Package 'coreval'

November 11, 2026

Title Check Clinical Trial Data Against 'CDISC' Open Rules

Version 0.4.0

Description Finds conformance problems in clinical trial data without leaving R, using the openly published 'CORE' conformance rules: the 'CORE' 1.0 rules maintained by Verisian for 'SDTM', and the 'CDISC' Open Rules for 'SEND', 'TIG' and 'USDM'. Check a single dataset while you are still writing the code that builds it, or a whole study folder once it exists, and get the findings back as a tidy data frame pointing at the exact row and variable. Reads transport ('XPT'), 'SAS', comma-separated and 'Dataset-JSON' files, plus 'Define-XML' when present, and 'USDM' study-design documents. Covers rules for the 'SDTM', 'SEND', 'TIG' and 'USDM' standards. The rules are bundled inside the package, so nothing is downloaded and your data never leaves your machine: no internet, no API key, no account. When a rule cannot be checked - because it needs a dataset you did not supply, for instance - it is reported as skipped with the reason, never counted as a pass. Meant as a quick first pass before a qualified validation system, never as a replacement for one. An independent project: not affiliated with or endorsed by 'CDISC' or Verisian, and not a 'CORE'-certified conformance engine.

Copyright Hrach Gevorgyan holds the copyright in this package's own code. The bundled rule definitions under inst/extdata are derived from verisianHQ/core-contributor, copyright Verisian, and cdisc-org/cdisc-open-rules, copyright CDISC, and the bundled standards metadata and controlled terminology from cdisc-org/cdisc-rules-engine, copyright CDISC, and reference tables in inst/extdata/core1_reference.rds and the per-version SDTM Model in inst/extdata/sdtm_model_versions.rds from verisianHQ/cdisc-rules-engine, copyright CDISC, all used under their MIT licenses. Two further bundled components belong to others: inst/extdata/js/jsonata.min.js is JSONata, copyright IBM Corp., under the MIT license, and inst/extdata/schema/xml/xhtml-1.1 holds the XHTML Schema modules, copyright the World Wide Web Consortium, under the W3C's own permission notice. See inst/COPYRIGHTS for every required notice and for which files each covers.

License MIT + file LICENSE

URL https://hrach-gevorgyan.github.io/coreval/,
https://github.com/hrach-gevorgyan/coreval

BugReports https://github.com/hrach-gevorgyan/coreval/issues

Encoding UTF-8

Language en-US

Depends R (>= 4.1)

Imports data.table, haven

Suggests jsonlite, knitr, QuickJSR, rmarkdown, testthat (>= 3.0.0), writexl, xml2

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Hrach Gevorgyan [aut, cre, cph]

Maintainer Hrach Gevorgyan <hrach.gevorgyan@yandex.com>

Repository CRAN

Date/Publication 2026-11-11 13:40:02 UTC

Contents

check_dataset 3
check_study 5
filter_findings 7
list_ct_packages 8
list_rules 9
print.coreval_result 11
read_study 12
summary.coreval_result 13
write_findings 13

Index **16**

`check_dataset` *Check one dataset, without needing a study folder*

Description

For when you are writing the code that builds a domain and want to know what is wrong with it right now. Give it the data frame you already have open, or the path to a single file.

Usage

```
check_dataset(
  x,
  domain = NULL,
  standard = NULL,
  version = NULL,
  use_case = NULL,
  max_records = 1000,
  include_deprecated = FALSE,
  ct_package = NULL,
  substandard = NULL
)
```

Arguments

`x`	A data frame (or `data.table::data.table()`), or the path to one `.xpt`, `.sas7bdat` or `.csv` file. A CSV file records no types, so each column's is guessed: one of plain numbers is read as numeric, and a value with a leading zero such as `"007"` stays text. An identifier written as a plain number can then be reported as the wrong type; an `.xpt` file carries the types it was written with.
`domain`	Two-letter domain code, e.g. `"AE"`. Left as `NULL`, coreval takes it from your `DOMAIN` column, or the file name if there isn't one. For one file of a split dataset with no `DOMAIN` column, such as `ae1.xpt`, set it yourself.
`standard`	Required. The standard the data follows, e.g. `"SDTMIG"`, `"SENDIG"` or `"TIG"`. Only that standard's rules run: a SENDIG rule has nothing to say about an SDTM study. The report says how many rules were set aside.
`version`	The standard's version, e.g. `"3-4"`. A version older than any the rules are written for is checked against the oldest that has rules, and the report says so.
`use_case`	Optional use case (e.g. `"INDH"`), as in `list_rules()`.
`max_records`	Most records to keep per rule, default 1000. A rule that flags every row of a large dataset would otherwise produce more findings than anyone can read or Excel can hold. The true count is kept in `truncated`. Use `Inf` for every record.
`include_deprecated`	Also run retired rules: those CDISC has deprecated, and, for an SDTM study, CDISC's SDTM rules that CORE 1.0 no longer carries. `FALSE` by default: a retired rule has a current replacement, so running both can report the same defect twice.

`ct_package` Which CDISC Controlled Terminology package the study follows, e.g. `"sdtmct-2026-03-27"`. Rules that ask whether a value is a legal term need this, and are skipped with a reason without it - coreval will not pick a version for you, because terminology changes between releases and judging a study against one it never declared would both invent violations and hide real ones. Every package CDISC's engine carries, from 2014 on, is bundled; `list_ct_packages()` shows them.

`substandard` For `standard = "TIG"` only, and required there: whether the tobacco study is `"SDTM"` (clinical), `"SEND"` (nonclinical) or `"ADaM"`. A TIG rule's use case covers different datasets in each, so CDISC's engine will not check a TIG study without it, and neither will coreval.

Value

An object of class `coreval_result`: a list of the three tables `findings`, `skipped` and `truncated`, the same shape `check_study()` returns, so `write_findings()` and `filter_findings()` work on it unchanged. Because it has a class, typing the result's name prints a readable report rather than dumping the list.

`skipped` carries a `reason` for every rule that did not run - a dataset you did not supply, a missing Define-XML, or, for the rules that ask whether a value is an allowed term, CDISC's controlled terminology if you did not say which package the study follows - see `ct_package`. Nothing skipped is ever counted as a pass.

Provenance rides along as attributes: `checks_run` (how many rules were evaluated), `domains`, and `excluded_by_standard` (how many rules the `standard`/`version` filter set aside). `write_findings()` writes these into the file it saves.

What it cannot check on its own

Plenty of CDISC rules compare one dataset against another - an adverse event date against the subject's reference dates in `DM`, a visit against the trial design. Hand over a single dataset and those questions cannot be answered.

coreval does not guess. Those rules are skipped, and `$skipped` names the dataset each one wanted. Running them anyway would compare your data against columns that are not there and report problems that do not exist.

Most rules still run on a single dataset. But the ones that cannot are the cross-dataset checks, which are often the ones that matter.

So a short `$findings` table here does not mean the data is clean. It is a quick first pass, not a verdict. Run `check_study()` on the whole folder before drawing conclusions.

See Also

`check_study()` to check a whole study folder.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30") # 30 February is not a date
```

```
)
result <- check_dataset(ae, standard = "SDTMIG")
result$findings[result$findings$Value == "2024-02-30", ]

# Always look at what could not run:
nrow(result$skipped)
```

---

check_study *Check a whole study against CDISC Open Rules*

Description

Runs every rule that applies to every dataset in the study, including the ones that compare datasets against each other. Use this once the datasets exist as files; to check a single dataset while you are still writing the code that builds it, see `check_dataset()`.

Usage

```
check_study(
  study,
  standard = NULL,
  version = NULL,
  use_case = NULL,
  max_records = 1000,
  include_deprecated = FALSE,
  ct_package = NULL,
  usdm = FALSE,
  substandard = NULL
)
```

Arguments

study	A study folder path, or a study object from `read_study()`. Passing the path is the usual way; reading first is only worth it when you want to check the same large study more than once without re-reading it, or to look at what was parsed.
standard	Required. The standard the data follows, e.g. `"SDTMIG"`, `"SENDIG"` or `"TIG"`; `list_rules()` shows the rest. Rules are written per standard, and a study checked against all of them gets SEND and TIG problems reported on SDTM data, so coreval will not run without it, as CDISC's and Verisian's engines will not. A CORE test case that declares its standard in a `.env` file needs no argument.
version	The standard's version, e.g. `"3.4"`. Needs `standard` too, since a bare version is ambiguous across standards. A version older than any the rules are written for, such as SDTMIG 3.1.2, is checked against the oldest that has rules, and the report says so.
use_case	Optional use case (e.g. `"INDH"`) to further filter which rules apply, as in `list_rules()`.

`max_records` Most records to keep per rule, default 1000. A rule can flag every row - a missing `EPOCH` on a 200 000-row `LB` is 200 000 identical findings, more than Excel can hold. The true count is kept in `truncated` and the report shows it, so nothing is under-reported. Use `Inf` for every record.

`include_deprecated` Also run retired rules: those CDISC has deprecated, and, for an SDTM study, CDISC's SDTM rules that CORE 1.0 no longer carries. `FALSE` by default: a retired rule has a current replacement, so running both can report the same defect twice.

`ct_package` Which CDISC Controlled Terminology package the study follows, e.g. `"sdtmct-2026-03-27"`. Rules that ask whether a value is a legal term need this, and are skipped with a reason without it - coreval will not pick a version for you, because terminology changes between releases and judging a study against one it never declared would both invent violations and hide real ones. Every package CDISC's engine carries, from 2014 on, is bundled; `list_ct_packages()` shows them.

`usdm` Check a USDM study design document. `FALSE` by default, and only used when `study` is a folder path; see `read_study()`.

`substandard` For `standard = "TIG"` only, and required there: whether the tobacco study is `"SDTM"` (clinical), `"SEND"` (nonclinical) or `"ADaM"`. A TIG rule's use case covers different datasets in each, so CDISC's engine will not check a TIG study without it, and neither will coreval.

Details

Findings come back one row per (dataset, record, variable), pointing at the exact spot. Some rules ask about a dataset as a whole rather than a particular row - those leave `Record` blank. A few ask about the study as a whole, such as "is DM present at all?"; those are answered once and reported under `Dataset = "STUDY"` rather than repeated for every domain.

Rules comparing against a define.xml do run, as long as the study has one and the `xml2` package is installed. Without both, they are skipped with a reason instead of being run against columns that are not there, which would report problems that do not exist. The same goes for any rule needing an operator or join coreval does not implement yet.

Value

An object of class `coreval_result`, holding three tables. Because it has a class, typing the result's name prints a readable report rather than dumping the list, and provenance rides along as attributes: `checks_run`, `ran` (each rule and dataset checked, one row per check), `domains`, `excluded_by_standard`, and `define_only` (`TRUE` when the folder held a define.xml and no datasets, so only the rules comparing the define with the study ran).

- `findings` - what is wrong. One row per affected record and variable, with `Dataset`, `Record` (its row in the file, or for a rule about the variables themselves, such as a type or label check, the variable's position), `USUBJID` and `SEQ` (its subject and `--SEQ`, blank where the dataset has none or the row is not a record), `Variable`, `Value`, the `issue` in words, its `triage` and `rule_id`. `Not in dataset` under `Value` means the rule wanted a variable you do not have, which is usually the finding itself.

- skipped - what could not be checked, with a reason for each. Read this one: an empty findings table can mean clean data *or* rules that never ran, and they look identical otherwise. Reasons include a dataset the study does not have, a missing Define-XML, and a controlled terminology version that neither the study nor ct_package names. Nothing skipped is ever counted as a pass.
- truncated - rules that flagged more records than max_records kept, with how many they really found.

Progress

A large study takes long enough that silence looks like a hang, so an interactive session shows a progress bar naming the domain being checked and how far through the study it is:

```
AE         4/7  |====================        |  75%
```

The percentage is weighted by how many records each domain holds, not by a plain count of rules, because a check against a 161,600-row AE costs hundreds of times one against a 200-row SJ. It tracks elapsed time closely but is still an estimate - rules differ in cost among themselves too - so treat it as "roughly how far through", not a clock.

It is off in scripts and non-interactive runs, where it would only clutter a log. Turn it on or off with options(coreval.progress = TRUE) or FALSE.

Examples

```
dir <- tempfile("coreval_study_")
dir.create(dir)
haven::write_xpt(data.frame(USUBJID = c("1", "2"), AGE = c(30, 65)), file.path(dir, "dm.xpt"))

result <- check_study(dir, standard = "SDTMIG")
result$findings
unlink(dir, recursive = TRUE)
```

---

filter_findings *Narrow a result to the findings you care about*

Description

Saves writing subset() over $findings by hand, and - because it returns a result rather than a plain table - what comes back still prints as a readable report.

Usage

```
filter_findings(
  result,
  triage = NULL,
  dataset = NULL,
  rule = NULL,
  variable = NULL
)
```

Arguments

result	A result from check_dataset() or check_study().
triage	Keep only these triage levels. One or more of: • "wrong value" - the data contains something that breaks the rule, like a month of 13. Nothing about your study explains it away. Start here. • "missing required" - something the standard marks Required is absent. • "missing optional" - something Expected or Permissible is absent. Often legitimate: a screen-failure subject with no reference dates, a variable your raw data does not carry yet. These are coreval's own triage, not a CDISC severity: the rules carry no severity field, and this does not map onto anyone's scale.
dataset	Keep only these datasets, e.g. "AE".
rule	Keep only these rule ids, e.g. "CORE-000547".
variable	Keep only findings naming these variables, including a finding that lists the variable as missing (a Required variable the dataset lacks).

Value

A coreval_result holding the matching findings. $skipped is left alone: what could not be checked does not become less true because you narrowed what you are looking at.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30")
)
result <- check_dataset(ae, standard = "SDTMIG")

# Just the things that are definitely wrong:
filter_findings(result, triage = "wrong value")
```

---

list_ct_packages *Controlled Terminology packages coreval bundles*

Description

Which version of CDISC's Controlled Terminology a study follows decides whether a value is a legal term, and the answer changes between releases - SEX gained INTERSEX and lost UNDIFFERENTIATED. So coreval never picks one for you: pass the one your study declares as check_study(ct_package =) and this is the list to pick from.

Usage

```
list_ct_packages(family = NULL)
```

Arguments

family	Optional prefix to narrow to one terminology family, e.g. "sdtm" or "send" ("sdtmct" works too). An unknown family is an error.

Details

Only what a conformance rule can ask about is bundled - each codelist's submission value and C-code, its terms' submission values and C-codes, and whether it is extensible. Definitions and synonyms are not, which is how 438 MB of CDISC's own caches becomes half a megabyte here.

Value

A character vector of package names, oldest first.

Examples

```
head(list_ct_packages("sdtm"))
```

---

list_rules *Look up CORE rules*

Description

One way in for every question about the rule set: what rules exist, what a particular one checks, and which of them apply to a domain.

Usage

```
list_rules(
  id = NULL,
  domain = NULL,
  standard = NULL,
  version = NULL,
  use_case = NULL,
  include_deprecated = TRUE,
  substandard = NULL
)
```

Arguments

id	Return only these rules, e.g. "CORE-000547". This is how you look up a rule id the report gave you. An id is found whatever standard says, so a rule written for TIG or SEND is returned too.
domain	Return only rules that apply to this domain, e.g. "AE", resolving each rule's Scope > Classes and Scope > Domains.
standard	Return only rules for this standard, e.g. "SDTMIG". Matched exactly, so "SENDIG" does not pick up "SENDIG-DART".

`version` The standard's version, e.g. `"3.4"` (or `"3-4"`). Only narrows alongside `standard`, since a bare version is ambiguous across standards.

`use_case` Optional use case (e.g. `"INDH"`). Rules with no Use Case constraint always pass.

`include_deprecated` Include superseded rules. `TRUE` here, unlike `check_dataset()` and `check_study()`, which default to `FALSE`. The difference is deliberate: **listing is not running**. This function is the catalog of what is bundled, so hiding the retired rules would make `nrow(list_rules())` stop meaning "how many rules are there" - and `source` is right here to filter on. Checking is a different question, and there a superseded rule would report the same defect twice. Pass `FALSE` to see exactly what would run.

`substandard` For `standard = "TIG"` only: whether the tobacco study is `"SDTM"` (clinical), `"SEND"` (nonclinical) or `"ADaM"`, which decides the datasets a TIG rule's use case covers when `domain` is given. Optional here, unlike `check_study()`: left out, the use cases of all three apply.

Details

The columns are the same whatever you ask for, so the result is safe to filter, join and script against.

Value

A `data.table::data.table()`, one row per rule: `id`, the one-line `issue` it reports, its fuller `description`, the `guidance` sentence from the Implementation Guide it enforces, its `legacy_ids` (the ids the CDISC Conformance Rules and FDA Business Rules give it), `standard` and `standard_version`, `authority`, `rule_type`, `sensitivity`, `executability`, `source` and `status`.

The commit the bundled rules came from is on the result as `attr(x, "rules_version")`; `write_findings()` records it in every exported file.

What `source` tells you

Where a rule comes from, and what standing it has there:

- `"core_contributor"` - a CORE 1.0 SDTMIG or FDA Business Rule, from Verisian's core-contributor repository. These check an SDTM study.
- `"published"` - `Published/` in CDISC's cdisc-open-rules, fully tested.
- `"sendig_draft"`, `"sdtmig_draft"`, `"fda_business_rules_draft"` - drafts upstream that already ship test data.
- `"deprecated_dir"` - superseded by a published replacement. A check does not run it unless you ask, since running both reports the same defect twice.

Examples

```
# Everything
nrow(list_rules())

# What does the rule the report just named actually check?
list_rules(id = "CORE-000547")$issue
```

```
# What applies to AE under SDTMIG 3.4?
nrow(list_rules(domain = "AE", standard = "SDTMIG", version = "3.4"))

# Which snapshot of the rules is this?
attr(list_rules(), "rules_version")
```

`print.coreval_result` *Print a coreval check result as a readable report*

Description

Describes each problem in the rule's own words, worst first, with the rows and values that caused it. A whole-study result is grouped by dataset, with a summary first, so you can see where the trouble is before reading detail.

Usage

```
## S3 method for class 'coreval_result'
print(x, n = 10, rows = 3, guidance = FALSE, ...)
```

Arguments

`x`	A result from `check_dataset()` or `check_study()`.
`n`	Maximum problems to describe - per dataset, for a study result. The rest are counted, not listed. Default 10.
`rows`	Maximum example records to show per problem. Default 3.
`guidance`	Also print the sentence from the Implementation Guide that each rule enforces - the "why" behind it. Off by default: it roughly doubles the length of the report.
`...`	Ignored.

Value

`x`, invisibly.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30")
)
check_dataset(ae, standard = "SDTMIG")
```

read_study *Read a study into coreval's internal representation*

Description

Reads a study folder: SAS transport (`.xpt`) and SAS (`.sas7bdat`) files, Dataset-JSON (`.json`, `.ndjson`), or a USDM study document, plus a Define-XML when one is there. A CORE test-case `data/` folder (`_variables.csv` and one CSV per dataset) is read too. All of them come out in the same internal representation, so the evaluator never has to know which one it got.

Usage

```
read_study(path, usdm = FALSE)
```

Arguments

`path`	Directory path.
`usdm`	Read a USDM study design document. `FALSE` by default: USDM is not what most studies are checked for, and some of its rules take minutes on one file, so it has to be asked for. Without it, a folder holding only a USDM document is refused with a message saying so, never read as empty.

Details

Character columns use `""` for blank/missing (never `NA`) to match how SAS XPT round-trips blanks; numeric columns use `NA`. Column types are taken from the source (XPT's own types, or `_variables.csv`'s declared `Char`/`Num`) rather than guessed from the data, so numeric-looking identifiers (e.g. `"007"`) are never silently coerced.

Value

A study object: `list(datasets = <named list of domain -> list(data, meta, label, file_size)>, define, stf, define` is the parsed Define-XML if one was found in `path` and the `xml2` package is installed, and `NULL` otherwise; `stf` likewise holds the study tagging file (`stf*.xml`), one row per document file tag; `ct` is always `NULL`. Each `data` is a `data.table::data.table()`; each `meta` is a data.table with columns `variable`, `label`, `type`; `label` is the dataset's own label (e.g. `"Adverse Events"`), or `NA` if unavailable; `file_size` is the size in bytes of the file it was read from, which CORE 1.0's dataset size rules compare. `standard` is the study's declared standard/version (e.g. `list(product = "SDTMIG", version = "3-4")`), read from a CORE test case's `.env` file or a USDM document, and both `NA` for a folder of datasets, which does not declare one.

Examples

```
dir <- tempfile("coreval_study_")
dir.create(dir)
haven::write_xpt(data.frame(USUBJID = c("1", "2"), AGE = c(30, 65)), file.path(dir, "dm.xpt"))
study <- read_study(dir)
study$datasets$DM$data
```

```
unlink(dir, recursive = TRUE)
```

`summary.coreval_result` *Summarize a check result in a few lines*

Description

The short form of `print.coreval_result()`, for when you have run a check inside a script, or just want to know whether the last fix helped.

Usage

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

Arguments

`object`	A result from `check_dataset()` or `check_study()`.
`...`	Ignored.

Value

A one-row `data.table::data.table()`, invisibly, with the counts it printed - so it can be logged or compared.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30")
)
summary(check_dataset(ae, standard = "SDTMIG"))
```

`write_findings` *Write conformance findings to a file*

Description

Saves the result of `check_study()` to CSV or Excel, so findings can be shared with people who don't use R, tracked in a spreadsheet, or attached to a data-review document.

Usage

```
write_findings(result, path, tracking = TRUE)
```

Arguments

result	A result from check_dataset() or check_study().
path	Output file path. The extension decides the format: .xlsx for Excel, .csv (or anything else) for CSV.
tracking	Add the empty tracking columns. TRUE by default; set FALSE for a file you intend to read back into R.

Details

Both tables are always written, never just the findings. A short findings table can mean clean data, or it can mean many rules were skipped, and those two situations look identical if the skipped table is dropped:

- **Excel** (.xlsx) - four sheets laid out for a person to read:
  - Problems: one row per problem, definite errors first, then the ones touching most records. What is wrong and the rule's number, the variables it names (or the missing ones), and how many records it affects, or whole dataset.
  - Records: one row per affected record and problem, in the order of Problems, with USUBJID, SEQ and the values, such as AESTDTC = "2024-02-30". A dataset with no subject gives the row instead.
  - Not checked: the checks that could not run, grouped by reason, with what to do about each, then when and how the check was run.
  - Rules: every rule, and whether it found problems, found none or could not run. A rule that found nothing appears nowhere else.
- **CSV** (.csv) - the tables as they are in R, for reading back into a program. Findings go to path; the others go to sibling files with a suffix before the extension, so issues.csv gives issues_skipped.csv, issues_about.csv and issues_summary.csv (one row per problem).

An existing file at path, or at a sibling path, is replaced.

In CSV files, a text value that begins with =, +, - or @ and is not a plain number is written with a leading '. Findings quote the submission data, and a spreadsheet program opening the CSV would otherwise run such a value as a formula. Excel files store every value as text and need no such change. CSV files are written as UTF-8 with a byte order mark, which Excel needs to show non-English text correctly.

Excel output needs the writexl package. It is a Suggests, so if it isn't installed you get a clear message telling you to install it or use .csv instead, rather than a failure part-way through writing.

Value

The paths actually written, invisibly. One element for Excel (a single workbook). For CSV, one path per file written: the findings, plus siblings for skipped, about and summary, plus one for truncated when any rule flagged more records than were kept.

Columns for tracking

Empty columns are added for you to fill in by hand once the file is open: `Status` and `Notes` on `Problems` and `Notes` on `Records` in Excel, `Status`, `Owner` and `Notes` in CSV. They exist so a finding you have looked at and decided not to act on ("expected, see protocol deviation log") can be recorded next to the finding itself, rather than in a separate document nobody reads.

Examples

```
dir <- tempfile("coreval_study_")
dir.create(dir)
haven::write_xpt(data.frame(USUBJID = c("1", "2"), AGE = c(30, 65)), file.path(dir, "dm.xpt"))
study <- read_study(dir)
result <- check_study(study, standard = "SDTMIG")

out <- file.path(dir, "findings.csv")
written <- write_findings(result, out)
basename(written)

unlink(dir, recursive = TRUE)
```

Index

check_dataset, 3
check_dataset(), *5*, *8*, *10, 11*, *13, 14*
check_study, 5
check_study(), *4*, *8*, *10, 11*, *13, 14*

data.table::data.table(), *3*, *10*, *12, 13*

filter_findings, 7
filter_findings(), *4*

list_ct_packages, 8
list_rules, 9
list_rules(), *3*, *5*

print.coreval_result, 11
print.coreval_result(), *13*

read_study, 12
read_study(), *5, 6*

summary.coreval_result, 13

write_findings, 13
write_findings(), *4*, *10*