Package 'dqcheckr'

July 26, 2026

Type Package

Title Automated Data Quality Checks for Recurring Dataset Deliveries

Version 0.3.0

Date 2026-07-26

Description Automates quality verification of recurring external dataset deliveries. For each new file arrival, it runs single-snapshot quality checks, compares the file to the previous delivery, writes a self-contained 'HTML' report, and records summary statistics in a local 'SQLite' database for long-term trend tracking. Supports 'CSV' and fixed-width formats. Custom organisation-specific checks can be supplied as plain R files.

License MIT + file LICENSE

URL https://mickmioduszewski.github.io/dqcheckr/, https://github.com/mickmioduszewski/dqcheckr

BugReports https://github.com/mickmioduszewski/dqcheckr/issues

Encoding UTF-8

Language en-GB

Depends R (>= 4.2)

Imports readr, stringi, DBI, RSQLite, quarto, knitr, kableExtra, ggplot2, gridExtra, tidyr, yaml, rlang, stats, tools, utils

Suggests testthat (>= 3.1.0), withr, rmarkdown, pkgdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Mick Mioduszewski [aut, cre]

Maintainer Mick Mioduszewski <mick@mioduszewski.net>

Repository CRAN

Date/Publication 2026-07-26 03:00:08 UTC

Contents

check_allowed_values *QC-09: Check for values outside the allowed set*

Description

For each column that has `allowed_values` configured in `config$column_rules`, returns a `dq_result` flagging any non-empty values not in the allowed list. Returns an empty list when no `allowed_values` rules are configured.

Usage

```
check_allowed_values(df, config)
```

Arguments

`df`	A data frame with all columns as character vectors (as returned by `read_dataset`).
`config`	Named list. Merged configuration as returned by `load_config`.

Value

A list of `dq_result` objects, one per configured column. Status is `"FAIL"` when unexpected values are found; `"PASS"` otherwise. Returns an empty list if no `allowed_values` rules are configured.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_allowed_values(df, cfg)
```

check_col_count *QC-05: Report column count*

Description

Returns a single `"INFO"` `dq_result` recording the number of columns in the data frame. Never fails or warns.

Usage

```
check_col_count(df, config)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config. Currently unused; present for API consistency.

Value

A list containing one dq_result with status "INFO".

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_col_count(df, cfg)
```

---

check_distinct_counts *QC-08: Report distinct value counts for character columns*

Description

For each column whose resolved type is "character", returns one "INFO" dq_result with the count of distinct non-empty values. Columns inferred as numeric or date are silently skipped.

Usage

```
check_distinct_counts(df, config, types = NULL)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.
types	Optional named character vector of pre-resolved column types; see check_inferred_types.

Value

A list of dq_result objects (one per character column), all with status "INFO". Returns an empty list if no character columns are found.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_distinct_counts(df, cfg)
```

check_duplicate_rows *QC-03: Check for fully-duplicate rows*

Description

Returns a single dq_result for the whole table. A row is considered a duplicate when every column value is identical to another row.

Usage

```
check_duplicate_rows(df, config)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config. Currently unused; present for API consistency.

Value

A list containing one dq_result. Status is "WARN" if any duplicate rows exist; "PASS" otherwise.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_duplicate_rows(df, cfg)
```

check_empty_column *QC-02: Check for entirely empty columns*

Description

Returns a dq_result per column. A column is considered empty when every value is NA or the empty string "".

Usage

```
check_empty_column(df, config)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.

Value

A list of dq_result objects, one per column. Status is "FAIL" for entirely empty columns; "PASS" otherwise.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_empty_column(df, cfg)
```

---

check_file_encoding *QC-16: File encoding sanity*

Description

Verifies that the delivered file's bytes matched the encoding declared in the config. read_dataset scans the whole file for UTF-8 validity before parsing (when the effective encoding is UTF-8) and records the outcome on the returned data frame; this check turns that outcome into a result:

- **PASS** when the file was valid UTF-8 as declared, or when a declared single-byte encoding (e.g. ISO-8859-1, Windows-1252) made a validity scan meaningless – every byte sequence is valid in those encodings by construction.
- **FAIL** when the file was not valid UTF-8 as declared. The run still completes: the file is read using a single-byte fallback encoding, and the message reports the detector's best guess at the actual encoding so the config can be corrected.
- **WARN** when the declared encoding is multi-byte or unknown (e.g. UTF-16LE, Shift-JIS): dqcheckr scans only UTF-8, so such a file is read as declared but its validity is not verified – it is never reported as "valid by construction".
- **WARN** when the UTF-8 scan itself could not complete (for example out of memory on a very large delivery): validity is unknown, so it is neither a clean PASS nor a definitive FAIL.

A supplier can change their export encoding between deliveries, which is why this runs against every delivery rather than only at configuration time. Returns an empty list when df did not come from read_dataset (no scan outcome to report).

Usage

```
check_file_encoding(df, config)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config. Present for interface consistency; the scan outcome travels with df.

Value

A list with one dq_result object, or an empty list when no scan outcome is attached to df.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_file_encoding(df, cfg)
```

---

check_inferred_types *QC-06: Report inferred column types*

Description

Returns one "INFO" dq_result per column recording the type resolved by resolve_col_type ("date", "numeric", "character", or "unknown"). Per-column overrides from config$column_types are respected.

Usage

```
check_inferred_types(df, config, types = NULL)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.
types	Optional named character vector of pre-resolved column types (one element per column, as produced by resolve_col_type). When NULL (the default), types are resolved internally. Supplying this avoids re-running type inference when several checks share one data frame.

Value

A list of dq_result objects, one per column, all with status "INFO".

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_inferred_types(df, cfg)
```

check_key_uniqueness *QC-12: Check uniqueness of key column(s)*

Description

Checks that the column(s) listed in `config$key_columns` have no duplicate values. When `key_columns` is a single string, one result is returned for that column. When it is a character vector of length > 1, a single result covering the composite key is returned. Returns an empty list if `key_columns` is not configured.

Usage

```
check_key_uniqueness(df, config)
```

Arguments

`df`	A data frame with all columns as character vectors (as returned by `read_dataset`).
`config`	Named list. Merged configuration as returned by `load_config`.

Value

A list of `dq_result` objects. Status is `"FAIL"` when duplicates or missing key columns are detected; `"PASS"` otherwise. Returns an empty list if `key_columns` is not configured.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_key_uniqueness(df, cfg)
```

check_min_row_count *QC-14: Check row count bounds and optional file size*

Description

Runs up to four sub-checks, each returning a separate `dq_result`:

1. **Empty file** – FAIL when the file contains no data rows at all. Emitted unconditionally (independent of `min_row_count`) so that an empty delivery always fails the run.
2. **File size** – only when `file_path` is supplied and `max_file_size_mb` is configured in `rules`: FAIL if the file exceeds the size limit.
3. **Minimum row count** – FAIL if `row_count` < `min_row_count`. Skipped (PASS with a note) when `min_row_count` is `0`.
4. **Maximum row count** – only when `max_row_count` is configured in `rules`: FAIL if `row_count` > `max_row_count`.

Usage

```
check_min_row_count(df, config, file_path = NULL)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.
file_path	Character or NULL. Absolute path to the file on disk, required for the optional file-size sub-check.

Value

A list of dq_result objects (one to four entries depending on which sub-checks are active).

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_min_row_count(df, cfg, file_path = path)
```

---

check_missing_rate *QC-01: Check missing rate per column*

Description

Returns a dq_result per column flagging columns whose proportion of missing or empty values exceeds max_missing_rate.

Usage

```
check_missing_rate(df, config)
```

Arguments

df	A data frame with all columns as character vectors.
config	Named list as returned by load_config.

Value

A list of dq_result objects, one per column.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_missing_rate(df, cfg)
```

---

check_non_numeric *QC-11: Check non-numeric rate in numeric columns*

Description

For each column whose resolved type is `"numeric"`, computes the proportion of non-empty values that cannot be coerced to numeric. Returns `"FAIL"` when the rate exceeds `max_non_numeric_rate` (default 0.01), `"WARN"` when it exceeds `warn_non_numeric_rate` (default 0), and `"PASS"` otherwise. Both thresholds support per-column overrides via `config$column_rules`.

Usage

```
check_non_numeric(df, config, types = NULL)
```

Arguments

`df`	A data frame with all columns as character vectors (as returned by `read_dataset`).
`config`	Named list. Merged configuration as returned by `load_config`.
`types`	Optional named character vector of pre-resolved column types; see `check_inferred_types`.

Value

A list of `dq_result` objects, one per numeric column. Returns an empty list if no numeric columns are found.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_non_numeric(df, cfg)
```

check_numeric_bounds *QC-10: Check for out-of-range numeric values*

Description

For each column that has `min_value` or `max_value` configured in `config$column_rules`, returns a `dq_result` flagging any values that fall outside the specified range. Returns an empty list when no bound rules are configured.

Usage

```
check_numeric_bounds(df, config)
```

Arguments

`df`	A data frame with all columns as character vectors (as returned by `read_dataset`).
`config`	Named list. Merged configuration as returned by `load_config`.

Value

A list of `dq_result` objects, one per configured column. Status is `"FAIL"` when out-of-range values are found; `"PASS"` otherwise. Returns an empty list if no bound rules are configured.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_numeric_bounds(df, cfg)
```

check_numeric_stats *QC-07: Report numeric summary statistics*

Description

For each column whose resolved type is `"numeric"`, returns one `"INFO"` `dq_result` containing min, max, mean, and standard deviation of the parseable values. Columns inferred as non-numeric are silently skipped.

Usage

```
check_numeric_stats(df, config, types = NULL)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.
types	Optional named character vector of pre-resolved column types; see check_inferred_types.

Details

Non-finite values (Inf, -Inf) are excluded from the aggregates, and the count of excluded values is appended to observed so the anomaly is still visible in the report.

Value

A list of dq_result objects (one per numeric column), all with status "INFO". Returns an empty list if no numeric columns are found.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_numeric_stats(df, cfg)
```

---

check_outliers *QC-15: Detect statistical outliers in numeric columns*

Description

For each column whose resolved type is "numeric", applies up to two outlier detection methods (combined with logical OR):

1. **Z-score**: values whose absolute Z-score exceeds max_z_score are flagged.
2. **IQR fence**: values below Q1 - k * IQR or above Q3 + k * IQR (where k = iqr_fence_multiplier) are flagged.

Both thresholds support per-column overrides via config$column_rules. A column is skipped (PASS with a note) when neither threshold is configured or when it has fewer than four parseable values.

Usage

```
check_outliers(df, config, types = NULL)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.
types	Optional named character vector of pre-resolved column types; see check_inferred_types.

Value

A list of dq_result objects, one per numeric column. Status is "FAIL" when outliers are detected; "PASS" otherwise. Returns an empty list if no numeric columns are found.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_outliers(df, cfg)
```

---

check_pattern *QC-13: Check values against a regex pattern*

Description

For each column that has a pattern configured in config$column_rules, returns a dq_result reporting how many non-empty values do not match the Perl-compatible regular expression. Returns an empty list when no pattern rules are configured.

Usage

```
check_pattern(df, config)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.

Value

A list of dq_result objects, one per configured column. Status is "FAIL" when any values violate the pattern; "PASS" otherwise. Returns an empty list if no pattern rules are configured.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_pattern(df, cfg)
```

check_row_count *QC-04: Report row count*

Description

Returns a single "INFO" dq_result recording the number of rows in the data frame. Never fails or warns; use check_min_row_count for threshold-based row count checks.

Usage

```
check_row_count(df, config)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config. Currently unused; present for API consistency.

Value

A list containing one dq_result with status "INFO".

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_row_count(df, cfg)
```

check_schema_contract *SC-01 / SC-02: Check columns against the expected schema contract*

Description

Compares the columns present in df against config$expected_columns:

- **SC-01**: one "FAIL" result per column present in the file but not listed in expected_columns.
- **SC-02**: one "FAIL" result per column listed in expected_columns but absent from the file.

Returns an empty list if expected_columns is not configured.

Usage

```
check_schema_contract(df, config)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.

Value

A list of dq_result objects. Each schema violation produces one "FAIL" result; a "PASS" result is emitted for each sub-check when no violations are found. Returns an empty list if expected_columns is not configured.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
check_schema_contract(df, cfg)
```

---

compare_snapshots *Compare two snapshots from the SQLite database*

Description

Reads two historical snapshot records (by ID) from the SQLite database and computes table-level, schema, and per-column statistical drift. Optionally renders an HTML drift report.

Usage

```
compare_snapshots(
  dataset_name,
  snapshot_id_prev = NULL,
  snapshot_id_curr = NULL,
  db_path = NULL,
  config_dir = ".",
  report = TRUE,
  open_report = interactive()
)
```

Arguments

dataset_name	Character. Dataset name to compare.
snapshot_id_prev	Integer or NULL. ID of the earlier snapshot. If NULL, defaults to the second-most-recent snapshot by ID.

snapshot_id_curr
: Integer or NULL. ID of the later snapshot. If NULL, defaults to the most-recent snapshot by ID.

db_path
: Character or NULL. Path to the SQLite snapshot database. If NULL (the default), the path is resolved from snapshot_db the same way run_dq_check resolves it: from <dataset_name>.yml if set there, otherwise dqcheckr.yml, otherwise the built-in default "data/snapshots.sqlite".

config_dir
: Character. Path to the directory containing dqcheckr.yml. Used to read thresholds, report_output_dir, and (when db_path is NULL) snapshot_db.

report
: Logical. Whether to render an HTML drift report.

open_report
: Logical. Whether to open the HTML report in the browser after rendering (only takes effect in interactive sessions).

Value

Invisibly, a named list with elements dataset_name, snap_prev, snap_curr, table_drift, schema_changes, missing_rate_changes, non_numeric_changes, mean_shifts, distinct_changes, and report_path (the full path to the rendered HTML drift report, or NULL when no report was written). Callers should use report_path rather than reconstructing the filename from a pattern.

Note

As with run_dq_check, a relative snapshot_db or report_output_dir from the config resolves against the R process's working directory, not against config_dir.

Examples

```
tmp     <- tempdir()
db_path <- file.path(tmp, "snap.sqlite")
cfg_yml <- file.path(tmp, "dqcheckr.yml")
ds_yml  <- file.path(tmp, "starwars_csv.yml")
dat     <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
writeLines(c(
  paste0('snapshot_db: "', db_path, '"'),
  paste0('report_output_dir: "', tmp, '"'),
  'default_rules:',
  '  max_missing_rate: 0.60',
  '  min_row_count: 80'
), cfg_yml)
writeLines(c(
  'dataset_name: "starwars_csv"',
  paste0('current_file: "', dat, '"'),
  'format: csv',
  'encoding: "UTF-8"',
  'delimiter: ","'
), ds_yml)
run_dq_check("starwars_csv", config_dir = tmp, open_report = FALSE)
run_dq_check("starwars_csv", config_dir = tmp, open_report = FALSE)
drift <- compare_snapshots("starwars_csv", config_dir = tmp, report = FALSE)
names(drift)
```

detect_files *Detect current and previous dataset files*

Description

Resolves the current and previous file paths from the configuration. If `current_file` is set explicitly, it is used directly. Otherwise the two most recently modified files in `folder` are used.

Usage

```
detect_files(config)
```

Arguments

`config`	Named list. Merged configuration as returned by `load_config`.

Value

A named list with elements `current` (character path) and `previous` (character path or `NULL`).

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg <- load_config("starwars_csv", config_dir = cfg_dir)
cfg$current_file <- system.file("demonstrations/data/starwars.csv",
                                package = "dqcheckr")
files <- detect_files(cfg)
files$current
```

dq_result *Construct a data quality result object*

Description

Creates the atomic result unit returned by every check function.

Usage

```
dq_result(
  check_id,
  check_name,
  column = NA_character_,
  status,
  observed,
  threshold = NA_character_,
  message
)
```

Arguments

check_id	Character. Short identifier for the check (e.g. "QC-01").
check_name	Character. Human-readable name of the check.
column	Character. Column the check applies to, or NA_character_ for row-level or file-level checks.
status	Character. One of "PASS", "WARN", "FAIL", or "INFO".
observed	Character. What was observed (e.g. "5.2% missing").
threshold	Character. The configured threshold, or NA_character_ if not applicable.
message	Character. Human-readable description of the result.

Value

A named list with seven elements: check_id, check_name, column, status, observed, threshold, message. Every string field accepts any length-1 atomic value and is coerced with as.character(); a non-scalar value aborts with dqcheckr_invalid_argument.

Examples

```
dq_result("QC-01", "Missing rate", column = "age",
          status = "PASS", observed = "0% missing",
          message = "No missing values.")
```

---

generate_dataset_config

Generate a dataset config from a delivery file

Description

Sniffs the delivery (sniff_dataset) and writes a fully-optioned, self-documenting YAML config: every config key appears, detected values set live, optional settings commented out with their defaults shown, each key documented with the same description the validator's vocabulary uses. Duplicate header names arrive renamed positionally (with # was "..." annotations and csv_skip: 1); fixed-width files get a character-position ruler comment above fwf_widths, and a packed file (no detectable boundaries) gets explicit TODO widths that validate_config refuses to run until filled in.

Usage

```
generate_dataset_config(path, config_dir = ".", dataset_name = NULL)
```

Arguments

path	Character. Path to the delivery file to sniff.
config_dir	Character. Directory to write <dataset_name>.yml into (created if absent). Defaults to ".".
dataset_name	Character or NULL. Config identity; defaults to the delivery filename without extension, sanitised to [A-Za-z0-9_-].

Details

Create-only. An existing config for the dataset is never overwritten – generation aborts with dqcheckr_config_exists and the file is untouched. Once generated, a config is owned by hand edits; to re-sniff a changed delivery, generate under a different name (or into a scratch directory) and diff. The write is crash-safe (emitted to a temp file, renamed into place once complete); see the *Accepted design decisions* section of the specification vignette for the residual concurrent-generation caveat.

Value

Invisibly, the path of the written config file.

Examples

```
tmp <- file.path(tempdir(), "gen-demo")
dir.create(tmp, showWarnings = FALSE)
f <- file.path(tmp, "orders.csv")
writeLines(c("id,amount", "A1,10", "A2,20"), f)
cfg <- generate_dataset_config(f, config_dir = tmp)
# A deployment needs the global config too: validate_config() (like
# run_dq_check()) requires dqcheckr.yml to be present.
generate_global_config(tmp)
validate_config("orders", config_dir = tmp)$valid
```

`generate_global_config` *Generate the global configuration file*

Description

Writes a fully-optioned `dqcheckr.yml`: the shared infrastructure paths set live to their built-in defaults (visible and editable), and a commented `default_rules` block listing *every* rule key with its default value and description – the full rule vocabulary in one place, so thresholds can be changed by uncommenting a line rather than consulting the manual. Rules with no default are shown with an example value and a note that leaving them unset keeps the check off.

Usage

```
generate_global_config(config_dir = ".")
```

Arguments

`config_dir`	Character. Directory to write `dqcheckr.yml` into (created if absent). Defaults to `"."`.

Details

Create-only, like `generate_dataset_config`: an existing `dqcheckr.yml` is never overwritten – generation aborts with `dqcheckr_config_exists` and the file is untouched. The write is crash-safe (emitted to a temp file, renamed into place once complete); see the *Accepted design decisions* section of the specification vignette for the residual concurrent-generation caveat.

Value

Invisibly, the path of the written config file.

Examples

```
tmp <- file.path(tempdir(), "gen-global-demo")
cfg <- generate_global_config(config_dir = tmp)
yaml::read_yaml(cfg)$snapshot_db
```

infer_col_type *Infer the logical type of a character column*

Description

Classifies a character vector as "date", "numeric", "character", or "unknown" by applying rules in priority order.

Usage

```
infer_col_type(x, threshold = 0.9)
```

Arguments

x	Character vector to classify (as read from a CSV or FWF file).
threshold	Numeric. Minimum proportion of non-empty values that must parse as numeric for the column to be classified as "numeric". Defaults to 0.90. Configurable via type_inference_threshold in rule_overrides.

Details

Date formats are tried in this fixed precedence order: "%Y-%m-%d", "%d/%m/%Y", "%m/%d/%Y", "%Y%m%d", "%d-%m-%Y", and the two ISO-style date-time shapes "%Y-%m-%d %H:%M:%S" and "%Y-%m-%dT%H:%M:%S" (a timestamp column classifies as "date"; the type is a label with no time-of-day-aware stat or drift, so the time part carries no separate semantics). A column is classified as "date" only when *every* non-empty value both matches that format's exact character shape and parses as a valid calendar date; a single malformed date therefore flips the whole column to "numeric" or "character" (such flips between deliveries are surfaced by check CP-02c). The shape is anchored, so a value with trailing characters ("2024-01-15x") or extra digits (the 9-digit "202401159") is *not* treated as a date. Two caveats follow from the precedence rules: ambiguous day/month values resolve day-first ("%d/%m/%Y" is tried before "%m/%d/%Y"), and all-8-digit identifier columns whose values happen to be valid "%Y%m%d" dates classify as dates. Pin the type with an entry in the column_types config map when the heuristic gets a column wrong.

Value

A single character string: "date", "numeric", "character", or "unknown".

Examples

```
infer_col_type(c("2024-01-01", "2024-06-15"))   # "date"
infer_col_type(c("2024-01-01 09:59:22"))         # "date" (date-time)
infer_col_type(c("1.5", "2.0", "3.1"))          # "numeric"
infer_col_type(c("high", "low", "medium"))       # "character"
infer_col_type(c(NA, "", NA))                    # "unknown"
infer_col_type(c(rep("1", 17), "a", "b", "c"), threshold = 0.80)  # "numeric"
```

`list_runs` *List recent runs for a dataset by name*

Description

Name-based wrapper around `read_recent_snapshots`: resolves the snapshot database from the dataset's configuration exactly as `run_dq_check` does (the merged global + dataset config's `snapshot_db`, defaulting to `"data/snapshots.sqlite"`), so the whole workflow can be driven by dataset names without ever hand-typing a database path. The returned `id` column is the snapshot id that `compare_snapshots` takes to compare a specific pair of runs.

Usage

```
list_runs(dataset_name, config_dir = ".", n = 10)
```

Arguments

`dataset_name`	Character. Dataset name; must match `<dataset_name>.yml` in `config_dir`.
`config_dir`	Character. Path to the directory containing `dqcheckr.yml` and the dataset YAML file. Defaults to `"."`.
`n`	Integer. Maximum number of records to return. Defaults to 10.

Value

The data frame documented in `read_recent_snapshots`: one row per run, most recent first, empty (with the full column schema) when the database does not exist or holds no rows for the dataset.

Note

As with `run_dq_check`, a relative `snapshot_db` resolves against the R process's *working directory*, not against `config_dir`. Call from the deployment root or use an absolute path in the config, or the lookup will (like a run started from the wrong directory) point at a database that isn't the deployment's.

Examples

```
tmp <- gsub("\\\\", "/", tempdir())
writeLines(paste0('snapshot_db: "', tmp, '/snap.sqlite"'),
           file.path(tmp, "dqcheckr.yml"))
writeLines(c('dataset_name: "demo"', 'format: csv'),
           file.path(tmp, "demo.yml"))
list_runs("demo", config_dir = tmp)   # empty frame: no runs recorded yet
```

list_snapshots *List snapshots available in the database*

Description

Returns a data frame of snapshot records for the given dataset (or all datasets if `dataset_name` is `NULL`), ordered by dataset name and snapshot ID.

Usage

```
list_snapshots(dataset_name = NULL, db_path = NULL)
```

Arguments

`dataset_name`	Character or `NULL`. If supplied, only snapshots for that dataset are returned. If `NULL`, all datasets are returned.
`db_path`	Character. Path to the SQLite snapshot database. Required; there is no default (a relative default would be path-sensitive).

Value

A data frame with columns `id`, `dataset_name`, `file_name`, `run_timestamp`, `row_count`, `overall_status`. Returns an empty data frame if the database does not exist or contains no matching records.

Examples

```
list_snapshots(db_path = tempfile(fileext = ".sqlite"))
```

load_config *Load and merge dataset configuration*

Description

Reads the global `dqcheckr.yml` and the dataset-specific YAML, merging `rule_overrides` from the dataset config on top of `default_rules` from the global config. Top-level keys `snapshot_db` and `report_output_dir` are inherited from the global config when absent from the dataset config.

Usage

```
load_config(dataset_name, config_dir)
```

Arguments

`dataset_name`	Character. Dataset name; must match `<dataset_name>.yml` in `config_dir`.
`config_dir`	Character. Path to the directory containing both YAML files.

Value

A named list representing the merged configuration.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg <- load_config("starwars_csv", config_dir = cfg_dir)
cfg$format
```

`overall_status` *Compute the worst status across a list of dq_result objects*

Description

Returns the single worst status in precedence order: `"FAIL"` > `"WARN"` > `"PASS"` > `"INFO"`.

Usage

```
overall_status(results)
```

Arguments

`results`	A list of `dq_result` objects.

Value

A single character string: `"FAIL"`, `"WARN"`, `"PASS"`, or `"INFO"`.

Examples

```
r1 <- dq_result("QC-01", "test", status = "PASS", observed = "ok", message = "ok")
r2 <- dq_result("QC-02", "test", status = "WARN", observed = "ok", message = "ok")
overall_status(list(r1, r2))  # "WARN"
```

`read_dataset` *Read a dataset file into a data frame*

Description

Reads a CSV or fixed-width file, coercing all columns to character and trimming whitespace. Encoding and delimiter are taken from `config`. A declared encoding of ASCII (or a formal alias such as `US-ASCII`) is read as UTF-8: ASCII is a strict subset of UTF-8, so this is lossless, and it protects against deliveries whose non-ASCII bytes appear beyond any sample a sniffer looked at. When the effective encoding is UTF-8 the whole file is validity-scanned before parsing; a delivery that is not valid UTF-8 is read using a single-byte fallback encoding instead, and the mismatch is surfaced by `check_file_encoding` (QC-16) as a FAIL result rather than crashing the run.

Usage

```
read_dataset(path, config)
```

Arguments

`path`	Character. Path to the file to read.
`config`	Named list. Merged configuration as returned by `load_config`. Must include `format` (`"csv"` or `"fwf"`). For CSV files, `col_names` (an explicit column-name list) and `csv_skip` (number of leading lines to drop, e.g. a real header row that is being replaced by `col_names`) are optional and default to using the file's own header and `0L` respectively. For FWF files, `fwf_widths` is required and `fwf_col_names` and `fwf_skip` are optional.

Value

A data frame with all columns as character vectors.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg  <- load_config("starwars_csv", config_dir = cfg_dir)
path <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df   <- read_dataset(path, cfg)
```

`read_recent_snapshots` *Read recent snapshot history from the SQLite database*

Description

Retrieves the `n` most recent run records for a given dataset from the snapshot database, ordered newest-first.

Usage

```
read_recent_snapshots(db_path, dataset_name, n = 10)
```

Arguments

`db_path`	Character. Path to the SQLite database file.
`dataset_name`	Character. Dataset name to filter on.
`n`	Integer. Maximum number of records to return. Defaults to 10.

Value

A data frame with one row per run and columns including `id`, `dataset_name`, `run_timestamp`, `file_name`, `row_count`, `col_count`, `overall_status`, `check_pass_count`, `check_warn_count`, `check_fail_count`, `check_info_count`, `new_cols_vs_previous`, `missing_cols_vs_previous`, `new_cols_vs_schema`, `missing_cols_vs_schema`, `comparison_mode`, `render_status`, `type_changed_cols_vs_previou` and `report_file` (the rendered report's filename, `NA` for snapshots written before dqcheckr 0.2.3). `render_status` is one of `"pending"` (0.2.5+: the row was written but its report has not finished rendering yet – `report_file` is `NA` in this window), `"success"` (report written; `report_file` names it), or `"failed"` (render skipped or errored; `report_file` is `NA`). Consumers linking to a report should treat a `"pending"` row as not-yet-available rather than reconstructing a filename for a report that does not exist. Returns an empty data frame with the same columns if the database does not exist or contains no records for the dataset. If the database exists but cannot be read (corrupt file, permissions, an unresolved lock), it emits a warning naming the cause and returns the same empty data frame, so a read failure is visible rather than masquerading as an empty history.

Examples

```
history <- read_recent_snapshots(tempfile(fileext = ".sqlite"), "starwars_csv")
```

resolve_col_type *Resolve the effective type of a column, respecting config overrides*

Description

Returns the type for `col` from the `column_types` map in `config` if one is set, otherwise falls back to `infer_col_type`. Use this in custom check scripts instead of calling `infer_col_type()` directly so that type overrides are respected.

Usage

```
resolve_col_type(col, x, config)
```

Arguments

`col`	Character. Column name.
`x`	Character vector. The column's values (as read from the file).
`config`	Named list. Merged configuration as returned by `load_config`.

Value

A single character string: `"date"`, `"numeric"`, `"character"`, or `"unknown"`.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg <- load_config("starwars_csv", config_dir = cfg_dir)
resolve_col_type("name", c("Luke", "Leia", "Han"), cfg)   # "character"
```

run_comparison_checks *Run all version comparison checks between two dataset snapshots*

Description

Runs CP-01 to CP-08 comparing a current delivery against the previous one.

Usage

```
run_comparison_checks(
  df_current,
  df_previous,
  config,
  types_current = NULL,
  types_previous = NULL
)
```

Arguments

df_current	A data frame. The current delivery.
df_previous	A data frame. The previous delivery.
config	Named list. Merged configuration as returned by load_config.
types_current, types_previous	Optional named character vectors of pre-resolved column types for the current and previous data frames (as produced by resolve_col_type per column). When NULL (the default), types are resolved once here and shared by all type-dependent comparison checks.

Value

A list of dq_result objects. The list carries attributes new_cols, dropped_cols, and type_changed_cols (character vectors) for use by the snapshot writer.

Examples

```
cfg_dir   <- system.file("demonstrations/config", package = "dqcheckr")
cfg       <- load_config("starwars_csv", config_dir = cfg_dir)
curr_path <- system.file("demonstrations/data2/starwars_v2.csv", package = "dqcheckr")
prev_path <- system.file("demonstrations/data2/starwars_v1.csv", package = "dqcheckr")
curr      <- read_dataset(curr_path, cfg)
prev      <- read_dataset(prev_path, cfg)
results   <- run_comparison_checks(curr, prev, cfg)
```

---

run_custom_checks *Run organisation-specific custom checks*

Description

Sources the R file specified by config$custom_checks_file, which must define a function custom_checks(df) returning a list of dq_result objects. Returns an empty list if custom_checks_file is not set in the config.

Usage

```
run_custom_checks(df, config)
```

Arguments

df	A data frame. The current delivery.
config	Named list. Merged configuration as returned by load_config.

Details

The file is sourced into an isolated environment whose parent is `baseenv()`, so only base R functions are available by default. `dq_result` is explicitly injected and can be called without qualification. All other dqcheckr exports (e.g. `resolve_col_type`, `infer_col_type`) must be qualified: `dqcheckr::resolve_col_type()`. Any error – missing file, undefined function, runtime failure, or a malformed result element (each element must have the seven `dq_result` fields and a valid status) – stops the run with a clear message.

Value

A list of `dq_result` objects (may be empty).

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg     <- load_config("starwars_csv", config_dir = cfg_dir)
path    <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df      <- read_dataset(path, cfg)
results <- run_custom_checks(df, cfg)
```

---

run_dq_check *Run a full data quality check pipeline*

Description

Orchestrates the complete dqcheckr pipeline: validates the configuration, loads it, detects files, runs QC and comparison checks, writes a snapshot to SQLite, and renders an HTML report.

Usage

```
run_dq_check(dataset_name, config_dir = ".", open_report = TRUE)
```

Arguments

`dataset_name`	Character. Name of the dataset; must match a YAML config file `<dataset_name>.yml` in `config_dir`.
`config_dir`	Character. Path to the directory containing `dqcheckr.yml` and the dataset YAML file. Defaults to `"."`.
`open_report`	Logical. Whether to open the HTML report in the browser after rendering (only takes effect in interactive sessions).

Details

Validation (`validate_config`) runs first, before any other work: error-severity findings abort with a `dqcheckr_validation_error` condition whose message lists every finding, and no snapshot row is written. Warning-severity findings are reported via `message()` and the run proceeds. Run `validate_config()` standalone to see all findings after editing a config without starting a run.

Value

Invisibly, a named list with:

status Overall status string: `"PASS"`, `"WARN"`, `"FAIL"`, or `"INFO"`.

report_path Absolute path to the rendered HTML report, or `NULL` if rendering was skipped (no Quarto CLI) or failed. Both cases record `render_status = "failed"`; the skipped-vs-failed distinction is visible only in the run log, not the database.

snapshot_id Integer row ID of the snapshot written to SQLite, or `NULL` if the write failed.

Note

Relative `snapshot_db` and `report_output_dir` config values resolve against the R process's *working directory*, not against `config_dir`. Run from the deployment root (the directory containing `config/`, `data/`, `reports/`) or use absolute paths in the config; otherwise a fresh snapshot database is silently created wherever the process happens to be running.

Examples

```
tmp <- gsub("\\\\", "/", tempdir())
dat <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
writeLines(c(
  paste0('snapshot_db: "',       tmp, '/snap.sqlite"'),
  paste0('report_output_dir: "', tmp, '"'),
  'default_rules:',
  '  max_missing_rate: 0.60',
  '  min_row_count: 80'
), file.path(tmp, "dqcheckr.yml"))
writeLines(c(
  'dataset_name: "starwars_csv"',
  paste0('current_file: "', dat, '"'),
  'format: csv',
  'encoding: "UTF-8"',
  'delimiter: ","'
), file.path(tmp, "starwars_csv.yml"))
result <- run_dq_check("starwars_csv", config_dir = tmp, open_report = FALSE)
result$status
```

`run_qc_checks` *Run all generic quality checks on a dataset*

Description

Runs the full QC check suite (QC-01 to QC-16, SC-01, SC-02) against a single data frame snapshot.

Usage

```
run_qc_checks(df, config, file_path = NULL, types = NULL)
```

Arguments

df	A data frame with all columns as character vectors (as returned by read_dataset).
config	Named list. Merged configuration as returned by load_config.
file_path	Character or NULL. Absolute path to the file, used for the optional max_file_size_mb check in QC-14.
types	Optional named character vector of pre-resolved column types; see check_inferred_types. When NULL (the default), types are resolved once here and shared by all type-dependent checks.

Value

A list of dq_result objects.

Examples

```
cfg_dir <- system.file("demonstrations/config", package = "dqcheckr")
cfg     <- load_config("starwars_csv", config_dir = cfg_dir)
path    <- system.file("demonstrations/data/starwars.csv", package = "dqcheckr")
df      <- read_dataset(path, cfg)
results <- run_qc_checks(df, cfg)
```

---

sniff_dataset *Sniff a delivery file's structure*

Description

Pure inference, no side effects: inspects a delivery file and returns what a config would need – format, encoding, delimiter/quote, header presence, column names (with duplicate header names renamed positionally), per-column types, and key-column candidates – as a plain list. This is the detection half of the config generator; the writer turns it into a commented YAML.

Usage

```
sniff_dataset(path)
```

Arguments

path	Character. Path to the delivery file.

Details

Encoding uses the same full-file streamed scan as `read_dataset()` (`scan_file_encoding()`): a file that is not valid UTF-8 gets the same single-byte fallback the reader would use, recorded in `encoding_guess`. Types come from `infer_col_type` – the one implementation, so the sniff can never disagree with run-time classification. Structure detection reads a bounded head sample (`n_sample_rows` reports how many data rows informed it), never the file body.

Fixed-width files: when the sampled lines are all the same width and no delimiter splits them, boundaries are guessed with `readr::fwf_empty()` (blank-gutter detection). A packed file (no blank gutters) sets `fwf_packed = TRUE` and `fwf_widths = NULL` rather than guessing wrongly – the generator emits a `TODO` the validator refuses to run.

Every headline field's origin is recorded in `provenance` (`"detected"`, `"default"`, or `"generated"`), so the writer can emit detected values live and defaults commented out.

Value

A named list: `path`, `format` (`"csv"`/`"fwf"`), `encoding`, `encoding_valid_utf8`, `encoding_guess`, `bom`, `delimiter`, `quote_char`, `header`, `csv_skip`, `col_names`, `renamed_from` (new name -> original, `NULL` unless duplicates were renamed), `fwf_widths`, `fwf_col_names`, `fwf_packed`, `column_types`, `key_column_candidates`, `expected_columns`, `n_sample_rows`, `provenance`.

Examples

```
f <- tempfile(fileext = ".csv")
writeLines(c("id,amount", "A1,10", "A2,20"), f)
s <- sniff_dataset(f)
s$format; s$col_names; s$column_types
```

`validate_config` *Validate a dataset's configuration*

Description

Checks the global `dqcheckr.yml` and the dataset's YAML against the config vocabulary: unknown keys (with a did-you-mean suggestion), value types and ranges, rule placement (rules-level vs per-column), positional-list consistency (`fwf_col_names` vs `fwf_widths` lengths, duplicate column names), unresolved generator `TODO` placeholders, and the presence of a file source. All findings are collected and returned in one pass – nothing aborts on the first problem – so a hand-edited config can be fixed in one round trip.

Usage

```
validate_config(dataset_name, config_dir = ".")
```

Arguments

dataset_name	Character. Dataset name; must match <dataset_name>.yml in config_dir.
config_dir	Character. Path to the directory containing dqcheckr.yml and the dataset YAML file. Defaults to ".".

Details

Validation runs in two tiers. **Tier 1 (config-only)** reads nothing but the two YAML files and always runs. **Tier 2 (header cross-check)** additionally opens just the first line(s) of the delivery the config points at – never the file body, so it is cheap even for multi-GB files on a network share – and verifies the config against reality: col_names length vs the physical column count, key_columns/expected_columns/column_types/ column_rules naming columns that exist, and fwf_widths summing to the record length. When no delivery file is resolvable (config written ahead of the first delivery, empty folder, unreadable header), Tier 2 is skipped and the skip is *stated* in the result (tier2_skipped) and by print() – a verdict always says which tier it reached.

Findings have three severities, split by one rule: **config mistakes are errors** (wrong types, broken positional lists, misplaced rule keys, a missing file source — things only an edit can cause) and **delivery-facing findings are warnings** (a key column absent from the file, a column-count mismatch — these can equally mean the supplier changed the delivery, and drift must be recorded by a completed run, not abort it). "note" is informational. Unknown keys warn, so hand-kept extra keys round-trip. All Tier-2 findings are therefore warnings by construction.

A *dataset* config file that cannot be read at all aborts with a typed condition rather than returning findings: dqcheckr_missing_file, dqcheckr_empty_config, dqcheckr_config_parse_error, or dqcheckr_invalid_config (not a YAML key map). An empty or comments-only *global* dqcheckr.yml is tolerated as all-defaults with a warning finding, matching how the package has always run such deployments.

Value

An object of class dqcheckr_validation: a list with dataset_name, config_dir, findings (a data frame with columns file, key, severity, message), tier ("config+header" when Tier 2 ran, else "config-only"), tier2_skipped (NULL, or the reason Tier 2 did not run), and valid (TRUE when no error-severity findings exist).

Examples

```
tmp <- gsub("\\\\", "/", tempdir())
writeLines('snapshot_db: "snap.sqlite"', file.path(tmp, "dqcheckr.yml"))
writeLines(c('dataset_name: "demo"', 'format: csv', 'current_file: "x.csv"'),
           file.path(tmp, "demo.yml"))
v <- validate_config("demo", config_dir = tmp)
v$valid
```

Index