Package 'DCC'

September 26, 2026

Type Package

Title Data Cleaning Center for Survey and Assessment Data

Version 1.2.1

Description Rule-driven, auditable cleaning of survey and assessment response data, implementing the WeianData Detect-Execute-Report workflow. Provides a multi-format, multi-encoding input layer (CSV, 'Excel', 'SPSS', 'Stata', 'SAS', Parquet, JSON), the dcc_data container with a provenance chain, level-0 structural diagnostics, five built-in response-quality detectors (missing items, straight-lining, response time, trap items, score anomalies), a declarative YAML rule engine, an execution engine with a cell-level audit log, answer-key scoring, multi-form to master item bank mapping, a normalized report model rendered as bilingual staff workbooks and HTML, complete statistical bundles, and versioned machine JSON/JSONL with findings-to-changes reconciliation, cell-level lineage tracing, and manifest-based one-command reproduction. Includes a protected bilingual strict project workbook and matching JSON contract with cell-addressed validation, non-mutating preflight, preview-first execution, and localized staff guidance. All formally supported input backends install with the package; PDF is optional rather than a fixed report output.

License GPL (>= 2)

Copyright See file inst/COPYRIGHTS.

Encoding UTF-8

Language en

Depends R (>= 4.1)

Imports arrow, data.table (>= 1.14.0), haven (>= 2.5.5), jsonlite, openxlsx2 (>= 1.28), readODS (>= 2.3.5), readxl (>= 1.5.0), stringi (>= 1.7.0), methods, stats, tools, utils, writexl, yaml

Suggests knitr, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/weiandata/DCC

BugReports https://github.com/weiandata/DCC/issues

RoxygenNote 7.3.2

Config/DCC/Installation complete format backends in Imports; PDF optional

NeedsCompilation no

Author Kunxiang Ma [aut, cre], WEIAN DATA TECH (Beijing) Co., Ltd. [cph, fnd]

Maintainer Kunxiang Ma <makunxiang@weiandata.com>

Repository CRAN

Date/Publication 2026-09-25 22:40:07 UTC

Contents

`dcc_apply_codebook` *Apply a declarative codebook to a dataset*

Description

Applies a codebook to a dataset: per variable it can rename, recode values, declare missing codes, coerce type, and attach a label, value labels, and a role. The preview (`dry_run = TRUE`) describes every change and changes nothing; `dry_run = FALSE` returns a new `dcc_data` with a `codebook` provenance record. The raw input is never overwritten, and the preview and apply share one planner, so a change is previewed exactly as applied. Unknown variables and impossible type coercions raise `dcc_codebook_error`.

Usage

```
dcc_apply_codebook(x, codebook, dry_run = TRUE)
```

Arguments

x	A dcc_data object or data.frame.
codebook	A named list keyed by (current) variable name. Each entry is a list with any of: rename (new name), recode (a named old -> new value map), missing (values set to NA), type (target class), label, value_labels (a named vector), and role.
dry_run	If TRUE (default), return a dcc_codebook_preview; if FALSE, apply and return a new dcc_data.

Value

A dcc_codebook_preview (dry run) or a new dcc_data.

See Also

dcc_codebook_changes.

Examples

```
df <- data.frame(sid = c("S1", "S2"), age = c(25, -99),
                 sex = c("1", "2"), stringsAsFactors = FALSE)
cb <- list(
  age = list(missing = -99, type = "integer"),
  sex = list(rename = "gender", recode = c("1" = "M", "2" = "F"),
             label = "Gender")
)
dcc_apply_codebook(df, cb)
dcc_apply_codebook(df, cb, dry_run = FALSE)
```

---

dcc_audit_log *Accessors for dcc_result objects*

Description

Access the machine-readable audit log and the cleaned dataset produced by the Execute stage. The audit log is the backbone of DCC's auditable-reporting guarantee: every row records one change traceable to one finding.

Usage

```
dcc_audit_log(x)

dcc_cleaned(x)
```

Arguments

x	A dcc_result from dcc_execute.

Value

dcc_audit_log returns the cell-level audit log (data.table with columns record_id, variable, old_value, new_value, action, check_id, method, timestamp, dcc_version, ruleset_hash, keyfile_hash). dcc_cleaned returns the new dcc_data version.

Examples

```
df <- data.frame(sid = "S1", q1 = 9)
f <- dcc_findings("S1", variable = "q1", check_id = "C",
                  evidence = "legacy code")
res <- dcc_execute(df, f, actions = list(C = "set_na"), id_var = "sid")
dcc_audit_log(res)
dcc_cleaned(res)
```

---

dcc_capabilities *Machine-readable DCC capability document*

Description

Returns a versioned, deterministic description of every public capability, rule type, action type, and input format, plus the operations DCC deliberately does not support. General-purpose callers (including AI systems) can query it to discover what is Stable, Experimental, or Planned before building a pipeline. The status of each feature reflects the implemented state of the installed package, not a roadmap. Contract version 1.2 identifies invalid-numeric detection, declared YAML IDs, terminal dispositions, and atomic run publication as Stable capabilities.

Usage

```
dcc_capabilities()
```

Value

A named list with contract_version (the capability-contract version), package_version (the installed DCC version), features (a data.frame of name, status – "Stable"/"Experimental"/"Planned" – and since), rule_types, action_types, formats (a data.frame of format, status, extensions, backend, semantics, and limitations), and unsupported (operations DCC does not perform).

See Also

dcc_schema for the formal object schemas.

Examples

```
caps <- dcc_capabilities()
caps$action_types
caps$features[caps$features$status == "Stable", "name"]
```

dcc_check *Check a strict DCC project without changing data*

Description

Validates the plan, performs a strict canonical import, runs environment and data checks, and previews findings. It writes diagnostics and a bilingual staff report only; no action, cleaned dataset, audit log, or manifest is produced.

Usage

```
dcc_check(data, plan, output_dir)
```

Arguments

data	Existing source data file path.
plan	Strict .xlsx/.json plan path or a dcc_plan.
output_dir	New directory for check diagnostics. Required and never defaulted: the caller chooses every location DCC writes to.

Value

A dcc_check_result with status, validation, findings, imported data, and written files.

dcc_codebook_changes *The planned changes of a codebook preview*

Description

Returns the table of changes a codebook preview would apply.

Usage

```
dcc_codebook_changes(x)
```

Arguments

x	A dcc_codebook_preview from dcc_apply_codebook.

Value

A data.table of variable, op, and detail.

Examples

```
df <- data.frame(sid = "S1", age = -99)
prev <- dcc_apply_codebook(df, list(age = list(missing = -99)))
dcc_codebook_changes(prev)
```

dcc_config *A cleaning configuration*

Description

Bundles a rule set, an action map, a record-id column, and the item columns into a single object that dcc_run consumes.

Usage

```
dcc_config(rules, actions = list(), id_var = NULL, items = NULL)
```

Arguments

rules	A dcc_ruleset from dcc_rules.
actions	Named list mapping check_ids to actions (see dcc_execute).
id_var	Record-id column name, or NULL for row numbers.
items	Optional character vector of item column names.

Value

A dcc_config object.

See Also

dcc_run, dcc_validate_config.

Examples

```
rf <- tempfile(fileext = ".yaml")
writeLines(c("checks:", "  - id: R001", "    type: range",
             "    variable: score", "    min: 0", "    max: 100"), rf)
if (requireNamespace("yaml", quietly = TRUE)) {
  dcc_config(dcc_rules(rf), actions = list(R001 = "set_na"), id_var = "sid")
}
```

dcc_data *The dcc_data container*

Description

A dcc_data object bundles the dataset with its metadata, the level-0 read report, and an append-only provenance chain. Every DCC stage (detect, execute, report) receives and returns this container; stages append to the provenance chain and never rewrite it (raw data is immutable; cleaning produces new versions).

Usage

```
dcc_data(data, meta = list(), read_report = NULL, provenance = NULL,
  dictionary = NULL, missing_states = NULL, import_spec = NULL)
```

Arguments

data	A data.frame or data.table.
meta	Named list of source metadata (see dcc_read).
read_report	A dcc_read_report object, or NULL.
provenance	List of provenance records; normally created internally.
dictionary	Canonical variable dictionary with unique name values.
missing_states	Cell-level missing-state table using DCC's declared missing-state vocabulary.
import_spec	The dcc_import_spec that produced the canonical data, or NULL for data created directly in R.

Value

An object of class dcc_data: a list with elements data (a data.table), meta, read_report, provenance, dictionary, missing_states, and import_spec.

Examples

```
x <- dcc_data(data.frame(id = 1:3, score = c(90, 85, 77)))
dcc_provenance(x)
```

dcc_detect *Run a rule set against data (Detect stage)*

Description

Evaluates every check in a rule set and returns the combined findings table. Detection is pure and read-only: the same data and the same rule set always produce the same findings, and the input is never modified. Range checks report non-missing values that cannot be converted to numeric with code `INVALID_NUMERIC`; numeric bounds violations use `OUT_OF_RANGE`.

Usage

```
dcc_detect(x, rules, id_var = NULL)
```

Arguments

`x`	A `dcc_data` object or data.frame.
`rules`	A `dcc_ruleset` from `dcc_rules`.
`id_var`	Name of the record-id column, or `NULL` for row numbers.

Value

A `dcc_findings` table. If `x` is a `dcc_data`, the result carries a `dcc_data` attribute: the input container with a `detect` provenance record (rule file hash, findings count) appended.

Examples

```
df <- data.frame(sid = c("S1", "S2"), score = c(50, 150))
f <- tempfile(fileext = ".yaml")
writeLines(c(
  "checks:",
  "  - id: R001",
  "    type: range",
  "    variable: score",
  "    min: 0",
  "    max: 100"
), f)
if (requireNamespace("yaml", quietly = TRUE)) {
  dcc_detect(df, dcc_rules(f), id_var = "sid")
}
```

dcc_detect_chunked *Run record-local checks over a file in chunks*

Description

Larger-than-memory detection with an adaptive backend: chunks (or Arrow record batches) of `chunk_size` rows are read and checked one at a time, so peak memory is bounded by the chunk. Findings are identical to an in-memory `dcc_detect` for record-local checks (`range`, `set`, `missing_items`, `straightlining`, `trap_items`, and `response_time` with the median-relative cut disabled via `min_median_ratio: ~`). Cross-record checks (`score_anomaly`, median-relative response time) are rejected with typed errors; `expr` rules must not use aggregate functions.

Usage

```
dcc_detect_chunked(path, rules, chunk_size = 100000L, id_var = NULL,
  sep = NULL, backend = c("auto", "csv", "arrow"), encoding = "auto")
```

Arguments

`path`	Path to the input file: a delimited text file (CSV/TSV) for the `csv` backend, or a Parquet/Feather file for the `arrow` backend.
`rules`	A `dcc_ruleset` from `dcc_rules`.
`chunk_size`	Rows per chunk / record batch (default 100000).
`id_var`	Name of the record-id column, or `NULL` to use global row numbers (consistent across chunks).
`sep`	Field separator for the `csv` backend. `NULL` (default) infers the separator from the extension – a tab for `.tsv`, a comma otherwise; pass an explicit value to override. Ignored by the `arrow` backend.
`backend`	One of `"auto"` (default), `"csv"`, or `"arrow"`.
`encoding`	Encoding of the `csv` backend input: `"auto"` (default, auto-detected) or an explicit name such as `"UTF-8"` or `"latin1"`. Pass one explicitly when auto-detection misfires on pure-ASCII or low-signal data. Ignored by the `arrow` backend.

Details

Two backends share this entry point and produce identical findings: the `csv` backend streams a delimited file with `fread` (fread-native `UTF-8`/`latin1` encoding only; column types are locked from the first chunk so later chunks cannot drift; each record must lie on a single line, so embedded newlines in quoted fields are unsupported – read such files whole with `dcc_read`), and the `arrow` backend streams a Parquet/Feather file as record batches (requires the `arrow` package; types come from the file schema and the columnar format is always `UTF-8`, so the encoding restriction does not apply). With `backend = "auto"` the backend is chosen from the file extension: `arrow` for `.parquet`/`.feather`, `csv` for `.csv`/`.tsv`/`.txt`.

Value

A dcc_findings table with n_rows (total rows scanned), n_chunks, and backend attributes.

See Also

dcc_detect for in-memory detection.

Examples

```
csv <- tempfile(fileext = ".csv")
writeLines(c("sid,score", "S1,90", "S2,150", "S3,70"), csv)
rules_file <- tempfile(fileext = ".yaml")
writeLines(c(
  "checks:",
  "  - id: R001",
  "    type: range",
  "    variable: score",
  "    min: 0",
  "    max: 100"
), rules_file)
if (requireNamespace("yaml", quietly = TRUE)) {
  # stream the file two rows at a time; encoding set explicitly since
  # short ASCII files defeat charset auto-detection
  dcc_detect_chunked(csv, dcc_rules(rules_file), chunk_size = 2L,
                     id_var = "sid", encoding = "UTF-8")
}
```

---

dcc_detect_encoding *Detect the character encoding of a text file*

Description

Reads up to n_bytes from the file and detects the most likely character encoding via stringi::stri_enc_detect(). Detected encodings are normalized to the canonical names DCC supports as first-class: "UTF-8", "GB18030" (covers GBK/GB2312), "BIG5", and "latin1" (covers ISO-8859-1/windows-1252).

Usage

```
dcc_detect_encoding(path, n_bytes = 65536L)
```

Arguments

path	Path to the file.
n_bytes	Maximum number of bytes to sample (default 65536).

Value

A list with elements encoding (normalized name), confidence (0-1), and candidates (data.frame of raw detector output).

Examples

```
f <- tempfile(fileext = ".csv")
writeLines("id,name", f)
dcc_detect_encoding(f)$encoding
```

`dcc_dictionary` *Canonical variable dictionary*

Description

Returns the declared source name, canonical name, type, role, and any adapter metadata retained for each imported variable.

Usage

```
dcc_dictionary(x)
```

Arguments

`x`	A `dcc_data` object.

Value

A copy of the canonical variable dictionary.

`dcc_dispositions` *Terminal dispositions of a cleaning result*

Description

Returns one terminal disposition for every finding. Dispositions are the execution source of truth and are reconciled against the audit log.

Usage

```
dcc_dispositions(x)
```

Arguments

`x`	A `dcc_result` from `dcc_execute`.

Value

A `data.table` with `finding_id`, `action`, `status`, and `message`. Status is one of `"changed"`, `"excluded"`, `"flagged"`, `"skipped"`, `"failed"`, or `"unhandled"`.

dcc_doctor *Run every validator over a dataset and rule set*

Description

Runs the requested rule, data, and registered-format backend checks and merges their structured reports without changing data or files.

Usage

```
dcc_doctor(data = NULL, rules = NULL, id_var = NULL, formats = NULL)
```

Arguments

data	Optional dcc_data or data.frame.
rules	Optional dcc_ruleset from dcc_rules.
id_var	Optional record-id column.
formats	NULL, "all", or registered format names whose installed backends and platform limitations should be checked.

Value

A combined dcc_validation object.

Examples

```
rf <- tempfile(fileext = ".yaml")
writeLines(c("checks:", "  - id: R001", "    type: range",
             "    variable: score", "    min: 0", "    max: 100"), rf)
df <- data.frame(sid = c("S1", "S2"), score = c(50, 70))
if (requireNamespace("yaml", quietly = TRUE)) {
  dcc_doctor(df, dcc_rules(rf), id_var = "sid")
}
```

dcc_execute *Execute actions on detected findings (Execute stage)*

Description

Applies declarative actions to detected findings under the closed-loop rule: only cells and records named in the findings list are ever touched, and every change is logged at cell level (old value, new value, triggering check, method, timestamps, versions) carrying the exact finding_id that produced it. The whole plan is validated before any data changes – unknown action IDs, unmapped recodes, missing or duplicated record ids, and group-level cell actions are errors. Findings without a mapped action are returned unhandled rather than silently flagged or dropped. A cell action made inapplicable by an earlier record exclusion is recorded as skipped. The input is never modified.

Usage

```
dcc_execute(x, findings, actions = list(), id_var = NULL,
  default = "flag", ruleset_hash = NULL)
```

Arguments

x	A dcc_data object or data.frame.
findings	A dcc_findings table from the Detect stage with unique, non-empty finding_id values.
actions	Named list mapping check_ids to actions: "exclude", "set_na", "flag", or list(action = "recode", map = c(old = new)). Every name must match a check_id in findings; unknown action IDs are an error.
id_var	Name of the record-id column matching the findings' record_id, or NULL for row numbers. When supplied, the column must contain non-missing, unique ids.
default	Deprecated and no longer applied: findings without an explicit action are returned unhandled rather than auto-dispositioned. Retained only for call compatibility.
ruleset_hash	Optional rule-file hash stamped into the audit log (taken from the findings' dcc_data attribute when available).

Value

A dcc_result: list with data (the new dcc_data version), audit (cell-level audit log whose first column is finding_id), unhandled (findings with no explicit action), dispositions (one terminal row per finding), report_profile (aggregate pre-cleaning types, missingness, and complete frequency counts without raw rows), and n_excluded. Accessors: dcc_audit_log, dcc_cleaned, dcc_dispositions.

Examples

```
df <- data.frame(sid = c("S1", "S2"), score = c(50, 150))
f <- dcc_findings("S2", variable = "score", check_id = "R001",
                  evidence = "out of range", severity = "fail")
res <- dcc_execute(df, f, actions = list(R001 = "set_na"),
                   id_var = "sid")
dcc_audit_log(res)
```

---

dcc_export_log *Export an audit log for external auditors*

Description

Writes the cell-level audit log to disk. Parquet is the default storage format (design decision 3 in docs/design.md); CSV export exists so external auditors can open the log without special tooling.

Usage

```
dcc_export_log(x, path, format = c("parquet", "csv"))
```

Arguments

x	A dcc_result from dcc_execute.
path	Output file path.
format	"parquet" (default; schema-typed and compact, requires the **arrow** package) or "csv" (plain file for auditors without Parquet tooling).

Value

path, invisibly.

Examples

```
df <- data.frame(sid = "S1", q1 = 9)
f <- dcc_findings("S1", variable = "q1", check_id = "C",
                  evidence = "legacy code")
res <- dcc_execute(df, f, actions = list(C = "set_na"), id_var = "sid")
csv <- tempfile(fileext = ".csv")
dcc_export_log(res, csv, format = "csv")
```

---

dcc_findings *The dcc_findings table*

Description

A dcc_findings object is the structured violation list produced by the Detect stage and consumed by the Execute stage: columns finding_id, record_id, variable, check_id, evidence, severity, dimension, code and detector_id. The additive finding_id is deterministic within a run and includes the rule, record, variable, and zero-based occurrence. It is the single interface between detection and execution in the Detect-Execute-Report workflow.

Usage

```
dcc_findings(record_id = character(), variable = NA_character_,
  check_id = character(), evidence = character(), severity = "warn",
  dimension = NA_character_, run_id = "manual", code = check_id,
  detector_id = check_id)
```

Arguments

record_id	Character vector (or coercible) of record ids.
variable	Character vector of affected variables, or NA.
check_id	Character vector of check identifiers.
evidence	Character vector describing the measured evidence.
severity	One of "info", "warn", "fail" (recycled).
dimension	Quality dimension label (recycled).
run_id	One non-empty run identifier used to make finding IDs stable.
code	Stable machine-readable finding code (recycled). Defaults to check_id for backward compatibility.
detector_id	Stable detector implementation identifier (recycled). Defaults to check_id for direct legacy detector calls.

Value

A dcc_findings object (also a data.table).

Examples

```
dcc_findings(record_id = "S001", check_id = "R001",
             evidence = "value 7 outside range [1, 5]",
             severity = "fail", dimension = "validity")
```

---

dcc_help *Explain a DCC workflow code in Chinese or English*

Description

Returns stable plain-language explanations and suggested fixes for codes that can appear in strict-plan validation and preflight diagnostics.

Usage

```
dcc_help(code = NULL, language = "zh-CN")
```

Arguments

code	Optional stable issue code. NULL returns the complete table.
language	"zh-CN" (default) or "en".

Value

A data.frame with code, explanation, and fix.

dcc_import *Strict canonical import*

Description

Reads a source through its registered format adapter and applies an explicit import specification. Source names, canonical names, types, missing codes, and roles are declared rather than guessed. The source file is never modified.

Usage

```
dcc_import(path, spec)
```

Arguments

path	Path to the source file.
spec	A dcc_import_spec created internally from a strict DCC plan.

Value

A dcc_data object with canonical data, dictionary, missing states, import specification, source metadata, and import provenance.

dcc_item_map *Master item map of a form-mapped dataset*

Description

Returns the resolved form-to-master item map attached by dcc_map_forms. This is the public accessor for the mapping; callers should not read the hidden attribute directly.

Usage

```
dcc_item_map(x)
```

Arguments

x	A dcc_data returned by dcc_map_forms.

Value

The item-map data.frame (master, form, source, is_anchor, ...).

Examples

```
data <- data.frame(sid = c("S1", "S2"), form = c("A", "B"),
                   p1 = c(1, 2), p2 = c(5, 6))
fmap <- data.frame(form = c("A", "A", "B", "B"),
                   source = c("p1", "p2", "p1", "p2"),
                   master = c("M001", "M002", "M003", "M002"),
                   is_anchor = c(FALSE, TRUE, FALSE, TRUE))
mapped <- dcc_map_forms(data, fmap, form_var = "form")
dcc_item_map(mapped)
```

---

dcc_l0_diagnose *Level-0 structural diagnostics*

Description

Runs Eurostat-style level-0 (structural) validation on a freshly read table: dimensions, per-column type and missingness profile, duplicate or empty column names, all-missing columns and rows, and encoding confidence. Findings use the same shape as later detection stages so the read report feeds the same audit pipeline.

Usage

```
dcc_l0_diagnose(data, meta = list())
```

Arguments

`data`	A `data.frame` or `data.table`.
`meta`	Optional metadata list from `dcc_read` (used for encoding-confidence findings).

Value

A `dcc_read_report` object: a list with `n_rows`, `n_cols`, `columns` (per-column profile), and `findings` (L0 findings table with columns `check_id`, `severity`, `variable`, `evidence`).

Examples

```
rep <- dcc_l0_diagnose(data.frame(a = c(1, NA), b = c(NA, NA)))
rep$findings
```

dcc_manifest *Build a reproducibility manifest for a cleaning run*

Description

Captures everything needed to re-execute the read -> detect -> execute pipeline and verify byte-identical results: input file path and hash, rule file path and hash, the executed actions, id/default configuration, and content hashes of the cleaned data and the audit log (timestamps excluded).

Usage

```
dcc_manifest(x, path = NULL)
```

Arguments

x	A dcc_result from dcc_execute, produced from a dcc_read input and dcc_detect findings (so input and rule sources are known).
path	Optional path to write the manifest as YAML (requires the **yaml** package).

Value

A dcc_manifest object (named list with input, ruleset, action and output-hash sections), invisibly written to path when given.

See Also

dcc_rerun

Examples

```
csv <- tempfile(fileext = ".csv")
writeLines(c("sid,score", "S1,90", "S2,150"), csv)
rules_file <- tempfile(fileext = ".yaml")
writeLines(c(
  "checks:",
  "  - id: R001",
  "    type: range",
  "    variable: score",
  "    min: 0",
  "    max: 100"
), rules_file)
if (requireNamespace("yaml", quietly = TRUE)) {
  x <- dcc_read(csv)
  f <- dcc_detect(x, dcc_rules(rules_file), id_var = "sid")
  res <- dcc_execute(x, f, actions = list(R001 = "set_na"),
                     id_var = "sid")
  dcc_manifest(res)
}
```

dcc_mapping_findings *Mapping problems found while aligning forms*

Description

Returns the mapping-problem findings attached by dcc_map_forms. This is the public accessor; callers should not read the hidden attribute directly.

Usage

```
dcc_mapping_findings(x)
```

Arguments

x	A dcc_data returned by dcc_map_forms.

Value

A dcc_findings table (possibly empty) of mapping problems (unknown forms, missing source items).

Examples

```
data <- data.frame(sid = c("S1", "S2"), form = c("A", "B"),
                   p1 = c(1, 2), p2 = c(5, 6))
fmap <- data.frame(form = c("A", "A", "B", "B"),
                   source = c("p1", "p2", "p1", "p2"),
                   master = c("M001", "M002", "M003", "M002"),
                   is_anchor = c(FALSE, TRUE, FALSE, TRUE))
mapped <- dcc_map_forms(data, fmap, form_var = "form")
dcc_mapping_findings(mapped)
```

dcc_map_forms *Map multi-form responses onto the master item bank*

Description

Aligns item columns from multiple test forms onto the master item bank using an external form-item mapping table. Items not administered on a respondent's form are structural NA – the concurrent-calibration layout consumed by the downstream IRTC engine. Anchor flags are carried through for fixed-anchor equating. Mapping problems are findings (MAP_SOURCE_MISSING, MAP_UNKNOWN_FORM), not silent drops.

Usage

```
dcc_map_forms(x, form_item_map, form_var)
```

Arguments

x	A dcc_data object or data.frame of responses.
form_item_map	A data.frame with columns form, source (item column on that form), master (master item bank id) and optionally is_anchor (logical, default FALSE) – or a path to a CSV file with those columns (hash recorded in provenance).
form_var	Name of the column in x holding each respondent's form id.

Value

A new dcc_data version: non-item columns pass through, consumed source columns are dropped, one column per master item is appended (NA where not administered). The normalized item map (with is_anchor) is attached as the dcc_item_map attribute, mapping findings as the dcc_findings attribute, and a map_forms provenance record is appended.

Examples

```
df <- data.frame(sid = c("S1", "S2"), form = c("A", "B"),
                 p1 = c("X", "Y"))
map <- data.frame(form = c("A", "B"), source = c("p1", "p1"),
                  master = c("M001", "M002"))
as.data.frame(dcc_map_forms(df, map, form_var = "form"))
```

---

dcc_missing_states *Canonical cell-level missing states*

Description

Returns explicit missing semantics for imported and cleaned cells, including not administered, respondent omission, import missing, declared missing code, and cleared by cleaning.

Usage

```
dcc_missing_states(x)
```

Arguments

x	A dcc_data object.

Value

A copy of the cell-level missing-state table.

dcc_provenance *Provenance chain of a dcc_data object*

Description

Returns the append-only provenance chain recording every stage the object has passed through (read, detect, execute, ...). This chain is the backbone of DCC's auditable-reporting guarantee: any dataset version can be traced back to its source file and rule versions. Legacy records containing only `timestamp` remain readable.

Usage

```
dcc_provenance(x)
```

Arguments

x	A `dcc_data` object.

Value

A `data.table` with one row per provenance record: `stage`, `started_at`, `ended_at`, `outcome`, `dcc_version`, and list-columns `hashes`, `counts`, and `details`.

Examples

```
x <- dcc_data(data.frame(id = 1:2))
dcc_provenance(x)
```

dcc_read *Read a data file into a dcc_data object*

Description

Compatibility entry point over DCC's registered format adapters. New strict workflows use `dcc_import` with a declared import specification; this function retains automatic text-encoding detection and type inference for existing calls. The raw file is never modified.

Usage

```
dcc_read(path, format = "auto", encoding = "auto", ...)
```

Arguments

path	Path to the input file.
format	"auto", a registered format name, or the legacy "excel" alias for XLS/XLSX. Ambiguous extensions require an explicit format.
encoding	"auto" (default) or an explicit source encoding for text formats (e.g. "UTF-8", "GB18030", "BIG5", "latin1"). Ignored for binary formats.
...	Compatibility reader options. Strict protected options remain rejected by the adapter.

Value

A dcc_data object with meta, a read report, and a provenance chain whose first record is the read operation.

Examples

```
f <- tempfile(fileext = ".csv")
writeLines(c("id,score", "S1,90", "S2,85"), f)
x <- dcc_read(f)
dcc_read_report(x)
```

---

dcc_read_config *Read an Excel cleaning-plan configuration*

Description

Converts an Excel cleaning-plan workbook into a dcc_config, so survey staff specify the record ID, item columns, rules, and dispositions in a spreadsheet rather than writing YAML.

Usage

```
dcc_read_config(path)
```

Arguments

path	Path to the .xlsx cleaning-plan workbook.

Details

The workbook has two sheets. settings has key/value rows (id_var, optional items as a comma-separated list). rules has one row per check with columns id, type, variable, min, max, values, items, max_prop, max_run, time_var, min_seconds, traps, severity, action, and recode_map. Write a starter workbook with dcc_write_config_template.

Value

A dcc_config.

See Also

dcc_write_config_template, dcc_config, dcc_run.

Examples

```
if (requireNamespace("writexl", quietly = TRUE) &&
    requireNamespace("readxl", quietly = TRUE)) {
  path <- tempfile(fileext = ".xlsx")
  dcc_write_config_template(path)
  dcc_read_config(path)
}
```

---

dcc_read_plan *Read a strict DCC Excel or JSON plan*

Description

Reads only the exact version-1.0 contract. Unknown, missing, reordered, or renamed workbook sheets and columns are rejected instead of guessed.

Usage

```
dcc_read_plan(path)
```

Arguments

path	Existing .xlsx or .json plan path.

Value

A dcc_plan. Excel plans retain cell-location metadata used by dcc_validate_plan(); JSON validation uses JSON Pointers.

---

dcc_read_report *Read report of a dcc_data object*

Description

Accessor for the level-0 read report produced by dcc_read: dimensions, per-column profile, encoding information, and structural findings.

Usage

```
dcc_read_report(x)
```

Arguments

x	A dcc_data object.

Value

The dcc_read_report attached at read time (see dcc_l0_diagnose), or NULL if the object was not created by dcc_read.

Examples

```
f <- tempfile(fileext = ".csv")
writeLines(c("id,score", "S1,90"), f)
dcc_read_report(dcc_read(f))
```

---

dcc_reconcile *Reconcile findings against logged changes (closed loop)*

Description

Verifies DCC's closed-loop guarantee on the exact finding identity: every audit-log row must join back to a finding by finding_id, and every finding is assigned exactly one terminal status. There is no loose record_id + check_id matching, so a change can never be attributed to the wrong finding and an unhandled finding can never be reported as handled. An audit row whose finding_id is absent from the findings table, or a handled disposition without matching audit evidence, raises a dcc_reconcile_error.

Usage

```
dcc_reconcile(x)
```

Arguments

x	A dcc_result from dcc_execute.

Value

The findings table extended with the recorded action, status, message, and handled. Status is one of "changed", "excluded", "flagged", "skipped", "failed", or "unhandled".

Examples

```
df <- data.frame(sid = "S1", q1 = 9)
f <- dcc_findings("S1", variable = "q1", check_id = "C",
                  evidence = "legacy code")
res <- dcc_execute(df, f, actions = list(C = "set_na"), id_var = "sid")
dcc_reconcile(res)
```

dcc_report *Generate a cleaning report (Report stage)*

Description

Renders a self-contained HTML report in two audiences: a management summary (findings by quality dimension and severity, change volumes, exclusions, provenance and hashes) and an audit report that adds the reconciliation table and the cell-level change log. HTML is generated directly, without a pandoc/rmarkdown dependency.

Usage

```
dcc_report(x, path = NULL, audience = c("summary", "audit"),
  max_rows = 1000L)
```

Arguments

x	A dcc_result from dcc_execute.
path	Output .html file path, or NULL to only return the HTML string.
audience	"summary" (management layer, default) or "audit" (adds the full findings table with evidence, the findings-to-changes reconciliation, and the cell-level change log).
max_rows	Maximum audit-log rows embedded in the HTML (default 1000); the complete log is exported with dcc_export_log.

Value

The HTML document as a character string, invisibly. Written to path when given.

Examples

```
df <- data.frame(sid = c("S1", "S2"), score = c(50, 150))
f <- dcc_findings("S2", variable = "score", check_id = "R001",
                  evidence = "out of range", severity = "fail",
                  dimension = "validity")
res <- dcc_execute(df, f, actions = list(R001 = "set_na"),
                   id_var = "sid")
html <- dcc_report(res, audience = "audit")
substr(html, 1, 60)
```

dcc_report_machine *Render the machine report bundle*

Description

Writes deterministic JSON and JSONL artifacts, an SHA-256 manifest, and the versioned schemas an AI agent or external system needs to validate them.

Usage

```
dcc_report_machine(model, output_dir)
```

Arguments

model	A validated dcc_report_model.
output_dir	Existing or new directory for machine artifacts.

Value

Paths to the eight machine files and the schemas directory.

dcc_report_model *Build and validate the normalized report model*

Description

Creates the single normalized source of facts consumed by staff, statistical, and machine report renderers. The constructor copies source objects and verifies counts, finding identities, hashes, and timings.

Usage

```
dcc_report_model(result, run = NULL)

dcc_validate_report_model(x)
```

Arguments

result	A dcc_result returned by dcc_execute().
run	An optional dcc_run containing run and plan metadata.
x	A report-model-like named list to validate.

Value

dcc_report_model() returns a versioned dcc_report_model list. dcc_validate_report_model() returns a structured dcc_validation table with stable error codes.

dcc_report_staff *Render the bilingual staff report*

Description

Produces the staff workbook, dependency-free HTML report, and concise text summary from one normalized model. Sensitive examples are masked by default, and Excel row limits are checked before writing so data are never truncated.

Usage

```
dcc_report_staff(
  model,
  output_dir,
  formats = c("xlsx", "html"),
  language = c("zh-CN", "en"),
  include_examples = FALSE
)
```

Arguments

model	A validated dcc_report_model.
output_dir	Existing or new directory for the report files.
formats	Any combination of "xlsx" and "html".
language	Primary display language, "zh-CN" or "en".
include_examples	Whether raw examples may be disclosed. Defaults to FALSE.

Value

Character paths of files written, including run-summary.txt.

dcc_report_statistical *Render the statistical report bundle*

Description

Writes complete findings, audit, reconciliation, before/after profiles, scoring, mapping, provenance, parameters, and an optional methods narrative. The artifact manifest records SHA-256 hashes after every artifact is closed.

Usage

```
dcc_report_statistical(
  model,
  output_dir,
  table_format = c("parquet", "csv"),
  html = TRUE
)
```

Arguments

model	A validated dcc_report_model.
output_dir	Existing or new directory for report files.
table_format	Complete tables as "parquet" or "csv".
html	Whether to write the statistical HTML narrative.

Value

Character paths of every file written.

---

dcc_rerun *Re-run a cleaning pipeline from its manifest and verify the output*

Description

Implements one-command reproducibility (design principle 7): reads the raw input again and verifies its hash, reloads and verifies the rule file, re-runs detect and execute with the recorded actions, and compares the cleaned data and audit log (excluding timestamps) against the manifest hashes. An input or rule file whose hash no longer matches raises a typed error – changed inputs make reproduction claims meaningless.

Usage

```
dcc_rerun(manifest)
```

Arguments

manifest	A dcc_manifest, a dcc_result (a manifest is built from it first), or a path to a manifest YAML file.

Value

A dcc_rerun object: list with reproduced (logical), data_match, audit_match, and the re-run result.

See Also

dcc_manifest

Examples

```
csv <- tempfile(fileext = ".csv")
writeLines(c("sid,score", "S1,90", "S2,150"), csv)
rules_file <- tempfile(fileext = ".yaml")
writeLines(c(
  "checks:",
  "  - id: R001",
  "    type: range",
  "    variable: score",
  "    min: 0",
  "    max: 100"
), rules_file)
if (requireNamespace("yaml", quietly = TRUE)) {
  x <- dcc_read(csv)
  f <- dcc_detect(x, dcc_rules(rules_file), id_var = "sid")
  res <- dcc_execute(x, f, actions = list(R001 = "set_na"),
                     id_var = "sid")
  # re-run from the manifest and confirm byte-identical reproduction
  dcc_rerun(dcc_manifest(res))$reproduced
}
```

---

dcc_result_summary *Create a structured AI summary of a DCC result*

Description

Returns bounded, deterministic fields for AI agents without requiring them to parse console prose or hidden R attributes.

Usage

```
dcc_result_summary(result, detail = c("compact", "full"))
```

Arguments

result	A dcc_result returned by dcc_execute().
detail	"compact" returns at most 20 finding rows without raw evidence; "full" adds reconciliation, provenance, and hashes.

Value

A named structured list containing stable action codes.

dcc_rules *Load a declarative rule set from a YAML file*

Description

Rule files are YAML with embedded R expressions for complex logic. The top-level key `checks` holds a list of rules; each rule has `id`, `type`, an optional `severity` and `dimension`, and type-specific fields: `range` (`variable`, `min`/`max`), `set` (`variable`, `values`), `expr` (an R expression evaluated per record in a restricted environment), `skip_logic` (`when` {`variable`, `equals`} and `then_not_required`, marking skipped items as not administered for the missing-items detector), or a detector type (`missing_items`, `straightlining`, `response_time`, `trap_items`, `score_anomaly`) whose fields are passed to the matching `detect_*()` function.

Usage

```
dcc_rules(path)
```

Arguments

`path`	Path to the YAML rule file.

Value

A `dcc_ruleset` object (list of normalized rules, with the source file and its MD5 hash attached for the audit trail).

Examples

```
f <- tempfile(fileext = ".yaml")
writeLines(c(
  "checks:",
  "  - id: R001",
  "    type: range",
  "    variable: score",
  "    min: 0",
  "    max: 100"
), f)
if (requireNamespace("yaml", quietly = TRUE)) {
  dcc_rules(f)
}
```

dcc_run *Run a cleaning workflow with one command*

Description

The survey-staff entry point: orchestrates the Detect -> Execute -> Report pipeline from a dcc_config or strict plan and writes a fixed output layout. Preview is the default, so the safe path requires no extra care, and the raw input file is never modified in any mode. Files are written to a same-parent staging directory and published atomically. Existing output directories are never overwritten; failed runs publish a diagnostic .failed-<run_id> directory and raise dcc_run_error. A renderer failure publishes cleaning evidence under .partial-<run_id> and records the failed audience without claiming full success.

Usage

```
dcc_run(data, config = NULL, output_dir,
  mode = c("preview", "execute", "verify", "rerun"), id_var = NULL,
  plan = NULL)
```

Arguments

data	A data file path, a dcc_data, or a data.frame (or a manifest path in "rerun" mode).
config	Optional dcc_config from dcc_config. Supply this or plan, not both.
output_dir	Directory for the fixed output layout (created if needed). Required and never defaulted: the caller chooses every location DCC writes to.
mode	One of "preview" (default), "execute", "verify", "rerun".
id_var	Record-id column; defaults to the config's id_var.
plan	Optional strict .xlsx/.json plan path or dcc_plan.

Details

Modes: "preview" detects and reports only (no data change, no cleaned-data.csv); "execute" applies the configured actions and writes the cleaned data, audit log, and manifest; "verify" is like execute with a reconciliation summary; "rerun" reproduces a previous run from its manifest.yaml; pass the manifest path as data and use a new output_dir.

Output layout under output_dir: cleaned-data.csv (execute/verify), findings.xlsx (or findings.csv without the writexl package), audit-log.csv (execute/verify), management-report.html, audit-report.html, manifest.yaml (execute/verify), run-summary.txt, run-manifest.json, and selected staff/, statistical/, and machine/ audience directories.

Value

A dcc_run object with mode, terminal status, config, written paths (via dcc_run_files), normalized report model, and structured run manifest.

See Also

dcc_config, dcc_run_files, dcc_doctor.

Examples

```
rf <- tempfile(fileext = ".yaml")
writeLines(c("checks:", "  - id: R001", "    type: range",
             "    variable: score", "    min: 0", "    max: 100"), rf)
csv <- tempfile(fileext = ".csv")
writeLines(c("sid,score", "S1,90", "S2,150"), csv)
if (requireNamespace("yaml", quietly = TRUE)) {
  cfg <- dcc_config(dcc_rules(rf), actions = list(R001 = "set_na"),
                    id_var = "sid")
  run <- dcc_run(csv, cfg, tempfile("dcc-out"), mode = "preview")
  dcc_run_files(run)
}
```

---

dcc_run_files *Output files written by a run*

Description

Returns the output paths written by dcc_run.

Usage

```
dcc_run_files(x)
```

Arguments

x	A dcc_run from dcc_run.

Value

A character vector of the file paths the run wrote.

Examples

```
rf <- tempfile(fileext = ".yaml")
writeLines(c("checks:", "  - id: R001", "    type: range",
             "    variable: score", "    min: 0", "    max: 100"), rf)
csv <- tempfile(fileext = ".csv")
writeLines(c("sid,score", "S1,90", "S2,150"), csv)
if (requireNamespace("yaml", quietly = TRUE)) {
  cfg <- dcc_config(dcc_rules(rf), id_var = "sid")
  dcc_run_files(dcc_run(csv, cfg, tempfile("dcc-out")))
}
```

dcc_schema *Published JSON Schema for a DCC object*

Description

Returns the formal JSON Schema (draft-07) for one of DCC's public objects. The schemas are versioned artifacts installed with the package under `inst/schemas/`, so AI systems and external validators can check a strict plan, normalized report model, rule file, action map, findings table, disposition, provenance, audit log, or manifest against a stable contract.

Usage

```
dcc_schema(name, as = c("object", "path"))
```

Arguments

`name`	One of `"finding"`, `"disposition"`, `"provenance"`, `"audit_log"`, `"rules"`, `"actions"`, `"manifest"`, `"plan"`, or `"report-model"`, plus the machine-report schemas `"run"`, `"validation"`, `"summary"`, `"audit_record"`, `"reconciliation"`, `"machine_provenance"`, and `"artifact_manifest"`.
`as`	`"object"` (default) returns the parsed schema (requires the `jsonlite` package); `"path"` returns the installed file path.

Value

The parsed schema (a list) or the schema file path.

See Also

`dcc_capabilities` for the capability document.

Examples

```
dcc_schema("finding", as = "path")
if (requireNamespace("jsonlite", quietly = TRUE)) {
  dcc_schema("actions")$title
}
```

dcc_score *Score responses against an answer key*

Description

Scores responses using an external, versioned answer key. Single-choice items get full points on exact match. Multiple-select items (responses like `"AC"` or `"A,C"`) are all-or-nothing by default; with `partial = TRUE` the score is `points * max(0, (hits - false_alarms) / n_key)`. When `omit_policy = "na"`, a row with no observed item scores has an `NA` total rather than zero.

Usage

```
dcc_score(x, answer_key, omit_policy = c("zero", "na"),
  scoring_fn = NULL)
```

Arguments

`x`	A `dcc_data` object or data.frame of responses.
`answer_key`	A data.frame with columns `item`, `key`, and optionally `type` (`"single"`/`"multiple"`, default `"single"`), `points` (default 1), `partial` (default `FALSE`) – or a path to a CSV file with those columns (the file's MD5 hash is recorded in the provenance chain).
`omit_policy`	`"zero"` (omitted item scores 0, default) or `"na"` (omitted stays `NA`; use after `dcc_map_forms` so not-administered items are not scored as wrong).
`scoring_fn`	Optional `function(responses, key_row)` returning a numeric vector with exactly one value per response; replaces built-in scoring for every item. This is the extension point reserved for weighted/rubric/polytomous scoring (design decision 2).

Value

A new `dcc_data` version with one `<item>_score` column per keyed item and a `total_score` column appended, plus a `score` provenance record (key source, key hash, omit policy).

Examples

```
df <- data.frame(sid = c("S1", "S2"),
                 it1 = c("A", "B"), it2 = c("AC", "A"))
key <- data.frame(item = c("it1", "it2"), key = c("A", "AC"),
                  type = c("single", "multiple"))
as.data.frame(dcc_score(df, key))
```

dcc_template *Create the strict bilingual DCC Excel template*

Description

Writes a protected version-1.0 workbook for survey staff. Machine headers and workbook structure are locked; yellow input cells remain editable. Sheet protection has no password and prevents accidental edits only.

Usage

```
dcc_template(path, language = "zh-CN")
```

Arguments

path	Destination `.xlsx` path. Required and never defaulted: the caller chooses every location DCC writes to. Existing files are never overwritten.
language	Primary instruction language, `"zh-CN"` or `"en"`; both languages remain visible in the workbook.

Value

The normalized destination path, invisibly.

Examples

```
path <- tempfile(fileext = ".xlsx")
dcc_template(path)
file.exists(path)
```

dcc_trace *Trace the cleaning history of a record or cell*

Description

Reverse lookup from the cleaned data back through the pipeline: all findings and all logged changes for one record, optionally narrowed to one cell. This implements the auditable-reporting requirement that any cell in the final dataset can be traced to its cleaning history.

Usage

```
dcc_trace(x, record_id, variable = NULL)
```

Arguments

x	A dcc_result from dcc_execute.
record_id	Record identifier (as used in the findings).
variable	Optional variable name to narrow the trace to a single cell.

Value

A dcc_trace object: list with record_id, variable, findings (matching findings rows) and changes (matching audit rows).

Examples

```
df <- data.frame(sid = "S1", q1 = 9)
f <- dcc_findings("S1", variable = "q1", check_id = "C",
                  evidence = "legacy code")
res <- dcc_execute(df, f, actions = list(C = "set_na"), id_var = "sid")
dcc_trace(res, "S1", "q1")
```

---

dcc_unhandled *Findings left unhandled by execution*

Description

Returns the findings that had no explicit action in dcc_execute. This is the public accessor for the result's unhandled set; callers should not read the underlying list element directly.

Usage

```
dcc_unhandled(x)
```

Arguments

x	A dcc_result from dcc_execute.

Value

A dcc_findings table (possibly empty) of findings that had no explicit action and were therefore neither changed nor dispositioned.

Examples

```
df <- data.frame(sid = "S1", q1 = 9)
f <- dcc_findings("S1", variable = "q1", check_id = "C", evidence = "e")
res <- dcc_execute(df, f, actions = list(), id_var = "sid")
dcc_unhandled(res)
```

dcc_validate_config *Validate a cleaning configuration*

Description

Runs dcc_validate_rules over the config's rules and additionally checks that every action targets a check_id the rules can produce.

Usage

```
dcc_validate_config(config)
```

Arguments

config	A dcc_config from dcc_config.

Value

A dcc_validation object.

Examples

```
rf <- tempfile(fileext = ".yaml")
writeLines(c("checks:", "  - id: R001", "    type: range",
             "    variable: score", "    min: 0", "    max: 100"), rf)
if (requireNamespace("yaml", quietly = TRUE)) {
  cfg <- dcc_config(dcc_rules(rf), actions = list(R001 = "set_na"))
  dcc_validate_config(cfg)
}
```

dcc_validate_data *Validate data against a rule set before detection*

Description

Checks that a dataset can carry a cleaning run: the record-id column is present, non-missing, and unique, and every variable a rule references exists. It never changes the data.

Usage

```
dcc_validate_data(data, rules = NULL, id_var = NULL)
```

Arguments

data	A dcc_data or data.frame.
rules	Optional dcc_ruleset; when supplied, referenced variables are checked against the data columns.
id_var	Optional record-id column to check for presence, missingness, and duplication.

Value

A dcc_validation object (see dcc_validate_rules).

See Also

dcc_validate_rules, dcc_doctor.

Examples

```
df <- data.frame(sid = c("S1", "S1", "S2"), score = c(50, 150, 70))
dcc_validate_data(df, id_var = "sid")
```

---

dcc_validate_json *Validate DCC JSON and JSON Lines artifacts*

Description

Uses DCC's built-in structural validator, avoiding an additional runtime dependency. Published schema files remain compatible with full external JSON Schema validators.

Usage

```
dcc_validate_json(path, schema)

dcc_validate_jsonl(path, schema)
```

Arguments

path	Existing JSON or JSON Lines file.
schema	A schema name accepted by dcc_schema().

Value

TRUE when the artifact matches the published contract.

dcc_validate_plan *Validate a strict DCC project plan*

Description

Validates the common versioned contract used by strict Excel workbooks and JSON plans. Validation is read-only and returns stable issue codes suitable for staff help pages and AI agents.

Usage

```
dcc_validate_plan(x)
```

Arguments

x	A dcc_plan, normally returned by dcc_read_plan().

Value

A dcc_validation table. Blocking rows have severity "fail".

dcc_validate_rules *Validate a rule set before it is used*

Description

Checks a dcc_rules rule set for structural problems – unknown types, missing required fields, duplicate or empty IDs – and returns a structured report. It never evaluates rules against data and never changes anything.

Usage

```
dcc_validate_rules(rules)
```

Arguments

rules	A dcc_ruleset from dcc_rules (or the rules element of a dcc_config).

Value

A dcc_validation object: a data.table with code, severity, field, rows (affected row indices, a list column), fix, and optional workbook location fields workbook, sheet, row, column, and cell.

See Also

dcc_validate_data, dcc_doctor.

Examples

```
rf <- tempfile(fileext = ".yaml")
writeLines(c("checks:", "  - id: R001", "    type: range",
             "    min: 0", "    max: 100"), rf)
if (requireNamespace("yaml", quietly = TRUE)) {
  dcc_validate_rules(dcc_rules(rf))
}
```

---

dcc_validation_errors *The failing issues of a validation report*

Description

Filters a validation report to only the blocking (`"fail"`) issues.

Usage

```
dcc_validation_errors(x)
```

Arguments

`x`	A `dcc_validation` object.

Value

The subset of `x` with `severity == "fail"`.

Examples

```
df <- data.frame(sid = c("S1", "S1"), score = c(50, 70))
dcc_validation_errors(dcc_validate_data(df, id_var = "sid"))
```

---

dcc_write_config_template *Write a starter Excel cleaning-plan template*

Description

Writes an example two-sheet cleaning-plan workbook that `dcc_read_config` can read, for survey staff to edit.

Usage

```
dcc_write_config_template(path)
```

Arguments

path	Output .xlsx path.

Value

path, invisibly.

See Also

dcc_read_config.

Examples

```
if (requireNamespace("writexl", quietly = TRUE)) {
  dcc_write_config_template(tempfile(fileext = ".xlsx"))
}
```

---

detect_missing_items *Detect excessive item nonresponse per respondent*

Description

Flags respondents whose proportion of missing item responses exceeds max_prop. Like all detectors, it only finds; exclusion happens in the Execute stage.

Usage

```
detect_missing_items(x, items, max_prop = 0.5, id_var = NULL,
  severity = "warn", structural = NULL)
```

Arguments

x	A dcc_data object or data.frame.
items	Character vector of item column names.
max_prop	Maximum tolerated missing proportion (default 0.5).
id_var	Name of the record-id column, or NULL for row numbers.
severity	Severity assigned to findings (default "warn").
structural	Optional logical matrix (rows aligned to the data, columns to items) marking cells that were *not administered* – e.g. from a skip_logic rule. Not-administered cells are excluded from both the numerator and denominator of the missing proportion. NULL (default) treats every item as administered and is byte-identical to the pre-1.1.0 behaviour.

Value

A dcc_findings table (check id Q_MISSING_ITEMS, dimension completeness).

Examples

```
df <- data.frame(sid = c("S1", "S2"),
                 q1 = c(1, NA), q2 = c(2, NA), q3 = c(3, 1))
detect_missing_items(df, c("q1", "q2", "q3"), max_prop = 0.5,
                     id_var = "sid")
```

`detect_response_time` *Detect implausibly fast or anomalous response times*

Description

Flags respondents whose total response time is below an absolute minimum, or below a fraction of the median time. Each finding's evidence states which cut was triggered.

Usage

```
detect_response_time(x, time_var, min_seconds = NULL,
  min_median_ratio = 1/3, id_var = NULL, severity = "warn")
```

Arguments

`x`	A `dcc_data` object or data.frame.
`time_var`	Name of the total response-time column (numeric, seconds).
`min_seconds`	Absolute minimum plausible total time, or `NULL` to skip the absolute check.
`min_median_ratio`	Flag times below this fraction of the median (default 1/3), or `NULL` to skip the relative check.
`id_var`	Name of the record-id column, or `NULL` for row numbers.
`severity`	Severity assigned to findings (default `"warn"`).

Value

A `dcc_findings` table (check id `Q_RESPONSE_TIME`).

Examples

```
df <- data.frame(sid = c("S1", "S2", "S3"),
                 time_total = c(600, 45, 590))
detect_response_time(df, "time_total", min_seconds = 60, id_var = "sid")
```

`detect_score_anomaly` *Detect group-wise score anomalies*

Description

Flags respondents whose score is an outlier within their group (IQR fences or z-scores), and groups whose mean deviates strongly from the overall mean.

Usage

```
detect_score_anomaly(x, score_var, group_vars = NULL,
  method = c("iqr", "zscore"), k = 1.5, group_mean_z = 2,
  id_var = NULL, severity = "warn")
```

Arguments

`x`	A `dcc_data` object or data.frame.
`score_var`	Name of the numeric score column.
`group_vars`	Character vector of grouping columns; `NULL` treats the data as one group.
`method`	`"iqr"` (default) or `"zscore"` for within-group outliers.
`k`	Fence multiplier: IQR multiplier (default 1.5) or |z| cutoff (use e.g. 3 with `method = "zscore"`).
`group_mean_z`	Flag groups whose mean is more than this many overall standard deviations from the overall mean (default 2; `NULL` skips the group-level check).
`id_var`	Name of the record-id column, or `NULL` for row numbers.
`severity`	Severity assigned to findings (default `"warn"`).

Value

A `dcc_findings` table (check ids `Q_SCORE_OUTLIER` and `Q_GROUP_SCORE_SHIFT`; group-level findings have `record_id = NA` and the group label in `evidence`).

Examples

```
df <- data.frame(sid = sprintf("S%d", 1:8),
                 grp = rep(c("A", "B"), each = 4),
                 score = c(80, 82, 15, 81, 60, 62, 61, 59))
detect_score_anomaly(df, "score", group_vars = "grp", id_var = "sid")
```

detect_straightlining *Detect straight-lining (longstring)*

Description

Computes the longest run of identical consecutive item responses per respondent (the longstring index; cf. the CRAN package **careless**) and flags respondents whose run meets or exceeds `max_run`. The computation is vectorized over respondents.

Usage

```
detect_straightlining(x, items, max_run = 10L, id_var = NULL,
  severity = "warn", na_breaks_run = TRUE)
```

Arguments

`x`	A `dcc_data` object or data.frame.
`items`	Character vector of item column names, in presentation order.
`max_run`	Minimum run length considered straight-lining (default 10).
`id_var`	Name of the record-id column, or `NULL` for row numbers.
`severity`	Severity assigned to findings (default `"warn"`).
`na_breaks_run`	Should a missing response break a run? (default `TRUE`).

Value

A `dcc_findings` table (check id `Q_STRAIGHTLINING`).

Examples

```
df <- data.frame(sid = c("S1", "S2"),
                 q1 = c(1, 4), q2 = c(2, 4), q3 = c(1, 4), q4 = c(3, 4))
detect_straightlining(df, paste0("q", 1:4), max_run = 4, id_var = "sid")
```

detect_trap_items *Detect failed trap (attention-check) items*

Description

Compares responses on designated trap (attention-check) items with their expected values and flags respondents failing at least `max_failed` traps. Trap definitions are typically maintained in an external, versioned key file.

Usage

```
detect_trap_items(x, traps, max_failed = 1L, id_var = NULL,
  severity = "fail", na_fails = TRUE)
```

Arguments

x	A dcc_data object or data.frame.
traps	Named list or named vector: names are trap item columns, values the expected response.
max_failed	Number of failed traps that triggers a finding (default 1).
id_var	Name of the record-id column, or NULL for row numbers.
severity	Severity assigned to findings (default "fail").
na_fails	Should a missing response on a trap item count as a failure? (default TRUE).

Value

A dcc_findings table (check id Q_TRAP_ITEMS).

Examples

```
df <- data.frame(sid = c("S1", "S2"), trap1 = c(3, 5))
detect_trap_items(df, traps = list(trap1 = 3), id_var = "sid")
```

Index