

Package ‘datadiff’

September 30, 2026

Title Data Validation Based on 'YAML' Rules

Version 0.6.1

Description A comprehensive data validation package that allows comparing datasets using configurable validation rules defined in 'YAML' files. Built on top of the 'pointblank' package for robust data validation, it supports exact matching, tolerance-based numeric comparisons, text normalization, and row count validation.

License MIT + file LICENSE

Depends R (>= 4.0.0)

Imports dplyr (>= 1.0.0), pointblank (>= 0.12.3), rlang (>= 0.4.0), tidyselect (>= 1.0.0), utils, yaml (>= 2.2.0)

Suggests arrow, DBI, dbplyr, duckdb, gt, knitr, rmarkdown, RSQLite, stringr, testthat (>= 3.0.0), tibble

VignetteBuilder knitr

URL <https://github.com/ThinkR-open/datadiff>

BugReports <https://github.com/ThinkR-open/datadiff/issues>

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Author Vincent Guyader [cre, aut] (ORCID:
<<https://orcid.org/0000-0003-0671-9270>>),
ThinkR [cph],
Agence technique de l'information sur l'hospitalisation [spn]

Maintainer Vincent Guyader <vincent@thinkr.fr>

Repository CRAN

Date/Publication 2026-09-30 19:40:02 UTC

Contents

add_tolerance_columns	2
analyze_columns	3
compare_datasets_from_yaml	4
datadiff_report_html	7
derive_column_rules	8
detect_column_types	9
normalize_text	9
preprocess_dataframe	10
print.datadiff_coverage	11
print.datadiff_report	11
read_rules	12
setup_pointblank_agent	12
validate_row_counts	15
write_rules_template	16
Index	18

add_tolerance_columns	<i>Add tolerance columns for numeric comparisons</i>
-----------------------	--

Description

Creates additional columns in the comparison dataframe to handle numeric tolerance validation. For each numeric column with tolerance rules, adds absolute difference and threshold columns, plus a boolean column indicating if values are within acceptable tolerance.

Usage

```
add_tolerance_columns(cmp, tol_cols, col_rules, ref_suffix, na_equal)
```

Arguments

cmp	Comparison dataframe containing candidate and reference columns
tol_cols	Character vector of column names that have tolerance rules
col_rules	List of column-specific validation rules
ref_suffix	Suffix used for reference columns (default: "__reference")
na_equal	Logical indicating if NA values should be considered equal

Value

Modified dataframe with added tolerance columns

Examples

```
cmp <- data.frame(value = c(1.1, 2.2), value__reference = c(1.0, 2.0))
rules <- list(value = list(abs = 0.2, rel = 0.1))
add_tolerance_columns(cmp, "value", rules, "__reference", TRUE)
```

analyze_columns	<i>Analyze column differences between datasets</i>
-----------------	--

Description

Compares column names between reference and candidate datasets to identify common columns, missing columns, and extra columns. Can ignore specified columns.

Usage

```
analyze_columns(data_reference, data_candidate, ignore_columns = character(0))
```

Arguments

`data_reference` Reference dataframe
`data_candidate` Candidate dataframe to compare
`ignore_columns` Character vector of column names to ignore during comparison

Value

A list containing:

`cols_reference` Column names in reference data (excluding ignored)
`cols_candidate` Column names in candidate data (excluding ignored)
`missing_in_candidate`
Columns present in reference but missing in candidate
`extra_in_candidate`
Columns present in candidate but not in reference
`common_cols` Columns present in both datasets
`ignored_cols` Columns that were ignored from comparison

Examples

```
ref <- data.frame(a = 1, b = 2, c = 3, d = 4)
cand <- data.frame(a = 1, b = 2, c = 3, e = 5)
analyze_columns(ref, cand, ignore_columns = c("d", "e"))
```

compare_datasets_from_yaml

Compare datasets using YAML validation rules

Description

Main function for comparing reference and candidate datasets using configurable validation rules defined in a YAML file. Supports exact matching, tolerance-based comparisons, text normalization, and row count validation.

Usage

```
compare_datasets_from_yaml(
  data_reference,
  data_candidate,
  key = NULL,
  path = NULL,
  warn_at = 1e-14,
  stop_at = 1e-14,
  ref_suffix = "__reference",
  label = NULL,
  error_msg_no_key = "Without keys, both tables must have the same number of rows.",
  lang = getOption("datadiff.lang", "fr"),
  locale = getOption("datadiff.locale", "fr_FR"),
  extract_failed = TRUE,
  get_first_n = NULL,
  sample_n = NULL,
  sample_frac = NULL,
  sample_limit = 5000,
  duckdb_memory_limit = "8GB"
)
```

Arguments

data_reference	Reference dataframe, tibble, or lazy table (tbl_lazy)
data_candidate	Candidate dataframe to validate against reference
key	Optional character vector of column names to use as join keys for ordered comparison; NULL (the default) defers to the YAML keys field, and a positional comparison follows when neither gives a key. An empty vector is an error. Every key column must exist in both datasets; otherwise an error is raised naming the missing column(s) and the dataset(s) concerned. See the "Argument vs YAML precedence" section.
path	Path to YAML file containing validation rules. If NULL, default rules are generated automatically based on the reference dataset structure.
warn_at	Warning threshold as fraction of failing tests (default: 1e-14)

stop_at	Stop threshold as fraction of failing tests (default: 1e-14)
ref_suffix	Suffix for reference columns in comparison dataframe (default: "__reference")
label	Descriptive label for the validation report. See the "Argument vs YAML precedence" section.
error_msg_no_key	Text of the error raised when a positional (key-less) comparison receives datasets with different row counts; the actual row counts of both datasets are appended to this text.
lang	Language code for pointblank reports. Defaults to the <code>datadiff.lang</code> option if set, otherwise <code>"fr"</code> . Override globally with <code>options(datadiff.lang = "en")</code> . Supported values include <code>"en"</code> (English), <code>"fr"</code> (French), <code>"de"</code> (German), <code>"it"</code> (Italian), <code>"es"</code> (Spanish), <code>"pt"</code> (Portuguese), <code>"zh"</code> (Chinese), <code>"ja"</code> (Japanese), <code>"ru"</code> (Russian), etc. See pointblank documentation for full list.
locale	Locale code for number and date formatting. Defaults to the <code>datadiff.locale</code> option if set, otherwise <code>"fr_FR"</code> . Override globally with <code>options(datadiff.locale = "en_US")</code> . Examples: <code>"en_US"</code> , <code>"en_GB"</code> , <code>"de_DE"</code> , <code>"es_ES"</code> , <code>"pt_BR"</code> , <code>"zh_CN"</code> , <code>"ja_JP"</code> .
extract_failed	Logical indicating whether to collect rows that failed validation (default: TRUE). Set to FALSE to reduce memory usage for large datasets with many errors.
get_first_n	Integer specifying the maximum number of failed rows to extract per validation step (default: NULL, meaning all). Useful to limit memory when many rows fail.
sample_n	Integer specifying a fixed number of failed rows to randomly sample per validation step (default: NULL). Alternative to <code>get_first_n</code> for random sampling.
sample_frac	Numeric between 0 and 1 specifying the fraction of failed rows to sample (default: NULL). Used with <code>sample_limit</code> for proportional sampling.
sample_limit	Integer specifying the maximum number of rows when using <code>sample_frac</code> (default: 5000). Acts as a ceiling for sampled rows.
duckdb_memory_limit	Character string passed to DuckDB's <code>SET memory_limit</code> when Arrow datasets are used (default: <code>"8GB"</code>). Controls how much RAM DuckDB may use before spilling intermediate results to <code>tempdir()</code> . The default leaves headroom for R, Arrow, and the OS alongside DuckDB. Raise it (e.g. <code>"16GB"</code>) on machines with ample free RAM to reduce disk I/O; lower it (e.g. <code>"4GB"</code>) when memory is very constrained. The value is always validated as a size literal (an invalid one is an error on every path), but it only takes effect when Arrow datasets are used: plain <code>data.frames</code> or <code>tbl_lazy</code> inputs ignore a valid value.

Details

Structural failures. Three situations produce a check that always fails, one row in coverage and one failing step in the agent, without looking at the values: a column present in the reference but missing in the candidate (`missing_column`); a column whose type differs between the two datasets, integer and numeric being compatible (`type_mismatch`); and a tolerance rule (`abs` or `rel`, whether it comes from `by_name` or from a `by_type` block) on a column whose type in the reference is neither integer nor numeric, including character, date, datetime and logical

(`tolerance_on_non_numeric`). `{datadiff}` never converts the data, so such a rule cannot be honoured; rather than silently comparing the column as text, it is reported, a warning is raised, the column takes part in no other check and the rule is dropped from `applied_rules`. To compare the column by value, make it numeric in both datasets before the call; to compare it as text, remove the rule.

Value

A list of class `datadiff_result` containing:

<code>all_passed</code>	Logical; TRUE when every check passed. The first element of the returned list and the single verdict consumers should read.
<code>agent</code>	The configured pointblank agent as built (NOT interrogated: the verdict lives in response, which is the interrogated one)
<code>response</code>	Interrogated pointblank agent (class <code>datadiff_report</code>): usable by <code>pointblank::all_passed()</code> / <code>get_data_extracts()</code> ; printing it lazily renders a full pointblank-style report from coverage (built on demand, memoized). Reading it under its historical name <code>reponse</code> still works but is deprecated and emits a warning (once per session); the alias will be removed in a future release.
<code>missing_in_candidate</code>	Columns missing in candidate data
<code>extra_in_candidate</code>	Extra columns in candidate data
<code>applied_rules</code>	Final column-specific rules applied. A tolerance rule that could not be honoured (see Details) is not listed.
<code>coverage</code>	A <code>datadiff_coverage</code> data.frame: one row per check actually performed (column, check type, n, n_failed, status), always produced at negligible cost so the verified checks stay visible even when the fast path skips the per-column agent. Check types: <code>col_exists</code> , <code>tolerance</code> , <code>equality</code> , <code>row_count</code> , and the structural failures <code>missing_column</code> , <code>type_mismatch</code> and <code>tolerance_on_non_numeric</code> .
<code>summary</code>	Aggregate counts from coverage (<code>n_checks</code> , <code>n_pass</code> , <code>n_fail</code> , <code>n_rows_failed_total</code> , <code>all_passed</code>).

Argument vs YAML precedence

Some settings can come both from an explicit argument and from the YAML rules file. The resolution is always: explicit argument first, then the YAML defaults section, then the built-in default.

Setting	Explicit argument	YAML defaults field	Built-in default
join key	key	keys (legacy alias: key)	none (positional)
report label	label	label	"Comparing candidate vs reference"

When the YAML contains both `keys` and the legacy singular key field, `keys` (the canonical field written by `write_rules_template()`) wins, whatever its value: `keys: ~` or `keys: []` next to `key: [id]` means "no key", not id.

The key argument must be a non-empty character vector (a non-character or empty value is an error: use NULL for a positional comparison), while YAML-sourced key values are coerced to character (YAML being stringly typed, keys: [2024] designates the column named "2024"). In the YAML, an absent keys field, keys: ~ and keys: [] all mean "no key".

Note for path = NULL: the auto-generated rules template itself carries a label ("Comparison with default rules", or the explicit label argument), so that is the label effectively used without a YAML file; the built-in default above only applies when a YAML file is supplied with an empty or missing defaults\$label.

Examples

```
# Create test data
ref <- data.frame(id = 1:3, value = c(1.0, 2.0, 3.0))
cand <- data.frame(id = 1:3, value = c(1.1, 2.1, 3.1))

# Compare datasets without YAML (uses default rules, positional comparison)
result <- compare_datasets_from_yaml(ref, cand)

# Compare datasets with key but without YAML
result <- compare_datasets_from_yaml(ref, cand, key = "id")

# Compare datasets with custom YAML rules
tmp <- tempfile(fileext = ".yaml")
write_rules_template(ref, key = "id", path = tmp)
result <- compare_datasets_from_yaml(ref, cand, key = "id", path = tmp)
result$response
```

datadiff_report_html	<i>Render a comparison result as a pointblank HTML report</i>
----------------------	---

Description

Builds a full pointblank-style report (one validation step per column) from the comparison coverage, populated with the pre-computed pass/fail counts - without re-running the slow per-column interrogation. Optionally writes it to an HTML file.

Usage

```
datadiff_report_html(res, file = NULL, extracts_dir = NULL)
```

Arguments

res	The list returned by <code>compare_datasets_from_yaml()</code> .
file	Optional path to write the report to as a self-contained HTML file (via <code>pointblank::export_report()</code>). When NULL, nothing is written.

`extracts_dir` Optional directory where the failing-row extracts are also written as plain CSV files, one per failing validation step, named `extract_<step>_<column>.csv` with a zero-padded 4-digit step number (e.g. `extract_0002_price.csv`). Column names are sanitized for the file system: any character outside `[A-Za-z0-9_.-]` becomes `_`. The CSV buttons inside the HTML report are data: URI downloads, which some viewers block (Positron / Posit Workbench webview); files on disk are the robust alternative. The directory is created when needed; nothing is created when there is no extract (all-pass comparison or `extract_failed = FALSE`).

Value

The pointblank agent report object, invisibly.

<code>derive_column_rules</code>	<i>Derive column-specific validation rules</i>
----------------------------------	--

Description

Combines type-based and column-specific validation rules into a unified list of rules for each column in the dataset.

Usage

```
derive_column_rules(data_reference, rules)
```

Arguments

`data_reference` Reference dataframe used to determine column types

`rules` List of validation rules from `read_rules()`

Value

List of column-specific rules with type and name-based rules merged

Examples

```
df <- data.frame(a = 1:3, b = c(1.1, 2.2, 3.3))
rules <- list(by_type = list(numeric = list(abs = 0.1)), by_name = list(a = list(abs = 0.5)))
derive_column_rules(df, rules)
```

detect_column_types	<i>Detect column types in a dataframe</i>
---------------------	---

Description

Analyzes each column of a dataframe to determine its data type (integer, numeric, date, datetime, logical, or character)

Usage

```
detect_column_types(data)
```

Arguments

data	A dataframe or tibble
------	-----------------------

Details

Any column matching none of the first five types falls in the "character" bucket and receives the character rules. That is intentional for factors (compared as the character values they display, see [preprocess_dataframe\(\)](#)) and works for types whose == behaves like character (e.g. bit64::integer64). List-columns are not supported.

Value

A named character vector with column names as names and types as values

Examples

```
df <- data.frame(a = 1:3, b = c(1.1, 2.2, 3.3), c = c("x", "y", "z"))
detect_column_types(df)
```

normalize_text	<i>Normalize text for comparison</i>
----------------	--------------------------------------

Description

Applies text normalization transformations including case conversion and whitespace trimming. Non-character inputs are returned unchanged.

Usage

```
normalize_text(x, case_insensitive = FALSE, trim = FALSE)
```

Arguments

<code>x</code>	Vector to normalize (typically character)
<code>case_insensitive</code>	Logical indicating whether to convert to lowercase
<code>trim</code>	Logical indicating whether to trim leading/trailing whitespace

Value

Normalized vector of the same type as input

Examples

```
normalize_text(c(" Hello ", "WORLD "), case_insensitive = TRUE, trim = TRUE)
# Returns: c("hello", "world")
```

```
preprocess_dataframe
```

Preprocess dataframe according to column rules

Description

Applies preprocessing transformations to dataframe columns based on validation rules, such as text normalization for character columns. Supports both local data.frames and lazy tables (tbl_lazy) via `dplyr::mutate()`.

Usage

```
preprocess_dataframe(df, col_rules, schema = NULL)
```

Arguments

<code>df</code>	A dataframe or lazy table to preprocess
<code>col_rules</code>	A list of column-specific rules from <code>derive_column_rules()</code>
<code>schema</code>	Optional local data.frame with 0 rows used to determine column types when <code>df</code> is a lazy table. Obtained via <code>dplyr::collect(utils::head(df, 0L))</code> .

Details

On a local data.frame, every factor column (including key and non-compared columns) is first converted to character: factors are compared as the character values they display, so the text normalization rules apply to them and factors with differing level sets compare cleanly.

Value

Preprocessed dataframe (or lazy table) with transformations applied. Factor columns of a local data.frame come back as character vectors.

Examples

```
df <- data.frame(text_col = c(" HELLO ", "world"))
rules <- list(text_col = list(equal_mode = "normalized", case_insensitive = TRUE, trim = TRUE))
preprocess_dataframe(df, rules)
```

```
print.datadiff_coverage
```

Print method for a datadiff coverage summary

Description

Shows a one-line roll-up followed by the failing checks first, so an all-pass run still makes visible everything that was verified.

Usage

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

Arguments

x	A datadiff_coverage data.frame.
...	Unused.

Value

x, invisibly.

```
print.datadiff_report
```

Print method for a lazy datadiff report

Description

Prints an instant textual coverage summary, then renders the full pointblank-style report (HTML in interactive sessions / viewer). The report is built on first print and memoized.

Usage

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

Arguments

x	A datadiff_report (the response element of a comparison result).
...	Unused.

Value

x, invisibly.

read_rules	<i>Read and validate YAML rules file</i>
------------	--

Description

Loads validation rules from a YAML file and ensures the format is valid. Adds default values for missing configuration sections.

Usage

read_rules(path)

Arguments

path Character string specifying the path to the YAML rules file

Value

A list containing the parsed and validated rules configuration

Examples

```
tmp <- tempfile(fileext = ".yaml")
write_rules_template(data.frame(id = 1L, value = 1.0), key = "id", path = tmp)
rules <- read_rules(tmp)
```

setup_pointblank_agent	<i>Setup pointblank agent for data validation</i>
------------------------	---

Description

Creates and configures a pointblank validation agent with all necessary validation steps including column existence checks, exact value comparisons, and tolerance validations.

Usage

```

setup_pointblank_agent(
  cmp,
  cols_reference = NULL,
  common_cols,
  tol_cols,
  row_validation_info = NULL,
  ref_suffix,
  warn_at,
  stop_at,
  label,
  na_equal,
  lang = "fr",
  locale = "fr_FR",
  missing_in_candidate = character(0),
  type_mismatch_cols = character(0),
  add_col_exists_steps = TRUE,
  tolerance_non_numeric_cols = character(0)
)

```

Arguments

<code>cmp</code>	Comparison dataframe with candidate and reference data
<code>cols_reference</code>	Deprecated and unused; supplying any non-NULL value raises a warning (an explicit NULL is silent) and the argument will be removed in a future release
<code>common_cols</code>	Character vector of equality columns to validate (the internal pipeline passes only the failing ones on its failure path)
<code>tol_cols</code>	Character vector of columns with tolerance rules
<code>row_validation_info</code>	List with row validation information from <code>validate_row_counts</code>
<code>ref_suffix</code>	Suffix for reference columns
<code>warn_at</code>	Warning threshold (fraction of failing tests)
<code>stop_at</code>	Stop threshold (fraction of failing tests)
<code>label</code>	Descriptive label for the validation
<code>na_equal</code>	Logical indicating if NA values are considered equal
<code>lang</code>	Language code for pointblank reports (default: "fr"). Supported values include "en" (English), "fr" (French), "de" (German), "it" (Italian), "es" (Spanish), "pt" (Portuguese), "zh" (Chinese), "ja" (Japanese), "ru" (Russian), etc. See pointblank documentation for full list.
<code>locale</code>	Locale code for number and date formatting (default: "fr_FR"). Examples: "en_US", "en_GB", "de_DE", "es_ES", "pt_BR", "zh_CN", "ja_JP".
<code>missing_in_candidate</code>	Character vector of columns missing in candidate dataset

type_mismatch_cols

Character vector of columns whose type differs between reference and candidate (e.g. numeric in reference, character in candidate). A dedicated failing validation step labelled `type_mismatch: <column>` is added for each such column.

add_col_exists_steps

Logical indicating whether to add `col_exists` validation steps for common columns (default: `TRUE`). Set to `FALSE` for the non-local (lazy table) path where `cmp` only contains pre-computed boolean columns, not the original data columns.

tolerance_non_numeric_cols

Character vector of columns carrying a tolerance rule (`abs` or `rel`) although they are not numeric in the reference. `{datadiff}` never converts data, so the rule cannot be honoured: a dedicated failing validation step labelled `tolerance_on_non_numeric: <column>` is added for each. The comparison pipeline keeps such a column out of `common_cols` and `tol_cols`, and out of `type_mismatch_cols`; this function does not enforce that exclusion. The step itself is table-wide: it fails on any `cmp`, including a zero-row one, and does not depend on the number of rows.

Details

The equality steps validate a `<col>__eq` boolean (one-sided NA fails, two-sided NA follows `na_equal`, matching the tolerance kernel and the lazy SQL): when `cmp` does not already carry it, it is derived on the fly for a local `data.frame`. Tolerance steps expect the `<col>__ok` booleans to be precomputed (see [add_tolerance_columns\(\)](#)). A lazy `cmp` must carry every boolean column already.

Value

Configured pointblank agent ready for interrogation

Examples

```
cmp <- data.frame(
  a = 1:3, a__reference = 1:3,
  b = c(1.1, 2.2, 3.3), b__reference = c(1.0, 2.0, 3.0),
  b__ok = c(FALSE, FALSE, FALSE)
)
row_info <- list(check_count = FALSE)
agent <- setup_pointblank_agent(
  cmp,
  common_cols = "a", tol_cols = "b", row_validation_info = row_info,
  ref_suffix = "__reference", warn_at = 0.1, stop_at = 0.1,
  label = "Test", na_equal = TRUE
)
pointblank::all_passed(pointblank::interrogate(agent))
```

validate_row_counts	<i>Validate row counts according to rules</i>
---------------------	---

Description

Checks if the number of rows in candidate data matches expectations defined in rules. Can validate against a fixed expected count or against reference data count with tolerance. Returns validation information that can be used by pointblank agents.

Usage

```
validate_row_counts(  
  data_reference_p,  
  data_candidate_p,  
  rules,  
  count_rows = TRUE  
)
```

Arguments

data_reference_p	Preprocessed reference dataframe
data_candidate_p	Preprocessed candidate dataframe
rules	List of validation rules containing row_validation settings
count_rows	Logical; when FALSE, the row counts are not computed and ref_count/cand_count are NA. On lazy tables each count is a full COUNT(*) scan, wasted when nothing consumes it (keyed comparison with check_count: false). Ignored when the rules activate check_count: the counts are then always computed, so the returned structure stays consistent (an active check never sees NA counts).

Value

A list with validation information for pointblank integration

Examples

```
ref <- data.frame(a = 1:3)  
cand <- data.frame(a = 1:3)  
rules <- list(row_validation = list(check_count = TRUE, expected_count = 3, tolerance = 0))  
validate_row_counts(ref, cand, rules)
```

write_rules_template *Create a YAML rules template for data validation*

Description

Generates a comprehensive YAML configuration file with default validation rules based on the structure and types of the reference dataset. The template includes rules for different data types, column-specific rules, and row validation settings.

Usage

```
write_rules_template(
  data_reference,
  key = NULL,
  label = NULL,
  path = "rules.yaml",
  version = 1L,
  na_equal_default = TRUE,
  ignore_columns_default = character(),
  check_count_default = TRUE,
  expected_count_default = NULL,
  row_count_tolerance_default = 0,
  numeric_abs = 1e-09,
  numeric_rel = 0,
  integer_abs = 0L,
  character_equal_mode = "exact",
  character_case_insensitive = FALSE,
  character_trim = FALSE,
  date_equal_mode = "exact",
  datetime_equal_mode = "exact",
  logical_equal_mode = "exact"
)
```

Arguments

data_reference	A dataframe or tibble used as reference for rule generation
key	Character vector specifying column name(s) to use as join key(s) for data comparison. NULL (the default) writes keys: ~ and means a positional comparison (row by row); an empty vector is an error.
label	Descriptive label for the validation report
path	Character string specifying the output YAML file path (default: "rules.yaml")
version	Numeric version of the rules format (default: 1)
na_equal_default	Logical indicating if NA values should be considered equal by default
ignore_columns_default	Character vector of column names to ignore during comparison by default

`check_count_default`
 Logical indicating if row count validation should be enabled by default
`expected_count_default`
 Numeric value specifying expected row count (NULL uses reference count)
`row_count_tolerance_default`
 Numeric tolerance for row count validation
`numeric_abs`
 Default absolute tolerance for numeric columns
`numeric_rel`
 Default relative tolerance for numeric columns
`integer_abs`
 Default absolute tolerance for integer columns
`character_equal_mode`
 Default comparison mode for character columns: "exact", or "normalized" which implies `case_insensitive = TRUE` and `trim = TRUE` unless those flags are set explicitly (an explicit value always wins over the mode)
`character_case_insensitive`
 Logical for case-insensitive character comparison
`character_trim`
 Logical for trimming whitespace in character comparison
`date_equal_mode`
 Default comparison mode for date columns. Only "exact" has an effect: text normalization applies to character columns only, so "normalized" is accepted but changes nothing for this type.
`datetime_equal_mode`
 Default comparison mode for datetime columns (same caveat as `date_equal_mode`)
`logical_equal_mode`
 Default comparison mode for logical columns (same caveat as `date_equal_mode`)

Value

The path to the written YAML file, returned invisibly.

Examples

```
df <- data.frame(id = 1:3, value = c(1.1, 2.2, 3.3), name = c("A", "B", "C"))
write_rules_template(df, key = "id", path = tempfile(fileext = ".yaml"))
```

Index

`add_tolerance_columns`, [2](#)
`add_tolerance_columns()`, [14](#)
`analyze_columns`, [3](#)

`compare_datasets_from_yaml`, [4](#)
`compare_datasets_from_yaml()`, [7](#)

`datadiff_report_html`, [7](#)
`derive_column_rules`, [8](#)
`detect_column_types`, [9](#)

`normalize_text`, [9](#)

`pointblank::export_report()`, [7](#)
`preprocess_dataframe`, [10](#)
`preprocess_dataframe()`, [9](#)
`print.datadiff_coverage`, [11](#)
`print.datadiff_report`, [11](#)

`read_rules`, [12](#)

`setup_pointblank_agent`, [12](#)

`validate_row_counts`, [15](#)

`write_rules_template`, [16](#)
`write_rules_template()`, [6](#)