Package 'trialdiff'

September 27, 2026

Title Clinical Trial Data-Cut Change Detection and Impact Assessment

Version 0.2.0

Description A transparent, rule-based framework for detecting changes between successive data cuts of clinical trial datasets, classifying those changes into clinically meaningful categories, tracing user-defined data lineage, and assessing which downstream analyses and outputs may be affected. The package is designed to complement existing low-level data frame comparison tools by adding clinical-trial-specific classification, lineage and impact-assessment layers on top of deterministic comparison. The rule-based classification is similar in spirit to the data validation infrastructure of van der Loo and de Jonge (2021) <doi:10.18637/jss.v097.i10>.

License MIT + file LICENSE

URL https://github.com/Hirujan-R/trialdiff, https://Hirujan-R.github.io/trialdiff/

BugReports https://github.com/Hirujan-R/trialdiff/issues

Depends R (>= 4.1)

Imports cli, dplyr, htmltools, jsonlite, rlang, tibble

Suggests admiral, cards, covr, diffdf, haven, igraph, knitr, lintr, metacore, pharmaverseadam, pharmaversesdtm, pkgdown, rmarkdown, rtables, styler, tern, testthat (>= 3.0.0), waldo, withr

Config/testthat/edition 3

Config/Needs/website pkgdown

Encoding UTF-8

Language en-GB

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Hirujan Rangaraj [aut, cre]

Maintainer Hirujan Rangaraj <hirujanrangaraj@gmail.com>

Repository CRAN

Date/Publication 2026-09-27 16:30:02 UTC

Contents

`add_edges` *Add edges to a lineage graph*

Description

Merges edges (from `lineage_edge()`, `td_output()`, a registry or any edge data frame) into an existing `td_lineage` object and rebuilds the node table. Existing provenance and review information is preserved.

Usage

```
add_edges(x, ..., source = NULL)
```

Arguments

x	A td_lineage object.
...	Edge tables to add.
source	Optional value used to tag the source of every added edge.

Value

A td_lineage object.

---

add_outputs *Add an analysis/output registry to a lineage graph*

Description

Add an analysis/output registry to a lineage graph

Usage

```
add_outputs(x, outputs, ...)
```

Arguments

x	A td_lineage object.
outputs	A registry from output_registry() (or a single td_output() result).
...	Additional registries.

Value

A td_lineage object.

Examples

```
lineage <- define_lineage(
  lineage_edge("ADLB.AVAL", "ADLB.CHG", relationship = "derives")
)
registry <- output_registry(
  td_output("MMRM", depends_on = c("ADLB.AVAL", "ADLB.CHG")),
  td_output("Table_14_2_1", depends_on = "MMRM", type = "output")
)
trace_dependencies(add_outputs(lineage, registry), from = "ADLB.AVAL")
```

adlb_cut1 *Synthetic ADLB data cuts*

Description

Two synthetic laboratory analysis datasets (ADLB) with a long structure keyed by subject, parameter and visit. The later cut adds records for new subjects and includes corrected, missing-to-value and value-to-missing laboratory values.

Usage

```
adlb_cut1

adlb_cut2
```

Format

A data frame with one row per subject/parameter/visit and variables:

USUBJID Unique subject identifier.
PARAMCD Parameter code.
PARAM Parameter name.
AVISIT Analysis visit.
TRT01P Planned treatment for period 01.
AVAL Analysis value.
BASE Baseline value.
CHG Change from baseline.
ABLFL Baseline flag.

An object of class `data.frame` with 384 rows and 9 columns.

adsl_adlb_lineage *Example ADSL/ADLB lineage*

Description

A `td_lineage` object linking treatment assignment, population flags and laboratory analysis values to downstream summaries, an MMRM analysis and outputs. Intended for use in examples and vignettes.

Usage

```
adsl_adlb_lineage
```

Format

A `td_lineage` object (see `define_lineage()`).

adsl_cut1 *Synthetic ADSL data cuts*

Description

Two synthetic subject-level analysis datasets (ADSL) representing two data cuts of the same study. No proprietary or real patient data is used. The later cut contains new subjects, a withdrawn subject, a treatment-assignment correction, age corrections, missingness changes, a new variable (REGION) and a label change for AGE.

Usage

```
adsl_cut1

adsl_cut2
```

Format

A data frame with one row per subject and variables:

STUDYID Study identifier.

USUBJID Unique subject identifier.

SUBJID Subject identifier within study.

SITEID Investigator site identifier.

AGE Age.

SEX Sex.

RACE Race.

TRT01P Planned treatment for period 01.

TRT01A Actual treatment for period 01.

SAFFL Safety population flag.

ITTFL Intention-to-treat population flag.

REGION Geographic region (later cut only).

An object of class `data.frame` with 31 rows and 12 columns.

`assess_impact` *Assess downstream impact of classified changes*

Description

Combines a classified change set with a lineage graph to identify which downstream variables, analyses and outputs *could* be affected. Impact is graded as `"definitely_affected"`, `"potentially_affected"` or `"unlikely"`, with an explicit rationale and the triggering change identifiers.

Usage

```
assess_impact(changes, lineage, policy = impact_policy(), ...)
```

Arguments

`changes`	A classified `tdiff` object (output of `classify_changes()`) or a change register with a `category` column.
`lineage`	A `td_lineage` object from `define_lineage()`.
`policy`	An `impact_policy()` object.
`...`	Reserved for future extensions.

Details

`trialdiff` never claims statistical impact. Every affected analysis or output is reported as requiring programmer/statistician review and, where appropriate, a rerun.

Value

An object of class `td_impact` with elements `impacts`, `by_subject`, `unlinked`, `summary`, `policy` and `meta`.

Examples

```
old <- data.frame(USUBJID = c("S1", "S2"), TRT01P = c("Placebo", "Drug A"))
new <- data.frame(USUBJID = c("S1", "S2"), TRT01P = c("Drug A", "Drug A"))
diff <- compare_cut(old, new, by = "USUBJID", dataset = "ADSL") |>
  classify_changes()
lineage <- define_lineage(
  lineage_edge("ADSL.TRT01P", "ADLB.TRT01P", relationship = "groups_by"),
  lineage_edge("ADLB.TRT01P", "MMRM", relationship = "models"),
  lineage_edge("MMRM", "Table_14_2_1", relationship = "reports")
)
assess_impact(diff, lineage)
```

as_igraph *Convert lineage to an igraph object*

Description

Requires the **igraph** package.

Usage

```
as_igraph(x, ...)
```

Arguments

x	A td_lineage object.
...	Unused.

Value

An igraph object.

as_json *Machine-readable report data*

Description

Machine-readable report data

Usage

```
as_json(x, pretty = TRUE, ...)
```

Arguments

x	A tdiff, td_impact or td_report object.
pretty	Pretty-print JSON.
...	Unused.

Value

For as_json(), a JSON string. For as_report_list(), a list.

as_register *Tidy register of every detected change*

Description

Flattens a `tdiff` object into a single long table with one row per change (added record, removed record, modified cell or schema change). This is the machine-readable form intended for QC pipelines.

Usage

```
as_register(x, ...)
```

Arguments

`x`	A `tdiff` object.
`...`	Unused.

Value

A tibble.

classify_changes *Classify detected changes*

Description

`classify_changes()` turns the raw output of `compare_cut()` into a clinically meaningful change register. Every change is assigned a transparent, rule-based category (see `td_default_rules()`) together with a human-readable explanation.

Usage

```
classify_changes(x, rules = td_default_rules(), context = list(), ...)
```

Arguments

`x`	A `tdiff` object returned by `compare_cut()`, or a change register returned by `as_register()`.
`rules`	A list of `td_rule()` objects. Defaults to `td_default_rules()`.
`context`	Optional named list of extra context passed to rules, for example `list(has_paramcd = TRUE)`. Context is merged with context derived from `x`.
`...`	Unused.

Value

When x is a tdiff, the same object with classification columns added to $added, $removed, $modified and $schema, a $register element, and class c("tdiff_classified", "tdiff"). When x is a register (data frame), a classified tibble.

Examples

```
old <- data.frame(USUBJID = c("S1", "S2"), TRT01P = c("Placebo", "Drug A"))
new <- data.frame(USUBJID = c("S1", "S2"), TRT01P = c("Drug A", "Drug A"))
compare_cut(old, new, by = "USUBJID", dataset = "ADSL") |>
  classify_changes()
```

---

compare_cut *Compare two data cuts of a clinical dataset*

Description

compare_cut() performs a deterministic, key-based comparison of two versions of the same clinical dataset (for example ADSL at two data cuts). It reports observations added, removed and modified, and separately reports schema-level changes (variables added/removed, type and label changes).

Usage

```
compare_cut(
  old,
  new,
  by,
  compare_labels = TRUE,
  compare_types = TRUE,
  tolerance = 1e-09,
  ignore_vars = NULL,
  name_old = "old",
  name_new = "new",
  dataset = NULL,
  subject_var = NULL,
  backend = c("trialdiff", "waldo", "diffdf"),
  ...
)
```

Arguments

old, new	Data frames (or tibbles) representing the earlier and later data cut. new is compared against old.
by	Character vector of key variables that uniquely identify an observation within each dataset. Common clinical keys include c("USUBJID", "PARAMCD", "AVISIT") for ADLB or c("USUBJID", "VISIT") for ADVS. Must be present in both datasets.

`compare_labels` Logical. If `TRUE` (default) variable label changes are recorded in the schema comparison.

`compare_types` Logical. If `TRUE` (default) variable type changes are recorded in the schema comparison.

`tolerance` Numeric tolerance used when comparing numeric values. Set to `0` for exact comparison.

`ignore_vars` Character vector of variables to exclude from the value comparison (for example technical or audit columns).

`name_old`, `name_new` Character labels used in reports.

`dataset` Optional dataset name (for example `"ADSL"`). Inferred from the `old`/`new` expressions when not supplied.

`subject_var` Optional subject identifier variable. When `NULL` the function looks for `USUBJID`, then falls back to the first key.

`backend` Low-level comparison backend. `"trialdiff"` uses the built-in engine; `"waldo"` additionally uses **waldo** for whole-column equality short-circuiting when it is installed; `"diffdf"` delegates value comparison to **diffdf** and translates its output into a `tdiff` object (schema comparison remains built in). The `"diffdf"` backend requires the **diffdf** package.

`...` Reserved for future extensions.

Details

The function is deliberately a *low-level* comparison engine. Clinical meaning is added by `classify_changes()`, lineage by `define_lineage()` and impact by `assess_impact()`.

Value

An object of class `tdiff`: a list with elements `meta`, `added`, `removed`, `modified`, `schema`, `summary` and `register`. See tdiff-object for details.

Examples

```
old <- data.frame(
  USUBJID = c("S1", "S2", "S3"),
  TRT01P = c("Placebo", "Drug A", "Placebo"),
  AGE = c(54, 61, 47),
  stringsAsFactors = FALSE
)
new <- old
new$TRT01P[1] <- "Drug A"
new$AGE[2] <- 62
diff <- compare_cut(old, new, by = "USUBJID", dataset = "ADSL")
diff
```

define_lineage *Define a data lineage graph*

Description

`define_lineage()` collects lineage edges into an auditable graph that can be queried with `trace_dependencies()` and used by `assess_impact()`. The graph is stored as a tidy edge list plus a node table; no external graph library is required, although `as_igraph()` is provided for users who prefer one.

Usage

```
define_lineage(
  ...,
  edges = NULL,
  nodes = NULL,
  dataset = NULL,
  variable = NULL,
  downstream = NULL,
  relationship = "derives"
)
```

Arguments

`...`	One or more edge tibbles created by `lineage_edge()`, or a single data frame of edges.
`edges`	Optional data frame of edges with columns `from`, `to` and optionally `from_type`, `to_type`, `relationship`, `condition`.
`nodes`	Optional node table with columns `node` and `type`, used to override inferred types and to attach labels.
`dataset`, `variable`, `downstream`	Convenience arguments matching the common workflow: declare that `downstream` objects depend on `dataset.variable`.
`relationship`	Relationship used with the convenience arguments.

Value

An object of class `td_lineage`.

Examples

```
lineage <- define_lineage(
  lineage_edge("ADSL.TRT01P", "ADLB.TRT01P", relationship = "derives"),
  lineage_edge("ADLB.AVAL", "MMRM", relationship = "models"),
  lineage_edge("MMRM", "Table_14_2_1", relationship = "reports")
)
trace_dependencies(lineage, from = "ADSL.TRT01P")
```

impact_policy *Impact-assessment policy*

Description

Controls how aggressively changes are mapped onto downstream analyses. The policy is explicit and auditable: no statistical significance is ever claimed. Analyses and outputs are flagged as *requiring review/rerun*, never as "statistically affected".

Usage

```
impact_policy(
  value_direct = "definitely_affected",
  value_indirect = "potentially_affected",
  record = "potentially_affected",
  schema = "potentially_affected",
  label = "unlikely",
  max_depth = Inf
)
```

Arguments

value_direct	Level for a variable derived directly from a changed variable.
value_indirect	Level for more distant value changes.
record	Level for added/removed observations.
schema	Level for schema changes other than label changes.
label	Level for label-only changes.
max_depth	Maximum lineage depth to traverse.

Value

An object of class td_policy.

lineage_edge *Define a lineage edge*

Description

A lineage edge states that to depends on from. Node identifiers use a simple convention:

Usage

```
lineage_edge(
  from,
  to,
  from_type = NULL,
  to_type = NULL,
  relationship = "derives",
  condition = NA_character_,
  label = NA_character_
)
```

Arguments

from, to	Character node identifiers. to depends on from.
from_type, to_type	Optional node types ("dataset", "variable", "analysis", "output"). Inferred when NULL.
relationship	Edge semantics. One of "derives", "groups_by", "filters", "summarises", "models", "reports", "transports" or a custom string.
condition	Optional condition restricting when the edge applies (for example "TRT01P == 'Drug A'"). Recorded for transparency; not evaluated.
label	Optional human-readable label.

Details

- a dataset is written as "ADSL";
- a dataset variable is written as "ADSL.TRT01P";
- an analysis is written as "MMRM";
- an output (TLF) is written as "Table_14_2_1".

Value

A tibble with one row and class td_edge.

---

lineage_from_metadata *Derive lineage from study metadata*

Description

lineage_from_metadata() builds a define_lineage() graph from a **metacore** object (or any list with the same tables). Dataset and variable nodes come from ds_vars/ds_spec, and dependency edges are parsed from the derivations and where columns of value_spec.

Usage

```
lineage_from_metadata(
  metadata,
  include_where = TRUE,
  include_sdtm = TRUE,
  aliases = NULL,
  overrides = NULL,
  ...
)
```

Arguments

metadata	A metacore object (from **metacore**) or a list with elements ds_vars, ds_spec, value_spec and derivations.
include_where	Logical. If TRUE (default), variables referenced in value_spec$where clauses are added as "filters" edges.
include_sdtm	Logical. If TRUE (default), DATASET.VARIABLE references to non-ADaM domains (for example DM.ARM) are kept as source nodes.
aliases	Optional data frame with columns token and node, and an optional dataset column to scope the alias to one dataset. Aliases resolve tokens that would otherwise be ambiguous or unknown.
overrides	Optional edge table (for example from output_registry()) merged into the generated graph with add_edges().
...	Reserved for future extensions.

Details

Parsing is deterministic and conservative: only tokens that resolve to known variables (or explicit DATASET.VARIABLE references) become edges. Every edge records its provenance, and references that cannot be resolved uniquely are returned in a review table rather than guessed. See lineage_review().

Value

A td_lineage object with additional elements review (references needing manual attention) and provenance.

Examples

```
metadata <- list(
  ds_spec = data.frame(dataset = c("ADSL", "ADLB"), stringsAsFactors = FALSE),
  ds_vars = data.frame(
    dataset = c("ADSL", "ADSL", "ADLB", "ADLB", "ADLB"),
    variable = c("USUBJID", "TRT01P", "USUBJID", "AVAL", "CHG"),
    stringsAsFactors = FALSE
  ),
  value_spec = data.frame(
    dataset = c("ADSL", "ADLB", "ADLB"),
```

```
    variable = c("TRT01P", "AVAL", "CHG"),
    derivation_id = c("MT.ADSL.TRT01P", "MT.ADLB.AVAL", "MT.ADLB.CHG"),
    where = c(NA, "PARAMCD == 'ALT'", NA),
    stringsAsFactors = FALSE
  ),
  derivations = data.frame(
    derivation_id = c("MT.ADSL.TRT01P", "MT.ADLB.AVAL", "MT.ADLB.CHG"),
    derivation = c("DM.ARM", "ADSL.TRT01P", "AVAL - BASE"),
    stringsAsFactors = FALSE
  )
)
lineage_from_metadata(metadata)
```

`lineage_provenance` *Provenance of metadata-driven lineage edges*

Description

Provenance of metadata-driven lineage edges

Usage

```
lineage_provenance(x)
```

Arguments

`x`	A `td_lineage` object.

Value

A tibble with the metadata source of each edge.

`lineage_review` *References needing review after metadata-driven lineage*

Description

References needing review after metadata-driven lineage

Usage

```
lineage_review(x)
```

Arguments

`x`	A `td_lineage` object, typically from `lineage_from_metadata()`.

Value

A tibble of unresolved or undetected references.

output_registry *Build an analysis/output registry*

Description

Collects one or more `td_output()` declarations into a single registry table that can be grafted onto a lineage graph with `add_outputs()`.

Usage

```
output_registry(...)
```

Arguments

`...`	One or more `td_output()` results.

Value

A tibble of registry edges.

Examples

```
registry <- output_registry(
  td_output("MMRM", depends_on = c("ADLB.AVAL", "ADLB.CHG")),
  td_output("Table_14_2_1", depends_on = "MMRM", type = "output")
)
registry
```

remove_edges *Remove edges from a lineage graph*

Description

Removes edges matching any supplied filter. Useful for overriding or replacing automatically generated lineage.

Usage

```
remove_edges(x, from = NULL, to = NULL, relationship = NULL, source = NULL)
```

Arguments

`x`	A `td_lineage` object.
`from`, `to`, `relationship`, `source`	Optional character vectors of values to match. `NULL` (default) ignores that field.

Value

A td_lineage object.

`report_diff` *Produce a change-review report*

Description

`report_diff()` renders a structured review report from a comparison, its classification and (optionally) an impact assessment. Human-readable HTML or Quarto output is supported alongside machine-readable JSON and list output for automated QC pipelines.

Usage

```
report_diff(
  diff,
  impact = NULL,
  classified = NULL,
  output = "html",
  format = c("html", "quarto", "json", "list"),
  title = NULL,
  max_rows = 25L,
  include_disclaimer = TRUE,
  ...
)
```

Arguments

`diff`	A `tdiff` object, ideally already passed through `classify_changes()`.
`impact`	Optional `td_impact` object from `assess_impact()`.
`classified`	Optional classified `tdiff` object. If supplied it takes precedence over `diff` for classification content.
`output`	Either a format (`"html"`, `"quarto"`, `"json"`, `"list"`) or a file path. When a path is given the format is inferred from the extension.
`format`	Report format. Ignored when `output` is a path.
`title`	Report title.
`max_rows`	Maximum number of rows shown per table in HTML output.
`include_disclaimer`	Include the regulatory/statistical disclaimer.
`...`	Reserved for future extensions.

Value

An object of class `td_report`. When a file path is supplied the report is written and the path is stored in `$path`.

td_categories *Category label lookup*

Description

Category label lookup

Usage

```
td_categories()
```

Value

A tibble with `category` and `label` columns.

td_default_rules *Default rule set*

Description

Default rule set

Usage

```
td_default_rules()
```

Value

A list of `td_rule()` objects ordered by priority.

td_output *Declare an analysis or output and its inputs*

Description

`td_output()` is the building block of an analysis/output registry. It states that an analysis or output depends on one or more upstream nodes (typically `dataset.variable` identifiers, but any node is allowed).

Usage

```
td_output(
  id,
  depends_on,
  type = c("analysis", "output"),
  relationship = NULL,
  condition = NA_character_,
  label = id
)
```

Arguments

id	Node identifier for the analysis or output (for example "MMRM" or "Table_14_2_1").
depends_on	Character vector of upstream node identifiers.
type	Either "analysis" or "output".
relationship	Edge relationship. Defaults to "models" for analyses and "reports" for outputs.
condition	Optional applicability condition (recorded, not evaluated).
label	Human-readable label.

Value

A tibble of registry edges.

---

td_rule *Change classification rules*

Description

Classification in trialdiff is transparent and rule-based. A *rule* is a named predicate evaluated against the long change register produced by as_register(). Rules are applied in priority order; the first matching rule determines the primary category, while every matched rule is retained in the category_all column so that nothing is hidden.

Usage

```
td_rule(name, label, test, priority = 100L, reason = NULL, description = NULL)
```

Arguments

name	Stable machine-readable category name (snake_case).
label	Human-readable category label.
test	A function function(register, context) returning a logical vector the same length as nrow(register).
priority	Integer priority; lower values are evaluated first.

reason	Optional function function(register, context) returning a character vector of explanations.
description	Optional longer description.

Value

td_rule() returns an object of class td_rule.

---

trace_dependencies *Trace downstream dependencies*

Description

Performs a breadth-first traversal of the lineage graph starting from from and returns every reachable node, its depth and the path taken.

Usage

```
trace_dependencies(
  x,
  from,
  direction = c("downstream", "upstream"),
  max_depth = Inf,
  include_self = FALSE,
  ...
)
```

Arguments

x	A td_lineage object.
from	Character vector of starting node identifiers.
direction	"downstream" (default) or "upstream".
max_depth	Maximum traversal depth.
include_self	Include the starting node(s) in the result.
...	Unused.

Value

A tibble with node, node_type, depth, path and edge_relationship.

Index