Package: ksCompare (via r-universe)

July 6, 2026

Title Smart Data Frame Comparison in the Spirit of SAS PROC COMPARE

Version 0.2.3

Description A tidyverse-native engine for comparing two data frames in the spirit of SAS 'PROC COMPARE', with extensions for clinical/pharma workflows: per-column tolerances (absolute, relative, and ULP), smart column matching, intelligent type reconciliation, SAS special-missing awareness, diff-pattern detection, and rich HTML/Excel reports.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Depends R (>= 4.1)

Imports base64enc, cli, digest, dplyr, rlang, stringi, tibble, vctrs

Suggests arrow, arsenal, broom, clue, haven, htmltools, knitr, openxlsx2, pointblank, reactable, rmarkdown, stringdist, testthat (>= 3.0.0), withr

Config/testthat/edition 3

URL https://al-garik.github.io/ksCompare/, https://github.com/al-garik/ksCompare, https://crow16384.r-universe.dev/ksCompare

BugReports https://github.com/al-garik/ksCompare/issues

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev

Repository https://crow16384.r-universe.dev

Date/Publication 2026-07-06 13:43:16 UTC

RemoteUrl https://github.com/al-garik/ksCompare

RemoteRef HEAD

RemoteSha 554841cfa9485bd78809a67cac190b15b6504d09

Contents

`as_ks_comparison` *Convert an* `arsenal::comparedf` *result into a* `ks_comparison`

Description

Lossy interop: takes the two source frames stored on a `arsenal::comparedf` object and reruns `ks_compare()` using the same BY keys. Useful for teams migrating from `arsenal::comparedf()` to ksCompare without rewriting their inputs.

Usage

```
as_ks_comparison(x, ...)
```

Arguments

`x`	An `arsenal::comparedf` object.
`...`	Additional arguments forwarded to `ks_compare()` (e.g. `tolerance`, `options`).

Details

Note that ksCompare's diff is computed from scratch — it does not attempt to translate `comparedf`'s internal diff tables row-by-row.

Value

A `ks_comparison` object.

`ks_assert_clean` *Use a comparison as a pipeline gate*

Description

Treats a `ks_comparison` as a pass/fail check. By default a "passing" comparison has zero schema, key, and value differences; callers can relax those expectations to allow a budget of known acceptable diffs.

Usage

```
ks_assert_clean(
  x,
  max_value_diffs = 0L,
  max_schema_diffs = 0L,
  max_unmatched_rows = 0L
)

ks_pointblank_step(
  agent,
  comparison,
  max_value_diffs = 0L,
  max_schema_diffs = 0L,
  max_unmatched_rows = 0L,
  label = "ksCompare clean"
)
```

Arguments

`x`	A `ks_comparison` object returned by `ks_compare()`.
`max_value_diffs`	Integer. Maximum number of differing cells allowed in matched rows. Default `0L` (strict).
`max_schema_diffs`	Integer. Maximum schema differences allowed, counted as the sum of base-only columns, comp-only columns, and matched columns whose `kind` differs. Default `0L` (strict).
`max_unmatched_rows`	Integer. Maximum number of unmatched rows allowed (`n_base_only_rows + n_comp_only_rows`). Default `0L` (strict).
`agent`	A `ptblank_agent` (typically piped in from `pointblank::create_agent()`) or a data frame / tibble. The comparison gate is added as a step on this object and `agent` is returned, so it composes with the rest of a pointblank pipeline.

comparison	A ks_comparison produced by ks_compare() that the step will gate on.
label	Optional label for the pointblank step.

Details

Two flavours are provided:

- ks_assert_clean() throws a classed error (ksCompare_assertion_failed) and halts a pipeline when expectations are unmet. Returns the comparison invisibly on success, so it can be dropped into a |> chain.
- ks_pointblank_step() returns a pointblank step that wraps ks_assert_clean() for use inside pointblank::create_agent() / pointblank::action_levels() flows.

Value

ks_assert_clean() returns x invisibly when expectations are met. On failure, raises a condition of class ksCompare_assertion_failed (catchable in CI with tryCatch(..., ksCompare_assertion_failed = function(e) ...)). ks_pointblank_step() returns a pointblank step.

Examples

```
a <- data.frame(id = 1:3, x = 1:3)
ks_compare(a, a, by = "id") |> ks_assert_clean()

# Allow a small budget of known diffs
## Not run:
  ks_compare(a, b, by = "id") |>
    ks_assert_clean(max_value_diffs = 5)

## End(Not run)
```

---

ks_cause_summary *Diff causes summary*

Description

Aggregates the note column of cmp$value_diff into a tidy taxonomy: one row per distinct cause, with counts and the columns it affects. Helps answer "what kinds of discrepancies do we have?" before drilling into individual cells.

Usage

```
ks_cause_summary(x)
```

Arguments

x	A ks_comparison.

Details

Compound notes (multiple cues joined by `"; "`) are split into their elementary cues, so a single cell can contribute to several causes. Cells with `note = NA` (a "plain" value change with no recognised cue) are bucketed as `"plain value change"`.

Value

A tibble with columns:

- `cause`: short label (e.g. `"letter case differs"`).
- `n_cells`: number of cells contributing this cue.
- `n_columns`: number of distinct base columns affected.
- `columns`: comma-separated sample of the affected columns (capped at 5 names). Sorted by `n_cells` descending. Zero-row tibble when there are no value differences.

See Also

`ks_compare()`, `ks_row_diff_summary()`.

Examples

```
a <- data.frame(id = 1:3, x = c("a", "b ", "c"), y = c(1, 2, 3))
b <- data.frame(id = 1:3, x = c("A", "b",  "c"), y = c(1, 2, 4))
cmp <- ks_compare(a, b, by = "id")
ks_cause_summary(cmp)
```

---

ks_comp_options *Comparison options*

Description

Builds an options object consumed by `ks_compare()` that controls missing-value semantics, label / format comparison, string normalisation, and time-zone handling.

Usage

```
ks_comp_options(
  na_equal = getOption("ksCompare.na_equal", TRUE),
  sas_special_missing = getOption("ksCompare.sas_special_missing", TRUE),
  compare_labels = getOption("ksCompare.compare_labels", TRUE),
  compare_formats = getOption("ksCompare.compare_formats", TRUE),
  str_trim = getOption("ksCompare.str_trim", FALSE),
  str_case = getOption("ksCompare.str_case", "sensitive"),
  str_norm = getOption("ksCompare.str_norm", "none"),
  tz = getOption("ksCompare.tz", "preserve"),
  path = getOption("ksCompare.path", NULL)
)
```

Arguments

na_equal	Treat two NAs as equal? Default TRUE. With FALSE, any cell where one side is NA and the other is not is reported as a value diff; cells where *both* sides are NA are also reported (mirroring SAS PROC COMPARE behaviour with the NOMISS option turned off).
sas_special_missing	Distinguish SAS special missings (.A-.Z and ._) when comparing numeric columns imported via haven? Default TRUE. When TRUE, .A and .B compare as different even though both are NA in R; relies on the tagged_na tag attached by haven::read_sas().
compare_labels	Compare attr(x, "label") between matched columns and surface differences in cmp$schema_diff? Default TRUE. Set to FALSE to suppress label-only diffs (e.g. when metadata drift is expected).
compare_formats	Compare attr(x, "format.sas") between matched columns? Default TRUE. Comparison is trailing-dot- and case-tolerant, so DATE9., DATE9, and date9. all compare as equal.
str_trim	Trim leading/trailing whitespace with stringi::stri_trim_both() before comparing strings? Default FALSE. Useful when one source has been padded by a fixed-width exporter.
str_case	One of "sensitive" (default) or "fold". When "fold", strings are compared after Unicode case folding (stringi::stri_trans_tolower()).
str_norm	One of "none" (default) or "NFC". When "NFC", strings are Unicode-normalised before comparing so that visually-identical sequences with different code-point compositions match.
tz	One of "preserve" (default), "UTC", or "strip". Controls how POSIXct columns are reconciled when the two sides carry different tzone attributes: • "preserve": values are compared as-is (a tz mismatch surfaces as a diff if it changes the underlying instant). • "UTC": both sides are converted to UTC before compare. • "strip": tzone is dropped and values compared as POSIXct.
path	Optional output folder used by downstream writers (ks_report_html(), ks_report_xlsx()) when their own path = argument is left blank. Default NULL means "use the current working directory". The folder is created on first use if it does not already exist. Setting it once on ks_comp_options() and passing the result through ks_compare() keeps every artefact from a single comparison run in the same place.

Value

A ks_comp_options S3 list.

Global defaults via options()

Each argument falls back to a global R option in the ksCompare.* namespace, then to the package default. Precedence is: explicit argument > getOption("ksCompare.<arg>")

> built-in default. That makes it possible to set project-wide defaults once and then call `ks_compare()` without repeating yourself:

```
# In .Rprofile or at the top of a script
options(ksCompare.path     = "out/qc",
        ksCompare.str_case = "fold")

ks_compare(base, comp) |> ks_report_html()   # honours the globals
```

`ks_set_comp_options()` is a small wrapper around `base::options()` that takes the same argument names without the `ksCompare.` prefix.

Recognised options: `ksCompare.na_equal`, `ksCompare.sas_special_missing`, `ksCompare.compare_labels`, `ksCompare.compare_formats`, `ksCompare.str_trim`, `ksCompare.str_case`, `ksCompare.str_norm`, `ksCompare.tz`, `ksCompare.path`.

See Also

`ks_set_comp_options()` for setting the `ksCompare.*` globals from R code.

Examples

```
# Defaults: strict NAs, label & format comparison on
ks_comp_options()

# Loose strings: trim padding and ignore case
ks_comp_options(str_trim = TRUE, str_case = "fold")

# Suppress label/format drift, but keep cell-level strictness
ks_comp_options(compare_labels = FALSE, compare_formats = FALSE)

# Treat any NA as a difference (PROC COMPARE without NOMISS)
ks_comp_options(na_equal = FALSE)

# Pin every report from this run to a dedicated folder
ks_comp_options(path = tempfile("ksCompare_"))

# Pick up a project-wide global path
withr::with_options(
  list(ksCompare.path = tempfile("ksCompare_")),
  ks_comp_options()$path
)
```

`ks_compare` *Compare two data frames*

Description

`ks_compare()` is the single entry point for the package. It compares `base` against `comp` and returns a `ks_comparison` S3 object containing schema, key, row, and value differences plus a small QC manifest.

Usage

```
ks_compare(
  base,
  comp,
  by = NULL,
  mapping = NULL,
  tolerance = ks_tol(),
  coerce = c("safe", "strict", "lossy"),
  dup_keys = c("first", "last", "keep_all", "all_pairs", "error"),
  options = ks_comp_options(),
  base_name = NULL,
  comp_name = NULL,
  find_patterns = FALSE,
  n_first_last = 5L,
  max_unmatched_rows = 100L,
  loglevel = c("verdict", "verbose")
)
```

Arguments

`base`, `comp`
: Either an in-memory data frame (or anything coercible via `tibble::as_tibble()`: tibbles, data.tables, Arrow tables) or a single character file path. When given a path, the file is read automatically based on its extension:

  - `.sas7bdat`, `.xpt` (via `haven`)
  - `.parquet`, `.feather` / `.arrow` (via `arrow`)
  - `.rds`, `.rdata` / `.rda`
  - `.csv`, `.tsv` / `.txt`

  `base` is treated as the reference; `comp` is the candidate. When both inputs are file paths that share the same basename (e.g. `prod/adsl.sas7bdat` vs `qc/adsl.sas7bdat`), display names are automatically disambiguated with the parent directory.

`by`
: Optional column name(s) to use as the matching key.

  - `NULL` (default): rows are matched by row position. A warning banner is shown in the HTML report so this is never silent.
  - `character`: same column name on both sides. Use a named vector when the key is renamed: `c(USUBJID = "SUBJID")`.
  - `"auto"`: search the smallest combination of shared columns that is unique on **both** sides (capped at 4 columns). Falls back to position match with a warning if nothing qualifies.

`mapping`
: Optional explicit column mapping for renamed columns. Either a fully named character vector (`c(base_col = "comp_col", ...)`) or a 2-column data frame with columns `base` and `comp`. Mapping pairs always win over exact-name matching.

`tolerance`
: A `ks_tol()` specification. Defaults to strict equality. Per-column overrides supported via `ks_tol(per_column = list(col = ks_tol(...)))`.

`coerce`
: Type-coercion strictness. One of:

- `"safe"` (default): `vctrs::vec_ptype2()` plus integer double, factor character, `haven_labelled` unwrapping.
- `"strict"`: only common types `vctrs::vec_ptype2()` agrees on. Type mismatches surface as `kind = "type_mismatch"` rows.
- `"lossy"`: additionally allows numeric character, date/datetime character (ISO-8601), factor integer (level codes).

`dup_keys` Strategy for duplicate keys (per-side independently). One of:

- `"first"` (default): keep the first occurrence per side. Drops rows; emits a `ksCompare_dup_keys_resolved` info message.
- `"last"`: keep the last occurrence per side. Same message.
- `"keep_all"`: pair duplicate rows positionally within each key group (1 1, 2 2, …). Leftover rows on the longer side become base- or compare-only.
- `"all_pairs"`: cartesian-pair every base row with every comp row sharing a key value. Emits a `ksCompare_all_pairs_cardinality` warning when group sizes disagree.
- `"error"`: raise on any duplicate key.

`options` A `ks_comp_options()` specification controlling NA / SAS special-missing semantics, label / format comparison, string normalisation, and the default output folder for reports.

`base_name`, `comp_name` Optional display names for the two frames (used in `print()` and reports). Default to the names of the supplied expressions.

`find_patterns` Logical (default `FALSE`). When `TRUE`, run the pattern-detection pass (constant offsets, sign flips, unit rescales, epoch shifts, etc.) and populate `cmp$pattern_summary`. The detectors iterate over every column with a diff and can be noticeably slow on large value-diff tables, so they are off by default. When `FALSE`, `cmp$pattern_summary` is an empty tibble with the usual column layout.

`n_first_last` Non-negative integer (default `5`). Per matched column with at least one cell diff, capture the first and last `n_first_last` differences in `key_id` order on `cmp$first_last_unequal`. Mirrors the *First / Last N Obs With Some Compared Variables Unequal* tables of SAS `PROC COMPARE`. Set to `0` to skip.

`max_unmatched_rows` Non-negative integer (default `100`). Cap on how many full base-only / comp-only rows are stored on `cmp$unmatched_rows` for inspection (and surfaced by `ks_unmatched_rows()` / the HTML / XLSX reports). The cap is apportioned proportionally between the two sides; truncation is recorded on the result's `truncated` and `n_total` attributes. Set to `0` to skip storing the row data entirely (the row counts on `cmp$key_diff` and `summary()` are unaffected).

`loglevel` Controls what `print()` emits after the comparison. One of:

- `"verdict"` (default): prints a single coloured verdict line (`cli_alert_success` / `_warning` / `_danger`) plus any `"critical"` recommendations. The

full schema / rows / values breakdown is omitted. Pipeline info messages (duplicate-key resolution, auto-key selection, etc.) are unaffected.

- `"verbose"`: `print()` emits the complete schema / rows / values breakdown. Pipeline info messages fire normally as well.

Value

A `ks_comparison` object with components:

- `meta`: counts, keys, matching strategy, row-key lookup table.
- `schema_diff`: one row per matched / unmatched column with kind / label / format comparison.
- `key_diff`: counts of matched / base-only / comp-only rows.
- `row_diff`: long table of `key_id`, `base_row`, `comp_row`, `status`.
- `value_diff`: long table of differing cells with `column_base`, `column_comp`, `kind`, `base`, `comp`, `diff`, `note`.
- `pattern_summary`: detected recurring shapes per column (only populated when `find_patterns = TRUE`; otherwise an empty tibble with the standard column layout).
- `unmatched_rows`: full base-only / comp-only rows (capped by `max_unmatched_rows`); see `ks_unmatched_rows()`.
- `first_last_unequal`: per-column first / last `n_first_last` differing observations in `key_id` order (PROC COMPARE-style).
- `verdict`: a list with `headline` (character), `severity` (`"ok"`, `"info"`, `"warn"`, or `"critical"`), `pct_match` (0–100 numeric), `n_cells`, and `n_diffs`; the same summary shown by `print()`.
- `options`, `tolerance`: the inputs (echoed for the manifest).
- `manifest`: input hashes + run metadata.

Inspect with `print()`, `summary()`, `tibble::as_tibble()`, or render via `ks_report_html()` / `ks_report_xlsx()`.

Pipeline

1. **Schema alignment** — columns are paired by an explicit `mapping` first, then by exact name match. Unmatched columns become `base_only` / `comp_only` and are reported in `cmp$schema_diff`.
2. **Key resolution** — `by` is taken literally if supplied, auto-inferred from shared columns when `by = "auto"`, or rows are matched by row position when `by = NULL`.
3. **Row matching** — keyed-unique join, duplicate-key resolution (`dup_keys`), or position-zip; details land in `cmp$meta$matching` (also displayed by `print()` and in the HTML report).
4. **Cell diff** — per-column equality respecting `ks_tol()` (abs / rel / ULP) for numerics, `ks_comp_options()` for strings / NAs / SAS special missings, and `vctrs::vec_cast()` for compatible types.

5. **Pattern detection** — recurring shapes across diffs (constant offset, sign flip, trim-only, …) are scored and surfaced on `cmp$pattern_summary`.
6. **Manifest** — `xxhash64` digests of inputs / options / tolerance plus a timestamp and package version, for QC traceability.

Examples

```
a <- data.frame(id = 1:3, x = c(1, 2, 3), y = c("a", "b", "c"))
b <- data.frame(id = 1:3, x = c(1, 2, 4), y = c("a", "B", "c"))

# Strict compare on `id`
cmp <- ks_compare(a, b, by = "id")
cmp
tibble::as_tibble(cmp)

# Tolerant numeric compare
ks_compare(
  data.frame(id = 1, x = 1.0),
  data.frame(id = 1, x = 1.0001),
  by = "id",
  tolerance = ks_tol(abs = 1e-3)
)

# Renamed key column
ks_compare(
  data.frame(USUBJID = "S001", x = 1),
  data.frame(SUBJID  = "S001", x = 1),
  by = c(USUBJID = "SUBJID")
)
```

`ks_glance` *Glance at a ks_comparison*

Description

Returns a one-row tibble with the same headline counts as `summary.ks_comparison()`. Convenient for binding many comparisons together in a QC loop.

Usage

```
ks_glance(x, ...)
```

Arguments

`x`	A `ks_comparison`.
`...`	Unused.

Value

A one-row tibble with columns `n_base_rows`, `n_comp_rows`, `n_matched_rows`, `n_base_only_rows`, `n_comp_only_rows`, `n_matched_columns`, `n_value_diffs`, `n_columns_with_diffs`.

Examples

```
cmp <- ks_compare(
  data.frame(id = 1:2, x = c(1, 2)),
  data.frame(id = 1:2, x = c(1, 3)),
  by = "id"
)
ks_glance(cmp)
```

`ks_match_health` *Match-quality health check*

Description

Computes a small set of metrics that flag *suspicious* matching: high diff density, many fully-different rows, duplicate keys resolved positionally, and similar patterns that often indicate the wrong key was used.

Usage

```
ks_match_health(x)
```

Arguments

`x`	A `ks_comparison`.

Value

A list with components:

- `diff_density`: `n_value_diffs / (n_matched_rows * n_matched_columns)` (0-1).
- `n_fully_diff_rows`: matched rows where every matched column differs (or, more permissively, >= 80% of columns differ).
- `pct_fully_diff_rows`: as proportion of matched rows.
- `n_value_to_na`, `n_na_to_value`: structural NA transitions.
- `dup_keys`: `TRUE` if either side had duplicate key values.
- `dup_positional`: `TRUE` if duplicates were paired positionally (`keep_all`) — the most error-prone path.
- `row_count_delta`: `n_comp_rows - n_base_rows` (signed).
- `flags`: character vector of human-readable warnings (possibly empty).
- `severity`: `"ok"`, `"info"`, `"warn"`, or `"critical"`.

See Also

ks_recommendations(), ks_compare().

Examples

```
a <- data.frame(id = c(1, 1, 2), x = c(1, 2, 3))
b <- data.frame(id = c(1, 1, 2), x = c(9, 9, 3))
cmp <- ks_compare(a, b, by = "id", dup_keys = "keep_all")
ks_match_health(cmp)
```

---

ks_recommendations *Actionable recommendations for a comparison*

Description

Synthesises ks_match_health() flags, ks_suggest_key() hints, and a few other heuristics into a short list of user-facing "what to look at first" recommendations.

Usage

```
ks_recommendations(x)
```

Arguments

x	A ks_comparison.

Value

A tibble with one row per recommendation:

- severity ("critical", "warn", "info", "ok")
- title short label
- message longer human-readable text
- action suggested next step (may be NA)

See Also

ks_match_health(), ks_suggest_key().

`ks_report_html` *Render a self-contained HTML report*

Description

Produces a polished, single-file HTML report summarising a `ks_comparison`. The output uses `htmltools` for layout and `reactable` for interactive tables. No internet access, no Quarto, and no Pandoc are required.

Usage

```
ks_report_html(
  x,
  path = NULL,
  title = "ksCompare report",
  subtitle = NULL,
  max_rows = 100L,
  theme = c("default", "slate"),
  group_by_key = FALSE,
  max_groups = 200L
)
```

Arguments

`x`	A `ks_comparison` object.
`path`	File path for the report. Three behaviours: • missing / `NULL` (default): a filename is auto-generated from the comparison's `base_name` (and, when distinct, `comp_name`) written into `options$path` (if set on `ks_comp_options()`) or the current working directory, e.g. `ksCompare_adsl_vs_adsl_qc.html`. • `NA`: nothing is written; the assembled `htmltools::tagList()` is returned for embedding in another document. • a character path: written to that path. A `.html` extension is appended if missing. A bare filename (no directory component) is resolved relative to `options$path` when that has been set on `ks_comp_options()`, otherwise relative to the working directory.
`title`	Title shown at the top of the report.
`subtitle`	Optional subtitle (e.g. study identifier or run id).
`max_rows`	Per-table row cap (default `100`). When the value-diff table exceeds this cap, the report shows a *smart stratified sample* covering each column and each distinct diff-cause (`note`), prioritising the largest numeric magnitudes (at least one example per column and per cause is always retained). A notice indicates the sample size and recommends `as_tibble(cmp)` for the full set. `as_tibble(x)` is never truncated.

`theme`	One of `"default"` (steel blue) or `"slate"` (neutral dark headers on light background).
`group_by_key`	Logical (default `FALSE`). When `TRUE` and a `by =` key was used in `ks_compare()`, the value-diff section is rendered as one collapsed `<details>` block per key value, sorted by number of diffs (most-affected first). For `dup_keys = "keep_all"` / `"all_pairs"`, each block also shows a *Pair* column so you can tell pairs apart. Silently falls back to the flat table when no `by =` was supplied (position match).
`max_groups`	Maximum number of key-value blocks to render when `group_by_key = TRUE` (default `200`). Excess groups are summarised in a closing notice.

Details

Layout: a left-hand sticky table of contents, a header with dataset names, a status pill, KPI cards, then sections for schema, row diff, value diff, patterns, and the run manifest. Tables use friendly column labels, column groups, and `tick/cross` markers instead of raw `TRUE/FALSE`.

Value

Invisibly returns `path` (or the assembled tag list when `path = NULL`).

Examples

```
## Not run:
  cmp <- ks_compare(iris, iris, by = "Species")

  # Explicit path
  ks_report_html(cmp, "report.html", title = "ADSL QC")

  # Auto-generated filename in working directory
  ks_report_html(cmp)

  # In-memory tagList for embedding
  tags <- ks_report_html(cmp, path = NA)

## End(Not run)
```

---

ks_report_xlsx *Render an Excel workbook report*

Description

Writes a multi-sheet workbook summarising a `ks_comparison`. Sheets: `Summary`, `Schema`, `KeyDiff`, `Values`, `Patterns`, `OUT_BASE`, `OUT_COMP`, `OUT_DIF`, `OUT_NOEQUAL`, `Manifest`. Cells with value differences are highlighted on the wide `OUT_DIF` sheet.

Usage

```
ks_report_xlsx(x, path = NULL, highlight = TRUE, threshold = 0)
```

Arguments

x	A ks_comparison object.
path	File path for the workbook. When NULL (default), the filename is auto-generated from the comparison's base_name / comp_name and placed in options$path (if set on ks_comp_options()) or the current working directory, e.g. ksCompare_adsl_vs_adsl_qc.xlsx. A .xlsx extension is appended if missing. A bare filename is resolved against options$path when that has been set.
highlight	If TRUE (default), apply conditional formatting to highlight numeric OUT_DIF cells whose magnitude is above threshold.
threshold	Numeric threshold for highlighting (default 0, so any non-zero diff is highlighted).

Value

Invisibly returns path.

Examples

```
## Not run:
  cmp <- ks_compare(iris, iris, by = "Species")
  ks_report_xlsx(cmp, "report.xlsx")

  # Auto-named output in a pinned folder
  cmp2 <- ks_compare(
    iris, iris, by = "Species",
    options = ks_comp_options(path = tempfile("ksCompare_"))
  )
  ks_report_xlsx(cmp2)

## End(Not run)
```

---

ks_row_diff_summary *Per-row diff summary*

Description

Tallies the number of cell differences per matched observation, sorted descending. Useful for spotting rows where most/all columns disagree (often a sign of a wrong row match, a duplicated key resolved differently, or a systemic shift on a single record).

Usage

```
ks_row_diff_summary(x, n = NULL)
```

Arguments

x	A ks_comparison.
n	Optional cap on the number of rows returned (default NULL = all). When set, rows beyond n are dropped.

Details

Only matched rows are reported; unmatched rows are available via ks_unmatched_rows().

Value

A tibble with one row per matched observation that has at least one cell diff:

- key_id, base_row, comp_row, key_label
- n_diffs: count of differing cells on this observation.
- columns: comma-separated sample of the affected columns (capped at 8 names). Sorted by n_diffs descending. Zero-row tibble when there are no value differences.

See Also

ks_compare(), ks_cause_summary().

Examples

```
a <- data.frame(id = 1:3, x = 1:3, y = 1:3)
b <- data.frame(id = 1:3, x = c(1, 9, 9), y = c(1, 9, 3))
cmp <- ks_compare(a, b, by = "id")
ks_row_diff_summary(cmp)
```

---

ks_set_comp_options *Set ksCompare global options*

Description

Thin wrapper around base::options() for the ksCompare.* namespace. Accepts the same argument names as ks_comp_options() (without the ksCompare. prefix) and forwards them into base::options(), returning the previous values invisibly so the call can be unwound with base::options() or withr::with_options().

Usage

```
ks_set_comp_options(...)
```

Arguments

...	Named arguments. Recognised names match the formals of ks_comp_options(): na_equal, sas_special_missing, compare_labels, compare_formats, str_trim, str_case, str_norm, tz, path. A single unnamed ks_comp_options object may also be passed in, in which case all of its fields are pushed.

Details

Globals set this way become the new defaults of `ks_comp_options()` (and therefore of `ks_compare()` when no `options =` argument is supplied).

Value

Invisibly, a named list of the previous values (suitable for `do.call(options, prev)` to restore).

Examples

```
old <- ks_set_comp_options(path = tempfile("ksCompare_"), str_case = "fold")
ks_comp_options()$path
do.call(options, old)  # restore
```

---

ks_suggest_key *Suggest additional key columns when duplicates exist*

Description

Scans matched columns to find candidates whose combination with the current `by` key would make the join key unique on at least one side (and ideally both). Returns columns ranked by how much they reduce duplicate-key cardinality. Useful when `ks_compare()` used `dup_keys = "keep_all"` and the result looks suspicious.

Usage

```
ks_suggest_key(x, base = NULL, comp = NULL, top_n = 10L)
```

Arguments

`x`	A `ks_comparison`.
`base`, `comp`	Optional: the original input data frames used to build `x`. Required to inspect candidate column uniqueness.
`top_n`	Maximum number of candidate columns to return.

Details

Because `ks_compare()` does not snapshot the input data, the original `base` and `comp` frames must be passed in to inspect candidate columns. When `base` / `comp` are not supplied (or do not contain the original columns) an empty tibble is returned.

Value

A tibble with columns `column`, `pct_unique_base`, `pct_unique_comp`, `would_make_unique`. Zero rows when no key is in use, no duplicates exist, or no improvement is possible.

See Also

`ks_match_health()`, `ks_compare()`.

`ks_sysinfo` *SAS SYSINFO-compatible bitmask for a comparison*

Description

SAS's `&SYSINFO` macro variable encodes the result of `PROC COMPARE` as a bitmask. `ks_sysinfo()` returns a compatible-style integer summarizing the same kinds of facts, plus a named integer vector showing which bits are set. Bit positions follow SAS documentation:

Usage

```
ks_sysinfo(x)
```

Arguments

`x`	A `ks_comparison` object.

Details

Bit	Mask	Meaning
1	1	Data set labels differ
2	2	Data set types differ
3	4	Variable has different informat
4	8	Variable has different format
5	16	Variable has different length
6	32	Variable has different label
7	64	Base data set has observation not in compare
8	128	Compare data set has observation not in base
9	256	Base data set has BY group not in compare
10	512	Compare data set has BY group not in base
11	1024	Base data set has variable not in compare
12	2048	Compare data set has variable not in base
13	4096	A value comparison was unequal
14	8192	Conflicting variable types
15	16384	BY variables do not match
16	32768	Fatal error - comparison not done

Length and informat are not currently tracked; their bits are always clear. `haven` does not surface SAS informats, and column "length" is not a meaningful R-side concept for numerics.

Value

An integer with class `ks_sysinfo`. Use `as.integer()` for the raw value or print to see the named bits.

Examples

```
a <- data.frame(id = 1:3, x = c(1, 2, 3))
b <- data.frame(id = 1:3, x = c(1, 2, 4), z = 1:3)
cmp <- ks_compare(a, b, by = "id")
ks_sysinfo(cmp)
as.integer(ks_sysinfo(cmp))
```

---

ks_tidy *Tidy a ks_comparison*

Description

Returns the long-format cell-level diff table. Equivalent to `as_tibble(cmp)` and identical to `cmp$value_diff` when `include_unmatched = FALSE`.

Usage

```
ks_tidy(x, ...)

## S3 method for class 'ks_comparison'
ks_tidy(x, include_unmatched = FALSE, ...)
```

Arguments

`x`	A `ks_comparison`.
`...`	Unused.
`include_unmatched`	Logical (default `FALSE`). When `TRUE`, append base-only / comp-only rows from `cmp$unmatched_rows`.

Details

When `include_unmatched = TRUE`, rows describing observations present on only one side of the comparison are appended to the result with `kind = "base_only"` / `"comp_only"` and `column_base` / `column_comp` set to `NA`. One row is added per unmatched observation, capped at `ks_compare(max_unmatched_rows = ...)`.

Value

A tibble of value differences, optionally with appended unmatched-row markers. Has columns `key_id`, `base_row`, `comp_row`, `column_base`, `column_comp`, `kind`, `base`, `comp`, `diff`, `na_flow`, `note`.

Examples

```
cmp <- ks_compare(
  data.frame(id = 1:2, x = c(1, 2)),
  data.frame(id = 1:3, x = c(1, 3, 4)),
  by = "id"
)
ks_tidy(cmp)
ks_tidy(cmp, include_unmatched = TRUE)
```

---

ks_tol *Tolerance specification for numeric comparisons*

Description

Builds a tolerance object consumed by `ks_compare()`. Two numeric values `a` and `b` are considered equal when **any** of the active rules pass:

Usage

```
ks_tol(abs = 0, rel = 0, ulp = 0, per_column = NULL)
```

Arguments

`abs`	Non-negative absolute tolerance (default `0` — strict equality).
`rel`	Non-negative relative tolerance, scaled by `max(abs(a), abs(b))` (default `0`).
`ulp`	Non-negative integer ULP tolerance. Typical values are `4`–`16`; defaults to `0`.
`per_column`	Optional named list of per-column `ks_tol()` overrides keyed by base-side column name.

Details

- `abs(a - b) <= abs`
- `abs(a - b) <= rel * max(abs(a), abs(b))`
- `ulp_distance(a, b) <= ulp` — IEEE-754 units in the last place, useful for catching floating-point round-trip noise without false positives on genuine differences.

Per-column overrides may be supplied via `per_column`, a named list whose names are **base-side** column names and whose values are themselves the result of `ks_tol()`. Columns not listed fall back to the top-level `abs` / `rel` / `ulp`.

Value

A `ks_tol` S3 list.

Examples

```
ks_tol(abs = 1e-9)
ks_tol(rel = 1e-6, per_column = list(price = ks_tol(abs = 0.005)))
ks_tol(ulp = 4)
```

---

ks_unmatched_rows *Unmatched rows from a comparison*

Description

Returns the rows that are present on only one side of a comparison (`"base_only"` or `"comp_only"`), with their original data columns. Mirrors the "Observations in only" tables produced by SAS `PROC COMPARE`.

Usage

```
ks_unmatched_rows(x, side = c("both", "base_only", "comp_only"))
```

Arguments

`x`	A `ks_comparison`.
`side`	One of `"both"` (default), `"base_only"`, or `"comp_only"`.

Details

The result is precomputed at `ks_compare()` time and capped by the `max_unmatched_rows` argument (default `100`). When the cap kicks in, the `truncated` attribute is set and the full counts are recorded on `n_total`.

Value

A tibble with columns `side`, `key_id`, `key_label`, `base_row`, `comp_row`, followed by the data columns from the side that holds each row (`base_row` is `NA` for `comp_only` rows and vice versa, making it trivial to look an observation up in the original `base` / `comp` frame). Empty (zero-row) tibble when there are no unmatched rows.

See Also

`ks_compare()`, `ks_tidy()`.

Examples

```
a <- data.frame(id = 1:3, x = c(1, 2, 3))
b <- data.frame(id = 2:4, x = c(2, 3, 4))
cmp <- ks_compare(a, b, by = "id")
ks_unmatched_rows(cmp)
```

`sas-out` *SAS PROC COMPARE-style output datasets*

Description

These helpers reshape the long-format `value_diff` table inside a `ks_comparison` into the four classic SAS PROC COMPARE output datasets:

Usage

```
as_outbase(x)

as_outcomp(x)

as_outdif(x)

as_outnoequal(x)
```

Arguments

`x`	A `ks_comparison` object.

Details

- `as_outbase()` — values from the base frame, only for matched rows that contain at least one differing column; one row per matched key, one column per compared variable.
- `as_outcomp()` — same shape as `as_outbase()` but values from the compare frame.
- `as_outdif()` — numeric difference (`base - comp`) for every matched row that contains at least one differing column; non-numeric cells are reported as `NA` with a `.note` column.
- `as_outnoequal()` — only the rows where at least one matched cell differs, in long format (one row per differing cell).

The shapes mirror SAS's `OUTBASE=`, `OUTCOMP=`, `OUTDIF=`, and `OUTNOEQUAL=` datasets in spirit; we do not replicate the exact metadata columns (`_TYPE_`, `_OBS_`, ...) but include `key_id` and the original key columns so the rows can be related back to the source frames.

Value

A tibble.

Examples

```
a <- data.frame(id = 1:3, x = c(1, 2, 3), y = c("a", "b", "c"))
b <- data.frame(id = 1:3, x = c(1, 2, 4), y = c("a", "B", "c"))
cmp <- ks_compare(a, b, by = "id")
as_outbase(cmp)
```

```
as_outcomp(cmp)
as_outdif(cmp)
as_outnoequal(cmp)
```

`summary.ks_comparison` *Summary statistics for a ks_comparison*

Description

Computes a small list of headline counts that drive the printed summary, the HTML report KPI cards, and CI gates.

Usage

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

Arguments

`object`	A `ks_comparison` returned by `ks_compare()`.
`...`	Unused; present for S3 conformance.

Value

A `ks_comparison_summary` list (also pretty-printed via `print()`) with components:

- `n_base_rows`, `n_comp_rows`: input row counts.
- `n_matched_rows`, `n_base_only_rows`, `n_comp_only_rows`: row counts after matching.
- `n_matched_columns`, `n_base_only_columns`, `n_comp_only_columns`: schema-side counts.
- `n_value_diffs`: number of differing cells in matched rows.
- `n_columns_with_diffs`: how many distinct columns hold those differences.

See Also

`ks_glance()` for a tibble version, `ks_tidy()` for the long cell-level table.

Examples

```
cmp <- ks_compare(
  data.frame(id = 1:3, x = c(1, 2, 3)),
  data.frame(id = 1:3, x = c(1, 2, 4)),
  by = "id"
)
summary(cmp)
```

Index