Package: tidyaudit (via r-universe)

May 28, 2026

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

Version 0.2.1

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 record without storing the data itself. Operation-aware taps enrich snapshots with join match rates and filter drop statistics. Trails can be serialized to 'JSON' or 'RDS' and exported as self-contained 'HTML' visualizations. Also includes diagnostic functions for interactive data analysis including frequency tables, string quality auditing, and data comparison.

License LGPL (>= 3)

URL https://fpcordeiro.github.io/tidyaudit/, 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, tools, utils

Suggests covr, jsonlite, knitr, pkgdown, rmarkdown, spelling, stringi, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-05-28 01:00:00 UTC

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

RemoteRef HEAD

RemoteSha 550f0aea4bdcef139502bc8bb0575dc231bd116c

Contents

audit_diff . . . 2
audit_export . . . 3
audit_report . . . 4
audit_tap . . . 5
audit_transform . . . 6
compare_tables . . . 8
diagnose_nas . . . 10
diagnose_strings . . . 11
filter_drop . . . 12
filter_keep . . . 13
filter_tap . . . 15
get_summary_table . . . 17
join_tap . . . 18
print.audit_snap . . . 21
read_trail . . . 22
summarize_column . . . 23
tab . . . 23
tab_tap . . . 25
trail_to_df . . . 26
trail_to_list . . . 27
validate_join . . . 28
validate_primary_keys . . . 29
validate_var_relationship . . . 31
write_trail . . . 32

Index **34**

`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(), tab_tap()

Examples

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

---

audit_export *Export an Audit Trail as a Self-Contained HTML File*

Description

Produces a standalone HTML file that visualises the audit trail as an interactive pipeline flow diagram. The file is completely self-contained — no server, internet connection, or R installation is required to view it. Open it in any browser.

Usage

```
audit_export(.trail, file = NULL)
```

Arguments

.trail	An audit_trail() object.
file	Path to the output .html file. If NULL (the default), writes to a temporary file and opens it in the default browser via utils::browseURL().

Details

The trail is serialised via `trail_to_list()` and embedded as JSON inside an HTML template with inline CSS and vanilla JavaScript. The visualisation features:

- Horizontal pipeline flow diagram with colour-coded nodes per operation type (snapshot, join, filter).
- Edges annotated with key deltas (match rate, drop \ added).
- Clickable nodes expanding to show column schema, operation diagnostics, and custom `.fns` results.
- Clickable edges showing the full diff between adjacent snapshots.
- Light / dark theme toggle.
- Collapsible JSON export panel.

Value

The file path (character), invisibly.

See Also

`trail_to_list()`, `write_trail()`

Other audit export: `read_trail()`, `trail_to_df()`, `trail_to_list()`, `write_trail()`

Examples

```
trail <- audit_trail("demo")
mtcars |> audit_tap(trail, "raw")
dplyr::filter(mtcars, mpg > 20) |> audit_tap(trail, "filtered")
audit_export(trail, tempfile(fileext = ".html"))
```

`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 = "console")
```

Arguments

`.trail`	An `audit_trail()` object.
`format`	Report format. Currently only `"console"` is supported.

Value

`.trail`, invisibly.

See Also

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

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,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = 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.

`.numeric_summary`
: Logical. If `FALSE`, skip numeric summary computation in the snapshot (default `TRUE`).

`.cols_include`
: Character vector of column names to include in the snapshot schema, or `NULL` (the default) to include all columns. Mutually exclusive with `.cols_exclude`.

`.cols_exclude`
: Character vector of column names to exclude from the snapshot schema, or `NULL` (the default). Mutually exclusive with `.cols_include`.

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()`, `tab_tap()`

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 Vector Transformation*

Description

Applies a transformation function to a vector and reports what changed. Works with any vector type: character, numeric, Date/POSIXct, factor, or logical. Diagnostics are adapted to the detected input type.

Usage

```
audit_transform(
  x,
  clean_fn,
  name = NULL,
  .tolerance = sqrt(.Machine$double.eps)
)

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

Arguments

x	Vector to transform. Accepted types: character, numeric, Date, POSIXct, factor, or logical.
clean_fn	A function applied to x that returns a vector of the same length, **or** a pre-computed vector of the same length (used directly as the transformation result).
name	Optional name for the variable (used in output). If NULL, captures the variable name from the call.
.tolerance	Numeric tolerance used for the "changed beyond tolerance" diagnostic (numeric type only). Defaults to sqrt(.Machine$double.eps).
...	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, or "<pre-computed>" when a vector was supplied directly

type_class Detected type: "character", "numeric", "Date", "POSIXct", "factor", or "logical"

n_total Total number of elements

n_changed Count of values that changed (including NA status changes)

n_unchanged Count of values that stayed the same

n_na_before Count of NA values before transformation

n_na_after Count of NA values after transformation

pct_changed Percentage of total elements that changed

change_examples Data frame with before/after pairs (up to 10)

diagnostics Type-specific diagnostic list, or NULL for character

cleaned The transformed vector, retaining its type

See Also

diagnose_strings()

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

Examples

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

# Numeric
prices <- c(10.5, 20.0, NA, 30.0)
audit_transform(prices, function(v) round(v))
```

```
# Pre-computed result
audit_transform(prices, round(prices))
```

`compare_tables` *Compare Two Tables*

Description

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

Usage

```
compare_tables(
  x,
  y,
  key_cols = NULL,
  tol = .Machine$double.eps,
  top_n = Inf,
  compare_cols = NULL,
  exclude_cols = NULL,
  on_non_unique = c("warn", "stop")
)

## S3 method for class 'compare_tbl'
print(x, show_n = 5L, ...)

## S3 method for class 'compare_tbl'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

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.
`tol`	Numeric tolerance for comparing numeric columns. Differences less than or equal to `tol` are considered equal. Defaults to `.Machine$double.eps` (machine double-precision).
`top_n`	Maximum number of row-level discrepancies to store **per column** (numeric and categorical), and maximum unmatched keys to store. Defaults to `Inf` (all). Unmatched keys are stored in arbitrary order.
`compare_cols`	Character vector of column names to compare. If `NULL` (default), all common non-key columns are compared. Mutually exclusive with `exclude_cols`.

exclude_cols	Character vector of column names to exclude from comparison. If NULL (default), no columns are excluded. Mutually exclusive with compare_cols.
on_non_unique	What to do when the chosen key_cols do not form a primary key (rows are duplicated on the key, or a key column contains NA) in x or y. "warn" (default) issues a warning and proceeds — note that comparisons will be inflated by cartesian row expansion at the merge. "stop" aborts with the same message.
show_n	Maximum number of rows to display for discrepancies and unmatched keys in the printed output. Defaults to 5L.
...	Additional arguments (currently unused).
row.names	Passed to as.data.frame(). Default NULL.
optional	Passed to as.data.frame(). Default FALSE.

Value

An S3 object of class compare_tbl containing:

name_x, name_y 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 List summarising the chosen keys and their overlap, or NULL if no keys could be determined. Fields: keys, auto (logical), x_unique, y_unique, matches, only_x, only_y, is_pk_x, is_pk_y (logical: do keys uniquely identify rows in each table), n_dup_combos_x, n_dup_combos_y (number of key combinations appearing more than once), has_na_keys_x, has_na_keys_y (NA values present in any key column).

numeric_summary Data.frame of numeric discrepancy quantiles (with n_over_tol count), or NULL

comparison_method How columns were compared ("keys", "row_index", or NA)

rows_matched Number of rows matched on keys

tol The tolerance used

top_n The top_n used

discrepancies Data.frame of row-level numeric discrepancies exceeding tol (or where one side is NA), with key columns (or row_index), column, value_x, value_y, abs_diff, and pct_diff (relative difference as a proportion). NULL if none.

categorical_summary Data.frame with column, n_compared, n_mismatched, pct_mismatched (proportion, 0–1), n_na_mismatch, or NULL

categorical_discrepancies Data.frame of row-level categorical discrepancies with key columns (or row_index), column, value_x, value_y. NULL if none.

total_discrepancies Total number of cell-level discrepancies across all column types (not limited by top_n)

only_x_keys Data.frame of key combinations only in x (up to top_n rows), or NULL

only_y_keys Data.frame of key combinations only in y (up to top_n rows), or NULL

match_summary List with only_x, only_y, matched_no_disc, matched_with_disc, pct_no_disc (proportion, 0–1), pct_with_disc (proportion, 0–1)

Use as.data.frame() to extract all discrepancies (numeric and categorical) as a single tidy data.frame.

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)

# With tolerance — differences <= 0.15 are considered equal
compare_tables(x, y, tol = 0.15)

# Categorical columns are also compared
a <- data.frame(id = 1:3, status = c("ok", "warn", "fail"),
                stringsAsFactors = FALSE)
b <- data.frame(id = 1:3, status = c("ok", "warn", "error"),
                stringsAsFactors = FALSE)
compare_tables(a, b)
```

---

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()`, `tab()`

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()`, `tab()`

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,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = NULL
)

filter_out_tap(
  .data,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .quiet = FALSE,
  .warn_threshold = NULL,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = 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.
.numeric_summary	Logical. If FALSE, skip numeric summary computation in the snapshot (default TRUE).
.cols_include	Character vector of column names to include in the snapshot schema, or NULL (the default) to include all columns. Mutually exclusive with .cols_exclude.
.cols_exclude	Character vector of column names to exclude from the snapshot schema, or NULL (the default). Mutually exclusive with .cols_include.

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()`, `tab()`

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,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = NULL
)

right_join_tap(
  .data,
  y,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = NULL
)

inner_join_tap(
  .data,
  y,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = NULL
)
```

```
full_join_tap(
  .data,
  y,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = NULL
)

anti_join_tap(
  .data,
  y,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = NULL
)

semi_join_tap(
  .data,
  y,
  ...,
  .trail = NULL,
  .label = NULL,
  .stat = NULL,
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = 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	An unquoted column name for stat tracking, e.g., amount. Passed to validate_join().
.numeric_summary	Logical. If FALSE, skip numeric summary computation in the snapshot (default TRUE).
.cols_include	Character vector of column names to include in the snapshot schema, or NULL (the default) to include all columns. Mutually exclusive with .cols_exclude.
.cols_exclude	Character vector of column names to exclude from the snapshot schema, or NULL (the default). Mutually exclusive with .cols_include.

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, show_custom = TRUE, ...)
```

Arguments

x	An object to print.
...	Additional arguments (currently unused).
name	Optional name for the trail. If NULL, a timestamped name is generated automatically.
show_custom	Logical. If TRUE (default), inline annotations (one indented line per custom function) are printed below each snapshot that has custom diagnostics. Set to FALSE to suppress them and display only the main timeline table.

Value

An audit_trail object (S3 class wrapping an environment).

See Also

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

Examples

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

`read_trail` *Read an Audit Trail from a File*

Description

Restores an `audit_trail()` previously saved with `write_trail()`. The file format is detected automatically from the file extension (`.rds` for RDS, `.json` for JSON), or can be specified explicitly via `format`.

Usage

```
read_trail(file, format = NULL)
```

Arguments

`file`	Path to an RDS or JSON file created by `write_trail()`.
`format`	One of `"rds"`, `"json"`, or `NULL` (default). When `NULL`, the format is inferred from the file extension.

Value

A reconstructed `audit_trail()` object with all S3 classes restored.

See Also

`write_trail()`

Other audit export: `audit_export()`, `trail_to_df()`, `trail_to_list()`, `write_trail()`

Examples

```
trail <- audit_trail("example")
mtcars |> audit_tap(trail, "raw")
tmp <- tempfile(fileext = ".rds")
write_trail(trail, tmp)
restored <- read_trail(tmp)
print(restored)
```

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, missing share (proportion from 0 to 1), 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(), tab()

Examples

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

tab *Tabulate Variables*

Description

Produces one-way frequency tables or two-way crosstabulations. One variable gives counts, percentages, and cumulative percentages; two variables give a crosstabulation matrix with row/column totals.

Usage

```
tab(
  .data,
  ...,
  .wt = NULL,
  .sort = c("value_asc", "value_desc", "freq_desc", "freq_asc"),
  .cutoff = NULL,
  .na = c("include", "exclude", "only"),
  .display = c("count", "row_pct", "col_pct", "total_pct")
)

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

## S3 method for class 'tidyaudit_tab'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

.data	A data.frame or tibble.
...	Additional arguments (currently unused).
.wt	Optional unquoted column to use as frequency weights. When supplied, frequencies are weighted sums instead of row counts.
.sort	How to order the rows (and columns in two-way tables). "value_asc" (default) sorts alphabetically (or by factor levels), "value_desc" sorts in reverse, "freq_desc" sorts by frequency descending, "freq_asc" sorts by frequency ascending.
.cutoff	Controls how many values to display. An integer >= 1 keeps the top-N values by frequency. A number in (0, 1) keeps values that cumulatively account for that proportion of the total. Remaining values are grouped under "(Other)". For two-way tables, the cutoff applies to the row variable only.
.na	How to handle NA values. "include" (default) treats NA as a category, "exclude" drops NA rows before tabulation, "only" shows only NA rows.
.display	Cell contents for two-way crosstabulations. One of "count" (default), "row_pct", "col_pct", or "total_pct". Ignored for one-way tables.
x	A tidyaudit_tab object.
row.names	Passed to as.data.frame(). Default NULL.
optional	Passed to as.data.frame(). Default FALSE.

Value

An S3 object of class tidyaudit_tab. Use as.data.frame() to extract the underlying table.

See Also

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

Examples

```
tab(mtcars, cyl)
tab(mtcars, cyl, .sort = "freq_desc")
tab(mtcars, cyl, gear)
tab(mtcars, cyl, gear, .display = "row_pct")
tab(mtcars, cyl, .wt = mpg)
tab(mtcars, cyl, .cutoff = 2)
```

---

tab_tap *Record a Tabulation Snapshot in a Pipeline*

Description

Transparent pipe pass-through that runs `tab()` on the data and stores the result as a custom diagnostic annotation in the audit trail snapshot. Returns `.data` unchanged.

Usage

```
tab_tap(
  .data,
  ...,
  .trail,
  .label = NULL,
  .wt = NULL,
  .sort = c("value_asc", "value_desc", "freq_desc", "freq_asc"),
  .cutoff = NULL,
  .na = c("include", "exclude", "only"),
  .display = c("count", "row_pct", "col_pct", "total_pct"),
  .numeric_summary = TRUE,
  .cols_include = NULL,
  .cols_exclude = NULL
)
```

Arguments

`.data`	A data.frame or tibble.
`...`	Additional arguments (currently unused).
`.trail`	An `audit_trail()` object.
`.label`	Character label for this snapshot.
`.wt`	Optional unquoted column to use as frequency weights. When supplied, frequencies are weighted sums instead of row counts.
`.sort`	How to order the rows (and columns in two-way tables). `"value_asc"` (default) sorts alphabetically (or by factor levels), `"value_desc"` sorts in reverse, `"freq_desc"` sorts by frequency descending, `"freq_asc"` sorts by frequency ascending.

.cutoff	Controls how many values to display. An integer >= 1 keeps the top-N values by frequency. A number in (0, 1) keeps values that cumulatively account for that proportion of the total. Remaining values are grouped under "(Other)". For two-way tables, the cutoff applies to the row variable only.
.na	How to handle NA values. "include" (default) treats NA as a category, "exclude" drops NA rows before tabulation, "only" shows only NA rows.
.display	Cell contents for two-way crosstabulations. One of "count" (default), "row_pct", "col_pct", or "total_pct". Ignored for one-way tables.
.numeric_summary	Logical. If FALSE, skip numeric summary computation in the snapshot (default TRUE).
.cols_include	Character vector of column names to include in the snapshot schema, or NULL (the default) to include all columns. Mutually exclusive with .cols_exclude.
.cols_exclude	Character vector of column names to exclude from the snapshot schema, or NULL (the default). Mutually exclusive with .cols_include.

Value

.data, unchanged, returned invisibly.

See Also

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

Examples

```
trail <- audit_trail("example")
result <- mtcars |>
  tab_tap(cyl, .trail = trail, .label = "by_cyl") |>
  dplyr::filter(mpg > 20) |>
  tab_tap(cyl, .trail = trail, .label = "by_cyl_filtered")
print(trail)
```

---

trail_to_df *Convert an Audit Trail to a Data Frame*

Description

Returns a plain data.frame with one row per snapshot. Nested fields (all_columns, schema, numeric_summary, changes, diagnostics, custom, pipeline, controls) become list-columns. Trail metadata is stored as attributes on the result.

Usage

```
trail_to_df(.trail)
```

Arguments

.trail	An audit_trail() object.

Value

A data.frame with columns index, label, type, timestamp, nrow, ncol, total_nas, all_columns, schema, numeric_summary, changes, diagnostics, custom, pipeline, and controls. Trail name and created_at are stored as attributes "trail_name" and "created_at".

See Also

Other audit export: audit_export(), read_trail(), trail_to_list(), write_trail()

Examples

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

---

trail_to_list *Convert an Audit Trail to a Plain List*

Description

Converts an audit_trail() object to a plain R list suitable for serialisation with jsonlite::toJSON(). All POSIXct timestamps are converted to ISO 8601 character strings and data.frames are converted to lists of named rows for JSON compatibility.

Usage

```
trail_to_list(.trail)
```

Arguments

.trail	An audit_trail() object.

Value

A named list with elements name, created_at (ISO 8601 string), n_snapshots, and snapshots (a named list keyed by snapshot label).

See Also

Other audit export: audit_export(), read_trail(), trail_to_df(), write_trail()

Examples

```
trail <- audit_trail("example")
mtcars |> audit_tap(trail, "raw")
lst <- trail_to_list(trail)
str(lst, max.level = 2)
```

`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")
```

`write_trail` *Write an Audit Trail to a File*

Description

Saves an `audit_trail()` to disk as either an RDS file (default) or a JSON file. The RDS format preserves all R types and can be restored perfectly with `read_trail()`. The JSON format produces a human-readable representation suitable for archiving or interoperability with other tools.

Usage

```
write_trail(.trail, file, format = c("rds", "json"))
```

Arguments

`.trail`	An `audit_trail()` object.
`file`	Path to the output file. A `.rds` extension is conventional for `format = "rds"`; `.json` for `format = "json"`.
`format`	One of `"rds"` (default) or `"json"`. The JSON format requires the **jsonlite** package to be installed.

Value

`.trail`, invisibly.

Note

Custom diagnostic results (the `custom` field, populated via `.fns` in `audit_tap()`) are serialised on a best-effort basis for JSON output. Complex R objects such as environments or functions cannot be represented in JSON and will cause an error.

See Also

`read_trail()`, `trail_to_list()`

Other audit export: `audit_export()`, `read_trail()`, `trail_to_df()`, `trail_to_list()`

Examples

```
trail <- audit_trail("example")
mtcars |> audit_tap(trail, "raw")
tmp <- tempfile(fileext = ".rds")
write_trail(trail, tmp)
restored <- read_trail(tmp)
```

Index

∗ **audit export**
  audit_export, 3
  read_trail, 22
  trail_to_df, 26
  trail_to_list, 27
  write_trail, 32
∗ **audit trail**
  audit_diff, 2
  audit_report, 4
  audit_tap, 5
  print.audit_snap, 21
  tab_tap, 25
∗ **data quality**
  audit_transform, 6
  diagnose_nas, 10
  diagnose_strings, 11
  get_summary_table, 17
  summarize_column, 23
  tab, 23
∗ **filter diagnostics**
  filter_drop, 12
  filter_keep, 13
∗ **join validation**
  compare_tables, 8
  validate_join, 28
  validate_primary_keys, 29
  validate_var_relationship, 31
∗ **operation taps**
  filter_tap, 15
  join_tap, 18
.Machine$double.eps, *8*

anti_join_tap *(*join_tap*)*, 18
as.data.frame(), *9, 10*, *24*
as.data.frame.compare_tbl
        *(*compare_tables*)*, 8
as.data.frame.tidyaudit_tab *(*tab*)*, 23
audit_diff, 2
audit_diff(), *5, 6*, *21*, *26*
audit_export, 3
audit_export(), *22*, *27*, *32*
audit_report, 4
audit_report(), *3*, *6*, *21*, *26*
audit_tap, 5
audit_tap(), *3*, *5*, *18*, *21*, *26*, *32*
audit_trail *(*print.audit_snap*)*, 21
audit_trail(), *3–5*, *15*, *19*, *22*, *25*, *27*, *32*
audit_transform, 6
audit_transform(), *11, 12*, *17*, *23, 24*

compare_tables, 8
compare_tables(), *29–31*

diagnose_nas, 10
diagnose_nas(), *7*, *12*, *17*, *23, 24*
diagnose_strings, 11
diagnose_strings(), *7*, *11*, *17*, *23, 24*
dplyr::filter(), *13–15*

filter_drop, 12
filter_drop(), *14–16*
filter_keep, 13
filter_keep(), *13*, *15, 16*
filter_out_tap *(*filter_tap*)*, 15
filter_tap, 15
filter_tap(), *20*
full_join_tap *(*join_tap*)*, 18

get_summary_table, 17
get_summary_table(), *7*, *11, 12*, *23, 24*

inner_join_tap *(*join_tap*)*, 18

join_tap, *16*, 18
jsonlite::toJSON(), *27*

left_join_tap *(*join_tap*)*, 18

print.audit_diff *(*audit_diff*)*, 2
print.audit_snap, 21
print.audit_snap(), *3*, *5, 6*, *26*

print.audit_trail *(*print.audit_snap*)*, 21
print.audit_transform
*(*audit_transform*)*, 6
print.compare_tbl *(*compare_tables*)*, 8
print.diagnose_na *(*diagnose_nas*)*, 10
print.diagnose_strings
*(*diagnose_strings*)*, 11
print.tidyaudit_tab *(*tab*)*, 23
print.validate_join *(*validate_join*)*, 28
print.validate_pk
*(*validate_primary_keys*)*, 29
print.validate_var_rel
*(*validate_var_relationship*)*, 31

read_trail, 22
read_trail(), *4*, *27*, *32*
right_join_tap *(*join_tap*)*, 18

semi_join_tap *(*join_tap*)*, 18
summarize_column, 23
summarize_column(), *7*, *11, 12*, *17*, *24*
summary.validate_join *(*validate_join*)*,
28

tab, 23
tab(), *7*, *11, 12*, *17*, *23*, *25*
tab_tap, 25
tab_tap(), *3*, *5, 6*, *21*
trail_to_df, 26
trail_to_df(), *4*, *22*, *27*, *32*
trail_to_list, 27
trail_to_list(), *4*, *22*, *27*, *32*

utils::browseURL(), *3*

validate_join, 28
validate_join(), *10*, *19, 20*, *30, 31*
validate_primary_keys, 29
validate_primary_keys(), *10*, *29*, *31*
validate_var_relationship, 31
validate_var_relationship(), *10*, *29, 30*

write_trail, 32
write_trail(), *4*, *22*, *27*