
- `.label` provided: warns that label is ignored

Value

The joined data.frame or tibble (same as the corresponding `dplyr::*_join()`).

See Also

Other operation taps: `filter_tap()`

Examples

```
orders <- data.frame(id = 1:4, amount = c(100, 200, 300, 400))
customers <- data.frame(id = c(2, 3, 5), name = c("A", "B", "C"))

# With trail
trail <- audit_trail("join_example")
result <- orders |>
  audit_tap(trail, "raw") |>
  left_join_tap(customers, by = "id", .trail = trail, .label = "joined")
print(trail)

# Without trail (plain join)
result2 <- left_join_tap(orders, customers, by = "id")
```

---

print.audit_snap *Create an Audit Trail*

Description

Creates an audit trail object that captures metadata snapshots at each step of a data pipeline. The trail uses environment-based reference semantics so it can be modified in place inside pipes via `audit_tap()`.

Usage

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

audit_trail(name = NULL)

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

Arguments

x	An object to print.
...	Additional arguments (currently unused).
name	Optional name for the trail. If NULL, a timestamped name is generated automatically.

Value

An audit_trail object (S3 class wrapping an environment).

See Also

Other audit trail: audit_diff(), audit_report(), audit_tap()

Examples

```
trail <- audit_trail("my_analysis")
print(trail)
```

---

summarize_column *Summarize a Single Column*

Description

Computes summary statistics for a vector. Handles numeric, character, factor, logical, Date, and other types with appropriate statistics for each.

Usage

```
summarize_column(x)
```

Arguments

x	A vector to summarize.

Value

A named character vector with summary statistics including: type, unique count, missing count, most frequent value (for non-numeric), mean, sd, min, quartiles (q25, q50, q75), max, and three example values.

See Also

Other data quality: audit_transform(), diagnose_nas(), diagnose_strings(), get_summary_table()

Examples

```
summarize_column(c(1, 2, 3, NA, 5))
summarize_column(c("a", "b", "a", "c"))
```

`validate_join` *Validate Join Operations Between Two Tables*

Description

Analyzes a potential join between two data.frames or tibbles without performing the full join. Reports relationship type (one-to-one, one-to-many, etc.), match rates, duplicate keys, and unmatched rows. Optionally tracks a numeric statistic column through the join to quantify impact.

Usage

```
validate_join(x, y, by = NULL, stat = NULL, stat_x = NULL, stat_y = NULL)

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

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

Arguments

`x`	A data.frame or tibble (left table).
`y`	A data.frame or tibble (right table).
`by`	A character vector of column names to join on. Use a named vector `c("key_x" = "key_y")` when column names differ between tables. Unnamed elements are used for both tables.
`stat`	Optional single column name (string) to track in both tables when the column name is the same. Ignored if `stat_x` or `stat_y` is provided.
`stat_x`	Optional column name (string) for a numeric statistic in `x`.
`stat_y`	Optional column name (string) for a numeric statistic in `y`.
`...`	Additional arguments (currently unused).
`object`	A `validate_join` object to summarize.

Value

An S3 object of class `validate_join` containing:

x_name, y_name Names of the input tables from the original call

by_x, by_y Key columns used for the join

counts List with row counts, match rates, and overlap statistics

stat When `stat`, `stat_x`, or `stat_y` is provided, a list with stat diagnostics per table. `NULL` when no stat is provided.

duplicates List with duplicate key information for each table

summary_table A data.frame summarizing the join diagnostics

relation Character string describing the relationship

keys_only_in_x Unmatched keys from x

keys_only_in_y Unmatched keys from y

See Also

Other join validation: `compare_tables()`, `validate_primary_keys()`, `validate_var_relationship()`

Examples

```
x <- data.frame(id = c(1L, 2L, 3L, 3L), value = c("a", "b", "c", "d"))
y <- data.frame(id = c(2L, 3L, 4L), score = c(10, 20, 30))
result <- validate_join(x, y, by = "id")
print(result)

# Track a stat column with different names in each table
x2 <- data.frame(id = 1:3, sales = c(100, 200, 300))
y2 <- data.frame(id = 2:4, cost = c(10, 20, 30))
validate_join(x2, y2, by = "id", stat_x = "sales", stat_y = "cost")
```

`validate_primary_keys` *Validate Primary Keys*

Description

Tests whether a set of columns constitute primary keys of a data.frame, i.e., whether they uniquely identify every row in the table.

Usage

```
validate_primary_keys(.data, keys)

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

Arguments

`.data`	A data.frame or tibble.
`keys`	Character vector of column names to test as primary keys.
`x`	An object to print.
`...`	Additional arguments (currently unused).

Value

An S3 object of class `validate_pk` containing:

table_name Name of the input table from the original call

keys Character vector of column names tested

is_primary_key Logical: TRUE if keys uniquely identify all rows AND no key column contains NA values

n_rows Total number of rows in the table

n_unique_keys Number of distinct key combinations

n_duplicate_keys Number of key combinations that appear more than once

duplicate_keys A data.frame of duplicated key values with their counts

has_numeric_keys Logical: TRUE if any key column is of type double

has_na_keys Logical: TRUE if any key column contains NA values

na_in_keys Named logical vector indicating which key columns contain NAs

See Also

Other join validation: `compare_tables()`, `validate_join()`, `validate_var_relationship()`

Examples

```
df <- data.frame(
  id = c(1L, 2L, 3L, 4L),
  group = c("A", "A", "B", "B"),
  value = c(10, 20, 30, 40)
)
validate_primary_keys(df, "id")
validate_primary_keys(df, "group")
```

---

validate_var_relationship

Validate Variable Relationship

Description

Determines the relationship between two variables in a data.frame: one-to-one, one-to-many, many-to-one, or many-to-many.

Usage

```
validate_var_relationship(.data, var1, var2)

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

Arguments

.data	A data.frame or tibble.
var1	Character string: name of the first variable.
var2	Character string: name of the second variable.
x	An object to print.
...	Additional arguments (currently unused).

Details

Only accepts variables of type character, integer, or factor. Numeric (double) variables are not allowed due to floating-point comparison issues.

Value

An S3 object of class `validate_var_rel` containing:

table_name Name of the input table

var1, var2 Names of the variables analyzed

relation Character string: "one-to-one", "one-to-many", "many-to-one", or "many-to-many"

var1_unique Number of distinct values in var1

var2_unique Number of distinct values in var2

n_combinations Number of unique (var1, var2) pairs

var1_has_dups Does any var1 value map to multiple var2 values?

var2_has_dups Does any var2 value map to multiple var1 values?

See Also

Other join validation: `compare_tables()`, `validate_join()`, `validate_primary_keys()`

Examples

```
df <- data.frame(
  person_id = c(1L, 2L, 3L, 4L),
  department = c("Sales", "Sales", "Engineering", "Engineering"),
  country = c("US", "US", "US", "UK")
)
validate_var_relationship(df, "person_id", "department")
```

Index