Package ‘keyed’

February 6, 2026

Title Explicit Key Assumptions for Flat-File Data

Version 0.1.3

Language en-US

Description Helps make implicit data assumptions explicit by attaching keys to flat-file data that error when those assumptions are violated. Designed for CSV-first workflows without database infrastructure or version control. Provides key definition, assumption checks, join diagnostics, and optional drift detection against reference snapshots.

License MIT + file LICENSE

URL https://github.com/gcol33/keyed, https://gillescolling.com/keyed/

BugReports https://github.com/gcol33/keyed/issues

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports cli, dplyr (>= 1.0.0), digest, pillar, rlang (>= 1.0.0), tibble, vctrs (>= 0.5.0)

Suggests joinspy, knitr, rmarkdown, testthat (>= 3.0.0), uuid

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Gilles Colling [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3070-6066>)

Maintainer Gilles Colling <gilles.colling051@gmail.com>

Repository CRAN

Date/Publication 2026-02-06 19:50:08 UTC

Contents

add_id *Add identity column*

Description

Adds a stable UUID column to each row. This is an opt-in feature for tracking row lineage through transformations.

Usage

```
add_id(.data, .id = ".id", .overwrite = FALSE)
```

Arguments

`.data`	A data frame.
`.id`	Column name for the ID (default: ".id").
`.overwrite`	If TRUE, overwrite existing ID column. If FALSE (default), error if column exists.

Details

IDs are generated using a hash of row content plus a random salt, making them stable for identical rows within a session but unique across different data frames.

If the uuid package is available, it will be used for true UUIDs. Otherwise, a hash-based ID is generated.

Value

Data frame with ID column added.

Examples

```
df <- data.frame(x = 1:3, y = c("a", "b", "c"))
df <- add_id(df)
df
```

`bind_id` *Bind data frames with ID handling*

Description

Binds data frames while properly handling ID columns. Checks for overlapping IDs, combines the data, and fills in missing IDs.

Usage

```
bind_id(..., .id = ".id")
```

Arguments

`...`	Data frames to bind.
`.id`	Column name for IDs (default: ".id").

Details

This function:

1. Checks if IDs overlap between datasets (warns if so)
2. Binds rows using `dplyr::bind_rows()`
3. Fills missing IDs using `extend_id()`

Use this instead of `dplyr::bind_rows()` when working with ID columns.

Value

Combined data frame with valid IDs for all rows.

Examples

```
df1 <- add_id(data.frame(x = 1:3))
df2 <- data.frame(x = 4:6)
combined <- bind_id(df1, df2)
```

`bind_keyed` *Bind rows of keyed data frames*

Description

Bind keyed data frames

Usage

```
bind_keyed(..., .id = NULL)
```

Arguments

`...`	Data frames to bind.
`.id`	Optional column name to identify source.

Details

Wrapper for `dplyr::bind_rows()` that attempts to preserve key metadata.

Value

A keyed data frame if key is preserved and unique, otherwise tibble.

`check_drift` *Check for drift from committed snapshot*

Description

Compares current data against its committed reference snapshot. Returns diagnostic information about changes.

Usage

```
check_drift(.data, reference = NULL)
```

Arguments

`.data`	A data frame with a snapshot reference.
`reference`	Optional content hash to compare against. If NULL, uses the attached snapshot reference.

Value

A drift report (class `keyed_drift_report`), or NULL if no snapshot found.

Examples

```
df <- key(data.frame(id = 1:3, x = c("a", "b", "c")), id)
df <- commit_keyed(df)

# Modify the data
df$x[1] <- "modified"
check_drift(df)
```

`check_id` *Check ID integrity*

Description

Validates ID column for common issues: missing values, duplicates, and suspicious formats.

Usage

```
check_id(.data, .id = ".id")
```

Arguments

`.data`	A data frame with ID column.
`.id`	Column name (default: ".id").

Value

Invisibly returns a list with:

- `valid`: TRUE if no issues found
- `n_na`: count of NA values
- `n_duplicates`: count of duplicate IDs
- `format_ok`: TRUE if IDs look like proper UUIDs/hashes

Examples

```
df <- add_id(data.frame(x = 1:3))
check_id(df)
```

`check_id_disjoint` *Check IDs are disjoint across datasets*

Description

Verifies that ID columns don't overlap between datasets. Useful before binding datasets to ensure no ID collisions.

Usage

```
check_id_disjoint(..., .id = ".id")
```

Arguments

`...`	Data frames to check.
`.id`	Column name for IDs (default: ".id").

Value

Invisibly returns a list with:

- `disjoint`: TRUE if no overlaps found
- `overlaps`: character vector of overlapping IDs (if any)

Examples

```
df1 <- add_id(data.frame(x = 1:3))
df2 <- add_id(data.frame(x = 4:6))
check_id_disjoint(df1, df2)
```

`clear_all_snapshots` *Clear all snapshots from cache*

Description

Clear all snapshots from cache

Usage

```
clear_all_snapshots(confirm = TRUE)
```

Arguments

`confirm`	If TRUE, require confirmation.

Value

No return value, called for side effects.

`clear_snapshot` *Clear snapshot for a data frame*

Description

Removes the snapshot reference from a data frame.

Usage

```
clear_snapshot(.data, purge = FALSE)
```

Arguments

`.data`	A data frame.
`purge`	If TRUE, also remove the snapshot from cache.

Value

Data frame without snapshot reference.

commit_keyed *Commit a keyed data frame as reference*

Description

Stores a hash-based snapshot of the current data state. Only one active reference per data frame (identified by its content hash).

Usage

```
commit_keyed(.data, name = NULL)
```

Arguments

`.data`	A data frame (preferably keyed).
`name`	Optional name for the snapshot. If NULL, derived from data.

Details

The snapshot stores:

- Row count
- Column names and types
- Hash of key columns (if keyed)
- Hash of full content

Snapshots are stored in memory for the session. They are keyed by content hash, so identical data shares the same snapshot.

Value

Invisibly returns `.data` with snapshot metadata attached.

Examples

```
df <- key(data.frame(id = 1:3, x = c("a", "b", "c")), id)
df <- commit_keyed(df)

# Later, check for drift
df2 <- df
df2$x[1] <- "z"
check_drift(df2)
```

compare_ids — *Compare IDs between data frames*

Description

Compares IDs between two data frames to detect lost rows.

Usage

```
compare_ids(before, after, .id = ".id")
```

Arguments

`before`	Data frame before transformation.
`after`	Data frame after transformation.
`.id`	Column name for IDs (default: ".id").

Value

A list with:

- `lost`: IDs present in `before` but not `after`
- `gained`: IDs present in `after` but not `before`
- `preserved`: IDs present in both

Examples

```
df1 <- add_id(data.frame(x = 1:5))
df2 <- df1[1:3, ]
compare_ids(df1, df2)
```

compare_keys — *Compare key values between two data frames*

Description

Identifies keys that are new, removed, or common between two keyed data frames. Does not compare values, only key membership.

Usage

```
compare_keys(x, y, by = NULL)
```

Arguments

`x`	First keyed data frame.
`y`	Second keyed data frame.
`by`	Column(s) to compare. If NULL, uses the key from x.

Value

A key comparison object.

Examples

```
df1 <- key(data.frame(id = 1:3, x = 1:3), id)
df2 <- key(data.frame(id = 2:4, x = 2:4), id)
compare_keys(df1, df2)
```

`compare_structure` *Compare structure of two data frames*

Description

Compares the structural properties of two data frames without comparing actual values. Useful for detecting schema drift.

Usage

```
compare_structure(x, y)
```

Arguments

`x`	First data frame.
`y`	Second data frame.

Value

A structure comparison object.

Examples

```
df1 <- data.frame(id = 1:3, x = c("a", "b", "c"))
df2 <- data.frame(id = 1:5, x = c("a", "b", "c", "d", "e"), y = 1:5)
compare_structure(df1, df2)
```

`diagnose_join` *Diagnose a join before executing*

Description

Analyzes join cardinality without performing the full join. Useful for detecting many-to-many joins that would explode row count.

Usage

```
diagnose_join(x, y, by = NULL, use_joinspy = TRUE)
```

Arguments

`x`	Left data frame.
`y`	Right data frame.
`by`	Join specification.
`use_joinspy`	If TRUE (default), use joinspy for enhanced diagnostics when available. Set to FALSE to use built-in diagnostics only.

Details

If the joinspy package is installed, this function delegates to `joinspy::join_spy()` for enhanced diagnostics including whitespace detection, encoding issues, and detailed match analysis.

Value

A list with cardinality information, or a JoinReport object if joinspy is used.

See Also

`joinspy::join_spy()` for enhanced diagnostics (if installed)

Examples

```
x <- data.frame(id = c(1, 1, 2), a = 1:3)
y <- data.frame(id = c(1, 1, 2), b = 4:6)
diagnose_join(x, y, by = "id", use_joinspy = FALSE)
```

`extend_id` *Extend IDs to new rows*

Description

Adds IDs to rows where the ID column is NA, preserving existing IDs. Useful after binding new data to an existing dataset with IDs.

Usage

```
extend_id(.data, .id = ".id")
```

Arguments

`.data`	A data frame with an ID column (possibly with NAs).
`.id`	Column name for IDs (default: ".id").

Value

Data frame with IDs filled in for NA rows.

Examples

```
# Original data with IDs
old <- add_id(data.frame(x = 1:3))

# New data without IDs
new <- data.frame(.id = NA_character_, x = 4:5)

# Combine and extend
combined <- dplyr::bind_rows(old, new)
extend_id(combined)
```

`find_duplicates` *Find duplicate keys*

Description

Identifies rows with duplicate key values.

Usage

```
find_duplicates(.data, ...)
```

Arguments

`.data`	A data frame.
`...`	Column names to check. If empty, uses the key columns.

Value

Data frame containing only the rows with duplicate keys, with a `.n` column showing the count.

Examples

```
df <- data.frame(id = c(1, 1, 2, 3, 3, 3), x = letters[1:6])
find_duplicates(df, id)
```

`get_id` *Get ID column*

Description

Get ID column

Usage

```
get_id(.data, .id = ".id")
```

Arguments

`.data`	A data frame.
`.id`	Column name (default: ".id").

Value

Character vector of IDs, or NULL if not present.

`get_key_cols` *Get key column names*

Description

Get key column names

Usage

```
get_key_cols(.data)
```

Arguments

`.data`	A keyed data frame.

Value

Character vector of column names, or NULL if no key.

`has_id` *Check if data frame has IDs*

Description

Check if data frame has IDs

Usage

```
has_id(.data, .id = ".id")
```

Arguments

`.data`	A data frame.
`.id`	Column name to check (default: ".id").

Value

Logical.

`has_key` *Check if data frame has a key*

Description

Check if data frame has a key

Usage

```
has_key(.data)
```

Arguments

`.data`	A data frame.

Value

Logical.

`key` *Define a key for a data frame*

Description

Attaches key metadata to a data frame, marking which column(s) form the unique identifier for rows. Keys are validated for uniqueness at creation.

Usage

```
key(.data, ..., .validate = TRUE, .strict = FALSE)

key(.data) <- value
```

Arguments

`.data`	A data frame or tibble.
`...`	Column names (unquoted) that form the key. Can be a single column or multiple columns for a composite key.
`.validate`	If `TRUE` (default), check that the key is unique.
`.strict`	If `TRUE`, error on non-unique keys. If `FALSE` (default), warn but still attach the key.
`value`	Character vector of column names to use as key.

Value

A keyed data frame (class `keyed_df`).

Examples

```
df <- data.frame(id = 1:3, x = c("a", "b", "c"))
key(df, id)

# Composite key
df2 <- data.frame(country = c("US", "US", "UK"), year = c(2020, 2021, 2020), val = 1:3)
key(df2, country, year)
```

`key_is_valid` *Check if the key is still valid*

Description

Checks whether the key columns still exist and are still unique.

Usage

```
key_is_valid(.data)
```

Arguments

`.data`	A keyed data frame.

Value

Logical. Returns FALSE with a warning if key is invalid.

`key_status` *Get key status summary*

Description

Returns diagnostic information about a keyed data frame.

Usage

```
key_status(.data)
```

Arguments

`.data`	A data frame.

Value

A key status object with diagnostic information.

Examples

```
df <- key(data.frame(id = 1:3, x = c("a", "b", "c")), id)
key_status(df)
```

`list_snapshots` *List all snapshots in cache*

Description

List all snapshots in cache

Usage

```
list_snapshots()
```

Value

Data frame with snapshot information.

`lock_complete` *Assert that data is complete (no missing values anywhere)*

Description

Checks that all columns have no NA values.

Usage

```
lock_complete(.data, .strict = FALSE)
```

Arguments

`.data`	A data frame.
`.strict`	If `TRUE`, error on failure. If `FALSE` (default), warn.

Value

Invisibly returns `.data` (for piping).

`lock_coverage` *Assert minimum coverage of values*

Description

Checks that the fraction of non-NA values meets a threshold. Useful after joins to verify expected coverage.

Usage

```
lock_coverage(.data, threshold, ..., .strict = FALSE)
```

Arguments

`.data`	A data frame.
`threshold`	Minimum fraction of non-NA values (0 to 1).
`...`	Column names (unquoted) to check. If empty, checks all columns.
`.strict`	If `TRUE`, error on failure. If `FALSE` (default), warn.

Value

Invisibly returns `.data` (for piping).

Examples

```
df <- data.frame(id = 1:10, x = c(1:8, NA, NA))
lock_coverage(df, 0.8, x)
lock_coverage(df, 0.9, x)  # warns (only 80% coverage)
```

`lock_no_na` *Assert that columns have no missing values*

Description

Checks that specified columns contain no NA values.

Usage

```
lock_no_na(.data, ..., .strict = FALSE)
```

Arguments

`.data`	A data frame.
`...`	Column names (unquoted) to check. If empty, checks all columns.
`.strict`	If `TRUE`, error on failure. If `FALSE` (default), warn.

Value

Invisibly returns `.data` (for piping).

Examples

```
df <- data.frame(id = 1:3, x = c("a", NA, "c"))
lock_no_na(df, id)
lock_no_na(df, x)  # warns
```

`lock_nrow` *Assert row count within expected range*

Description

Checks that the number of rows is within an expected range. Useful for sanity checks after filtering or joins.

Usage

```
lock_nrow(.data, min = 1, max = Inf, expected = NULL, .strict = FALSE)
```

Arguments

`.data`	A data frame.
`min`	Minimum expected rows (inclusive). Default 1.
`max`	Maximum expected rows (inclusive). Default Inf.
`expected`	Exact expected row count. If provided, overrides min/max.
`.strict`	If `TRUE`, error on failure. If `FALSE` (default), warn.

Value

Invisibly returns `.data` (for piping).

Examples

```
df <- data.frame(id = 1:100)
lock_nrow(df, min = 50, max = 200)
lock_nrow(df, expected = 100)
```

`lock_unique` *Assert that columns are unique*

Description

Checks that the combination of specified columns has unique values. This is a point-in-time assertion that either passes silently or fails.

Usage

```
lock_unique(.data, ..., .strict = FALSE)
```

Arguments

`.data`	A data frame.
`...`	Column names (unquoted) to check for uniqueness.
`.strict`	If `TRUE`, error on failure. If `FALSE` (default), warn.

Value

Invisibly returns `.data` (for piping).

Examples

```
df <- data.frame(id = 1:3, x = c("a", "b", "c"))
lock_unique(df, id)

# Fails with warning
df2 <- data.frame(id = c(1, 1, 2), x = c("a", "b", "c"))
lock_unique(df2, id)
```

`make_id` *Create ID from columns*

Description

Creates an ID column by combining values from one or more columns. Unlike `add_id()`, this produces deterministic IDs based on column values.

Usage

```
make_id(.data, ..., .id = ".id", .sep = "|")
```

Arguments

`.data`	A data frame.
`...`	Columns to combine into the ID.
`.id`	Column name for the ID (default: ".id").
`.sep`	Separator between column values (default: "|").

Value

Data frame with ID column added.

Examples

```
df <- data.frame(country = c("US", "UK", "US"), year = c(2020, 2020, 2021))
make_id(df, country, year)
#>   .id      country year
#> 1 US|2020  US      2020
#> 2 UK|2020  UK      2020
#> 3 US|2021  US      2021
```

`remove_id` *Remove ID column*

Description

Remove ID column

Usage

```
remove_id(.data, .id = ".id")
```

Arguments

`.data`	A data frame.
`.id`	Column name to remove (default: ".id").

Value

Data frame without the ID column.

summary.keyed_df *Summary method for keyed data frames*

Description

Summary method for keyed data frames

Usage

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

Arguments

`object`	A keyed data frame.
`...`	Additional arguments (ignored).

Value

Invisibly returns a summary list.

unkey *Remove key from a data frame*

Description

Remove key from a data frame

Usage

```
unkey(.data)
```

Arguments

`.data`	A keyed data frame.

Value

A tibble without key metadata.

Index