

Package: hvtiRdatabuild (via r-universe)

October 21, 2026

Type Package

Title Build and Verify Analytic Datasets for the HVTI CORR Group

Version 0.3.2

Date 2026-10-20

License GPL (>= 3)

Encoding UTF-8

URL <https://github.com/ehrlinger/hvtiRdatabuild>,
<https://ehrlinger.github.io/hvtiRdatabuild/>

BugReports <https://github.com/ehrlinger/hvtiRdatabuild/issues>

Description Builds analysis-ready clinical datasets for the clinical investigations statistics group within The Heart and Vascular Institute at the Cleveland Clinic, and verifies them against the legacy 'SAS' datasets they replace.

Depends R (>= 4.4.0)

Imports DBI, digest, dplyr, filelock, haven, hvtiRutilities (>= 1.5.2), tools, utils, yaml

Remotes ehrlinger/hvtiPlotR, ehrlinger/hvtiRutilities

Suggests arrow, duckdb, hvtiPlotR, jsonlite, keyring, knitr, odbc, quarto, testthat (>= 3.0.0), tidyselect, withr

VignetteBuilder quarto

SystemRequirements Quarto command line tool
(<<https://github.com/quarto-dev/quarto-cli>>)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Roxygen list(markdown = TRUE)

Config/pak/sysreqs make libicu-dev libx11-dev zlib1g-dev

Repository <https://ehrlinger.r-universe.dev>

Date/Publication 2026-10-20 17:21:51 UTC

RemoteUrl <https://github.com/ehrlinger/hvtiRdatabuild>

RemoteRef HEAD
RemoteSha 724235ab06959bf822b9764eb4e48c1c6b8f9daf

Contents

append_cohort_stage	2
backfill_corrections	3
build_dataset	5
cohort_summary	6
compare_built	7
decide_correction	8
derive_vars	10
dw_connect	11
dw_modules	13
dw_pull	13
lift_master	14
new_cohort_record	16
print.build_result	17
print.built_comparison	17
print.derive_result	18
print.pull_result	18
propose_correction	19
publish_dataset	21
read_analysis_set	22
read_master_config	24
read_study_config	25
snapshot_master	26
snapshot_oracle	27
str.build_result	29
str.built_comparison	29
str.cohort_record	30
str.derive_result	31
withdraw_dataset_release	31
write_analysis_set	33

Index	35
--------------	-----------

append_cohort_stage	<i>Append an outcome stage to a cohort history</i>
---------------------	--

Description

Append an outcome stage to a cohort history

Usage

append_cohort_stage(cohort, stage, outcomes)

Arguments

cohort	A cohort_record created by <code>new_cohort_record()</code> .
stage	A one-row data frame with the stages schema.
outcomes	A data frame with one patient_id, included, and reason outcome for every patient active at the preceding stage.

Value

An updated object of class cohort_record.

Examples

```
# Synthetic identifiers; no PHI.
cohort <- new_cohort_record(
  members = data.frame(
    patient_id = c("SYN101", "SYN102", "SYN103"), stage_id = "selected",
    included = TRUE, reason = NA_character_
  ),
  stages = data.frame(
    stage_id = "selected", order = 1L, label = "Selected",
    source = "upstream", provenance = "synthetic selection"
  ),
  patient_id_col = "masterid"
)
cohort <- append_cohort_stage(
  cohort,
  stage = data.frame(
    stage_id = "adults", order = 2L, label = "Adults only",
    source = "derive", provenance = "synthetic rule"
  ),
  outcomes = data.frame(
    patient_id = c("SYN101", "SYN102", "SYN103"),
    included = c(TRUE, FALSE, TRUE),
    reason = c(NA, "synthetic exclusion", NA)
  )
)
cohort
```

backfill_corrections	<i>Backfill a master's corrections from its SAS build, and regenerate its view</i>
----------------------	--

Description

Creates the master's corrections and decisions tables if absent, reads the inline patient-level fixes from the SAS build program, records each one that resolves to exactly one record as a correction with an unknown prior and a bake decision (present in the snapshot, not applied), and regenerates the view and its stale view. Fixes that change a key column, match no record, match several, or cannot be read are reported by reason and line number and never recorded.

Usage

```
backfill_corrections(config, con, dry_run = TRUE, dialect = "mssql")
```

Arguments

config	A master_config from read_master_config() .
con	A DBI connection.
dry_run	If TRUE, the default, report what would be recorded.
dialect	"mssql" or "duckdb".

Details

Requires a passing parity record from [lift_master\(\)](#). A dry run, the default, parses and resolves and writes nothing.

Value

Invisibly, a list of counts: dry_run, facts, resolved, unresolved, appended and stale.

See Also

[propose_correction\(\)](#), [decide_correction\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE) &&
    requireNamespace("tidyselect", quietly = TRUE) &&
    requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("withr", quietly = TRUE) &&
    requireNamespace("digest", quietly = TRUE)) {
  dir <- tempfile("backfill")
  dir.create(dir)
  pq <- file.path(dir, "built_demo.parquet")
  arrow::write_parquet(data.frame(id = c("K1", "K2"), age = c(60, 61)), pq)
  jsonlite::write_json(
    list(parquet_sha256 = digest::digest(pq, algo = "sha256", file = TRUE),
         columns = list(list(variable = "id", r_class = "character"),
                        list(variable = "age", r_class = "numeric"))),
    sub("\\.parquet$", ".meta.json", pq), auto_unbox = TRUE)
  writelines(c("data m; set base;", "if id = 'K1' then age = 60;", "run;"),
             file.path(dir, "bd.sas"))
  cfg_path <- file.path(dir, "master.yml")
  writelines(c("name: master_demo", "key: [id]", paste0("snapshots: ", dir),
              "current: built_demo.sas7bdat",
              paste0("build_program: ", file.path(dir, "bd.sas"))), cfg_path)
  cfg <- read_master_config(cfg_path)
  con <- DBI::dbConnect(duckdb::duckdb())
  lift_master(cfg, con, pq, dry_run = FALSE, dialect = "duckdb")
}
```

```

    backfill_corrections(cfg, con, dry_run = FALSE, dialect = "duckdb")
    DBI::dbDisconnect(con, shutdown = TRUE)
  }

```

build_dataset

Assemble the patient-level dataset

Description

Joins the patient-level base and vital-status modules while retaining repeated-event modules as supporting tables. The supplied cohort history is carried forward and receives a build stage after membership is verified.

Usage

```
build_dataset(raw, config, cohort)
```

Arguments

raw	A pull_result from dw_pull() .
config	A study_config from read_study_config() .
cohort	A cohort_record whose latest stage identifies the selected build membership.

Value

An object of class build_result with data, cohort, and tables elements. tables contains unjoined repeated-event module data.

Examples

```

# Synthetic identifiers and values; no PHI, no warehouse connection.
ids <- c("SYN201", "SYN202", "SYN203")
cohort <- new_cohort_record(
  members = data.frame(
    patient_id = ids, stage_id = "selected", included = TRUE,
    reason = NA_character_
  ),
  stages = data.frame(
    stage_id = "selected", order = 1L, label = "Selected",
    source = "upstream", provenance = "synthetic selection"
  ),
  patient_id_col = "masterid"
)
path <- tempfile(fileext = ".yaml")
yaml::write_yaml(list(
  study = "synthetic", cohort_table = "synthetic.cohort",
  warehouse = "synthetic_warehouse", view_schema = "dbo",

```

```

pull_date = "2026-10-01", modules = list("base", "vitalstatus"),
derive = list(scale_value = TRUE)
), path)
config <- read_study_config(path)
unlink(path)
raw <- structure(list(
  tables = list(
    bdbase = data.frame(masterid = ids, base_value = c(3, 5, 8),
                        keep = c(TRUE, FALSE, TRUE)),
    bdstat = data.frame(masterid = ids,
                        status_value = c("alive", "alive", "deceased"))
  ),
  manifest = data.frame(
    module = c("base", "vitalstatus"), output = c("bdbase", "bdstat"),
    n_rows = c(3L, 3L), n_cols = c(3L, 2L),
    pulled_at = as.POSIXct(rep("2026-10-01", 2L), tz = "UTC")
  )
), class = "pull_result")
built <- build_dataset(raw, config, cohort)
built

```

cohort_summary

Summarize cohort attrition without patient identifiers

Description

Summarize cohort attrition without patient identifiers

Usage

```
cohort_summary(cohort)
```

Arguments

cohort A cohort_record.

Value

A list with stages (stage counts) and reasons (exclusion counts) data frames. Neither table contains patient identifiers.

Examples

```

# Synthetic identifiers; no PHI.
cohort <- new_cohort_record(
  members = data.frame(
    patient_id = c("SYN101", "SYN102", "SYN103"), stage_id = "selected",
    included = TRUE, reason = NA_character_
  ),
  stages = data.frame(

```

```

      stage_id = "selected", order = 1L, label = "Selected",
      source = "upstream", provenance = "synthetic selection"
    ),
    patient_id_col = "masterid"
  )
  cohort <- append_cohort_stage(
    cohort,
    stage = data.frame(
      stage_id = "adults", order = 2L, label = "Adults only",
      source = "derive", provenance = "synthetic rule"
    ),
    outcomes = data.frame(
      patient_id = c("SYN101", "SYN102", "SYN103"),
      included = c(TRUE, FALSE, TRUE),
      reason = c(NA, "synthetic exclusion", NA)
    )
  )
  cohort_summary(cohort)$stages

```

compare_built

*Compare an R-built dataset against a SAS oracle***Description**

Joins `r` to oracle on `id` and classifies every variable. Row-set differences are reported separately from value differences: an identifier present on one side only is a cohort discrepancy, a different class of problem from a miscomputed variable, and conflating the two hides both.

Usage

```
compare_built(oracle, r, id = "ccfidu", tolerance = 1e-08)
```

Arguments

<code>oracle</code>	Data frame read from the parquet oracle. See snapshot_oracle() .
<code>r</code>	Data frame produced by the R pipeline.
<code>id</code>	Name of the identifier column present in both. Must be unique within each.
<code>tolerance</code>	Numeric. Absolute numeric differences at or below this are reported as "within_tolerance". A single absolute threshold cannot serve columns of very different magnitude; see the "Coming from SAS" vignette for how to read <code>max_rel_diff</code> alongside it.

Details

This function deliberately returns a table and never a single pass/fail. Collapsing several hundred variables into one boolean launders real differences into a green check. Read the table.

A difference does not automatically mean the R code is wrong. SAS orders missing numerics below all values, so if `bmi < 18.5` classifies a missing BMI as underweight where R yields NA. Record such cases as `intentional_divergence` in `equivalence_signoff.yaml`.

Value

An object of class `built_comparison`: a data frame with one row per variable and columns `variable`, `n_common`, `verdict`, `n_differ`, `max_abs_diff`, `max_rel_diff`, and `detail`. `n_common` is the number of rows compared (constant across all rows, so it survives `write.csv()` or `as.data.frame()`, unlike the `rows` attribute). `verdict` is one of "identical", "within_tolerance", "differs", "absent_in_r", "absent_in_sas", or "type_mismatch". The `rows` attribute holds a list with `n_oracle`, `n_r`, `n_common`, `only_oracle`, and `only_r`.

See Also

[snapshot_oracle\(\)](#)

Examples

```
o <- data.frame(ccfidu = c("A1", "A2"), age = c(65, 70))
r <- data.frame(ccfidu = c("A1", "A2"), age = c(65, 71))
compare_built(o, r, id = "ccfidu")
```

<code>decide_correction</code>	<i>Decide a correction: accept, reject, supersede or bake it</i>
--------------------------------	--

Description

Records the decision in the master's decisions table; the correction applies only when its latest decision is "accept". `dry_run = TRUE` validates and returns the row that would be written, without writing it.

Usage

```
decide_correction(
  config,
  con,
  correction_id,
  decision,
  decided_by,
  reason = NA_character_,
  dry_run = FALSE,
  dialect = "mssql"
)
```

Arguments

<code>config</code>	A <code>master_config</code> from read_master_config() .
<code>con</code>	A DBI connection.
<code>correction_id</code>	Character. The correction being decided.

decision	Character. One of DECISIONS.
decided_by	Character. Who is deciding.
reason	Character, or NA_character_, the default. A note on the decision.
dry_run	If TRUE, validate and return the row without writing it. The default, FALSE, writes it.
dialect	"mssql" or "duckdb".

Details

Requires the master's corrections tables, which `backfill_corrections()` creates.

Value

Invisibly, a list with verdict ("recorded", or "validated" on a dry run), decision_id, and row (the one-row data frame that was or would be appended).

See Also

`backfill_corrections()`, `propose_correction()`

Examples

```
if (requireNamespace("duckdb", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE) &&
    requireNamespace("tidyselect", quietly = TRUE) &&
    requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("withr", quietly = TRUE) &&
    requireNamespace("digest", quietly = TRUE)) {
  dir <- tempfile("decide")
  dir.create(dir)
  pq <- file.path(dir, "built_demo.parquet")
  arrow::write_parquet(data.frame(id = c("K1", "K2"), age = c(60, 61)), pq)
  jsonlite::write_json(
    list(parquet_sha256 = digest::digest(pq, algo = "sha256", file = TRUE),
         columns = list(list(variable = "id", r_class = "character"),
                        list(variable = "age", r_class = "numeric"))),
    sub("\\.parquet$", ".meta.json", pq), auto_unbox = TRUE)
  writeLines("data m; run;", file.path(dir, "bd.sas"))
  cfg_path <- file.path(dir, "master.yml")
  writeLines(c("name: master_demo", "key: [id]", paste0("snapshots: ", dir),
               "current: built_demo.sas7bdat",
               paste0("build_program: ", file.path(dir, "bd.sas"))), cfg_path)
  cfg <- read_master_config(cfg_path)
  con <- DBI::dbConnect(duckdb::duckdb())
  lift_master(cfg, con, pq, dry_run = FALSE, dialect = "duckdb")
  backfill_corrections(cfg, con, dry_run = FALSE, dialect = "duckdb")
  prop <- propose_correction(cfg, con, key_values = list(id = "K1"), variable = "age",
                           expected_prior = 60, new_value = 61,
                           evidence_type = "chart_review", evidence_ref = "invented",
                           asserted_by = "tester", dialect = "duckdb")
}
```

```

    decide_correction(cfg, con, prop$correction_id, "accept", "tester", dialect = "duckdb")
    DBI::dbDisconnect(con, shutdown = TRUE)
  }

```

derive_vars

Apply study-specific variable derivations

Description

Runs committed study-specific R code after dataset assembly. The callback may add analysis variables and append local exclusion stages to the cohort record; it may not rewrite earlier membership decisions.

Usage

```
derive_vars(build, config, study_derive)
```

Arguments

build	A build_result from build_dataset() .
config	A study_config from read_study_config() .
study_derive	A function with arguments data, cohort, and config. It returns a named list with updated data and cohort elements.

Value

An object of class `derive_result` with data, cohort, and unchanged supporting tables elements.

Examples

```

# Synthetic identifiers and values; no PHI, no warehouse connection.
ids <- c("SYN201", "SYN202", "SYN203")
cohort <- new_cohort_record(
  members = data.frame(
    patient_id = ids, stage_id = "selected", included = TRUE,
    reason = NA_character_
  ),
  stages = data.frame(
    stage_id = "selected", order = 1L, label = "Selected",
    source = "upstream", provenance = "synthetic selection"
  ),
  patient_id_col = "masterid"
)
path <- tempfile(fileext = ".yaml")
yaml::write_yaml(list(
  study = "synthetic", cohort_table = "synthetic.cohort",
  warehouse = "synthetic_warehouse", view_schema = "dbo",

```

```

    pull_date = "2026-10-01", modules = list("base", "vitalstatus"),
    derive = list(scale_value = TRUE)
  ), path)
config <- read_study_config(path)
unlink(path)
raw <- structure(list(
  tables = list(
    bdbase = data.frame(masterid = ids, base_value = c(3, 5, 8),
                        keep = c(TRUE, FALSE, TRUE)),
    bdstat = data.frame(masterid = ids,
                        status_value = c("alive", "alive", "deceased"))
  ),
  manifest = data.frame(
    module = c("base", "vitalstatus"), output = c("bdbase", "bdstat"),
    n_rows = c(3L, 3L), n_cols = c(3L, 2L),
    pulled_at = as.POSIXct(rep("2026-10-01", 2L), tz = "UTC")
  )
), class = "pull_result")
built <- build_dataset(raw, config, cohort)
study_derive <- function(data, cohort, config) {
  keep <- data$keep
  if (isTRUE(config$derive[["scale_value"]])) {
    data$analysis_value <- data$base_value * 2
  }
  reason <- ifelse(keep, NA_character_, "synthetic study exclusion")
  stage <- data.frame(
    stage_id = "study_eligibility", order = max(cohort$stages$order) + 1L,
    label = "Study eligibility", source = "derive",
    provenance = "synthetic study rule"
  )
  outcomes <- data.frame(patient_id = data$masterid, included = keep,
                        reason = reason)
  list(data = data[keep, , drop = FALSE],
       cohort = append_cohort_stage(cohort, stage, outcomes))
}
derived <- derive_vars(built, config, study_derive)
derived

```

dw_connect

*Open a connection to the data warehouse***Description**

Resolves credentials through the ladder documented in the design spec and opens an ODBC connection. Credentials never appear in a function argument, a configuration file, a log line, or an error message, and the connection string is never echoed.

Usage

```
dw_connect(
```

```

    server,
    database,
    dsn = NULL,
    port = NULL,
    encrypt = TRUE,
    trust_certificate = TRUE,
    ...
)

```

Arguments

<code>server</code>	Warehouse host name.
<code>database</code>	Database name.
<code>dsn</code>	Optional named ODBC DSN. When supplied, the driver holds the credentials and none enters R's memory.
<code>port</code>	Optional port number.
<code>encrypt</code>	Whether to request an encrypted connection.
<code>trust_certificate</code>	Whether to trust the server certificate without validating it against a CA chain.
<code>...</code>	Further arguments passed to <code>DBI::dbConnect()</code> .

Details

`trust_certificate` defaults to `TRUE` for a specific reason. Microsoft changed the `Encrypt` default from no in *ODBC Driver 17 for SQL Server* to yes in driver 18. A connection string that worked under the older driver fails with a certificate error under 18 unless it either disables encryption or trusts the server certificate. The legacy SAS connection carried `TrustServerCertificate=Yes`, which is why the pull works today. Whether trusting the certificate is the right long-term posture, versus installing the institutional CA chain, is a question for whoever administers the DSN, but changing it silently would break every pull.

Set the `HVI_DW_KERBEROS` environment variable to enable the top rung of the credential ladder, Kerberos integrated authentication: no stored secret is used, and the connection relies on the caller's existing ticket instead. It is read with `as.logical()`, which recognises `"true"`, `"True"`, `"T"`, and `"TRUE"` (and their `FALSE` counterparts). Set it to one of those, not `"yes"` or `"1"`, which `as.logical()` does not recognise and silently reads as `NA`.

File-based credentials (a DSN or `~/Renviron`) are unavailable on Windows: Windows has no POSIX file mode, so a credential file's protection can never be confirmed there, and `dw_connect()` refuses rather than trust it unverified. This package targets the Linux server described in the design spec.

Value

A `DBI::DBIConnection` object.

Examples

```
if (requireNamespace("odbc", quietly = TRUE)) {
  # Requires a configured DSN and warehouse access; not run in checks.
  # conn <- dw_connect(server = "<DW-SERVER>", database = "<DW-DB>",
  #                   dsn = "HVI_DW")
}
```

dw_modules

List the available warehouse modules

Description

Module definitions ship as data under `inst/extdata/modules/`, one YAML file per module, rather than as R code. They change independently of the code that executes them, they carry a description of the SAS block each one ports, and holding them outside R source is what keeps site-specific identifiers out of this repository. The SQL ships with placeholders and the real warehouse, schema, and cohort names arrive from `study.yaml` at run time.

Usage

```
dw_modules()
```

Value

A data frame with one row per module and columns `module`, `output`, `join_key`, and `optional`.

Examples

```
dw_modules()
```

dw_pull

Pull the enabled warehouse modules for a study

Description

Executes one query per module named in the study configuration and returns the raw tables alongside a manifest of what was pulled. This is the R port of `tp.stXXX_dwpull.sas`.

Usage

```
dw_pull(config, conn)
```

Arguments

`config` A `study_config` from `read_study_config()`.

`conn` A `DBI::DBConnection`, typically from `dw_connect()`.

Details

The port is **read-only**. The SAS templates also upload a cohort table to the warehouse; that write is deliberately not carried here and is deferred to a later slice with its own review.

A module that returns zero rows is an error unless its definition declares it optional. A silently empty module produces a quietly incomplete dataset, which is the failure this package exists to prevent.

Value

An object of class `pull_result`: a list with `tables`, a named list of data frames keyed by each module's output name, and `manifest`, a data frame with columns `module`, `output`, `n_rows`, `n_cols`, and `pulled_at`.

See Also

`dw_connect()`, `dw_modules()`, `compare_built()`

Examples

```
# Requires a warehouse connection; see ?dw_connect
# for a runnable version against a mocked connection.
```

lift_master

Lift a master's parquet snapshot into the warehouse as a view

Description

Checks the primary key, creates a base table named for the snapshot's release, loads it row group by row group, and proves it equal to the snapshot: counts, aggregates and a sampled compare first, then a full comparison of every column joined on the key. Only then is a parity record written and the master's view created. If the master already has a corrections table, the view is regenerated with corrections applied.

Usage

```
lift_master(config, con, parquet, dry_run = TRUE, dialect = "mssql")
```

Arguments

config	A master_config from <code>read_master_config()</code> .
con	A DBI connection, such as one from <code>dw_connect()</code> .
parquet	Path to a snapshot written by <code>snapshot_master()</code> ; its <code>.meta.json</code> sidecar must sit beside it, and its checksum must match the sidecar's <code>parquet_sha256</code> .
dry_run	If TRUE, the default, write the DDL and nothing else.
dialect	"mssql" for the warehouse, or "duckdb", used in tests.

Details

A dry run, the default, writes the table's DDL next to the snapshot for a hand-off and touches nothing else. Warehouse errors are reported by step, with the driver's message withheld, because it can carry a data value.

Value

Invisibly, a list with `dry_run`, `base_table`, `ddl_path` and `verdict` ("pass", or NA for a dry run).

See Also

`snapshot_master()`, `backfill_corrections()`

Examples

```
if (requireNamespace("duckdb", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE) &&
    requireNamespace("tidyselect", quietly = TRUE)) {
  dir <- tempfile("lift")
  dir.create(dir)
  pq <- file.path(dir, "built_demo.parquet")
  arrow::write_parquet(data.frame(id = c("K1", "K2"), x = c(1, 2)), pq)
  jsonlite::write_json(
    list(parquet_sha256 = digest::digest(pq, algo = "sha256", file = TRUE),
         columns = list(list(variable = "id", r_class = "character"),
                        list(variable = "x", r_class = "numeric"))),
    sub("\\.parquet$", ".meta.json", pq), auto_unbox = TRUE)
  cfg_path <- file.path(dir, "master.yml")
  writeLines(c("name: master_demo", "key: [id]", paste0("snapshots: ", dir),
              "current: built_demo.sas7bdat",
              paste0("build_program: ", file.path(dir, "bd.sas"))), cfg_path)
  con <- DBI::dbConnect(duckdb::duckdb())
  lift_master(read_master_config(cfg_path), con, pq, dialect = "duckdb")
  DBI::dbDisconnect(con, shutdown = TRUE)
}
```

new_cohort_record	<i>Create an ordered cohort history</i>
-------------------	---

Description

A cohort record stores each patient's outcome at every stage they entered. Patients excluded at a stage do not appear in later stages. Summaries are derived from these outcomes, so stage counts cannot drift from membership.

Usage

```
new_cohort_record(members, stages, patient_id_col)
```

Arguments

members	A data frame with patient_id, stage_id, included, and reason columns. Identifiers are runtime data and are never printed.
stages	A data frame with stage_id, order, label, source, and provenance columns, ordered by order.
patient_id_col	Name of the patient identifier column in associated patient-level data.

Value

An object of class cohort_record with members, stages, and patient_id_col elements.

Examples

```
# Synthetic identifiers; no PHI.
cohort <- new_cohort_record(
  members = data.frame(
    patient_id = c("SYN101", "SYN102", "SYN103"), stage_id = "selected",
    included = TRUE, reason = NA_character_
  ),
  stages = data.frame(
    stage_id = "selected", order = 1L, label = "Selected",
    source = "upstream", provenance = "synthetic selection"
  ),
  patient_id_col = "masterid"
)
cohort
```

```
print.build_result
```

Print a built dataset result without patient values

Description

Print a built dataset result without patient values

Usage

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

Arguments

x	A build_result from build_dataset() .
...	Ignored.

Value

x, invisibly.

```
print.built_comparison
```

Print a dataset comparison

Description

Reports row-set differences and a count per verdict, then lists every variable that is not "identical". No overall pass/fail is printed, by design: see [compare_built\(\)](#).

Usage

```
## S3 method for class 'built_comparison'
print(x, ..., show_ids = getOption("hvtiRdatabuild.show_ids", FALSE))
```

Arguments

x	An object of class built_comparison.
...	Ignored.
show_ids	Logical. Print the identifiers that appear on only one side. Defaults to the hvtiRdatabuild.show_ids option, itself FALSE. Enabling this may emit PHI. Only do so in a session whose output is not being logged or shared.

Details

Identifiers are **not** printed by default. In this group's datasets ccfidu is a medical record number concatenated with a date of surgery, which is PHI; printing it would place PHI in terminals, logs, and captured test failures. The identifiers remain available in the rows attribute for a caller who needs them to chase a discrepancy.

Value

x, invisibly.

print.derive_result	<i>Print a derived dataset result without patient values</i>
---------------------	--

Description

Print a derived dataset result without patient values

Usage

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

Arguments

x	A derive_result from <code>derive_vars()</code> .
...	Ignored.

Value

x, invisibly.

print.pull_result	<i>Print a pull result</i>
-------------------	----------------------------

Description

Print a pull result

Usage

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

Arguments

x	A pull_result from <code>dw_pull()</code> .
...	Ignored.

Value

x, invisibly.

propose_correction	<i>Propose a correction to a single cell of a master</i>
--------------------	--

Description

Validates the proposal, then appends it to the master's corrections table with no decision, so it is not applied until `decide_correction()` accepts it. `dry_run = TRUE` validates and returns the row that would be written, without writing it. A single correction is written by default because proposing one is a deliberate act. No message ever carries a key or a value: an unmatched or ambiguous key is reported by count only, and an over-width or uncastable value is reported by column and limit only.

Usage

```
propose_correction(
  config,
  con,
  key_values,
  variable,
  expected_prior,
  new_value,
  evidence_type,
  evidence_ref,
  asserted_by,
  alt_key = NULL,
  dry_run = FALSE,
  dialect = "mssql"
)
```

Arguments

config	A master_config from <code>read_master_config()</code> .
con	A DBI connection.
key_values	A named list of key values identifying the record: the primary key by default, or alt_key's columns when alt_key is given.
variable	Character. The column to correct; must not be a key or alternate-key column.
expected_prior	The value the record is expected to currently hold, or NA when it is expected to be missing.

<code>new_value</code>	The corrected value, or NA to correct to missing.
<code>evidence_type</code>	Character. One of EVIDENCE_TYPES.
<code>evidence_ref</code>	Character. A reference to the evidence, such as a document id; never a data value.
<code>asserted_by</code>	Character. Who is asserting the correction.
<code>alt_key</code>	Character, or NULL, the default. The name of an alternate key in <code>config\$alt_keys</code> that <code>key_values</code> matches instead of the primary key.
<code>dry_run</code>	If TRUE, validate and return the row without writing it. The default, FALSE, writes it.
<code>dialect</code>	"mssql" or "duckdb".

Details

By default, `key_values` must name exactly the master's primary key columns, each a single non-missing value. When `alt_key` is given instead, `key_values` must name exactly that alternate key's columns, none may be missing, and it must match exactly one row in the master's current base table; the correction is then stored against that row's primary key. Either way, every alternate-key column present in the base table is filled from that row and carried on the correction for reference; `variable` may not be the primary key or any alternate key's column. A failing warehouse call is reported by step, with the driver's message withheld, because it can quote a data value.

Requires the master's corrections tables, which `backfill_corrections()` creates.

Value

Invisibly, a list with `verdict` ("appended", or "validated" on a dry run), `correction_id`, `new_variable` (whether this is the first correction to `variable`, meaning the view should be regenerated), and `row` (the one-row data frame that was or would be appended).

See Also

[backfill_corrections\(\)](#), [decide_correction\(\)](#)

Examples

```
if (requireNamespace("duckdb", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE) &&
    requireNamespace("tidyselect", quietly = TRUE) &&
    requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("withr", quietly = TRUE) &&
    requireNamespace("digest", quietly = TRUE)) {
  dir <- tempfile("propose")
  dir.create(dir)
  pq <- file.path(dir, "built_demo.parquet")
  arrow::write_parquet(data.frame(id = c("K1", "K2"), age = c(60, 61)), pq)
  jsonlite::write_json(
    list(parquet_sha256 = digest::digest(pq, algo = "sha256", file = TRUE),
         columns = list(list(variable = "id", r_class = "character"),
```

```

        list(variable = "age", r_class = "numeric"))),
      sub("\\.parquet$", ".meta.json", pq), auto_unbox = TRUE)
writeLines("data m; run;", file.path(dir, "bd.sas"))
cfg_path <- file.path(dir, "master.yml")
writeLines(c("name: master_demo", "key: [id]", paste0("snapshots: ", dir),
            "current: built_demo.sas7bdat",
            paste0("build_program: ", file.path(dir, "bd.sas"))), cfg_path)
cfg <- read_master_config(cfg_path)
con <- DBI::dbConnect(duckdb::duckdb())
lift_master(cfg, con, pq, dry_run = FALSE, dialect = "duckdb")
backfill_corrections(cfg, con, dry_run = FALSE, dialect = "duckdb")
propose_correction(cfg, con, key_values = list(id = "K1"), variable = "age",
                  expected_prior = 60, new_value = 61, evidence_type = "chart_review",
                  evidence_ref = "invented", asserted_by = "tester", dialect = "duckdb")
DBI::dbDisconnect(con, shutdown = TRUE)
}

```

publish_dataset

Publish an immutable dataset release

Description

A draft is the file a programmer is still allowed to rebuild. Publication copies those bytes to a dated release file and adds the release to `dataset-catalog.yml`. Consuming studies can then discover the release, but they keep reading the release they already pinned until someone explicitly adopts the new one with `hvtiRutilities::adopt_data_update()`.

Usage

```

publish_dataset(
  draft,
  dataset_id,
  datasets_dir,
  extract_date = Sys.Date(),
  source = NULL,
  file_stem = dataset_id
)

```

Arguments

<code>draft</code>	Character. Path to a mutable draft dataset.
<code>dataset_id</code>	Character. Stable logical dataset identifier, using lower-case letters, digits, and underscores.
<code>datasets_dir</code>	Character. Existing directory that owns the published files and <code>dataset-catalog.yml</code> .
<code>extract_date</code>	A Date or YYYY-MM-DD string. Defaults to today.
<code>source</code>	Optional character string recording publisher provenance.
<code>file_stem</code>	Character. Safe basename stem for dated release files. Defaults to <code>dataset_id</code> .

Details

`publish_dataset()` reads the draft, stages a byte-for-byte copy in `datasets_dir`, and reads the staged copy again before recording its SHA-256, row count, and column count. It supports the clinical-data formats read by `hvtiRutilities::read_clinical_data()`: `.sas7bdat`, `.csv`, `.xlsx`, `.xls`, and `.rds`. This verifies that the release is readable and records its shape. It does not decide whether the cohort or its values are clinically correct.

The first release on a date is named `<file_stem>_YYYYMMDD.<ext>`. A different release published for the same extract date adds `_r2`, `_r3`, and so on. The catalog lock protects that sequence when two processes publish at once. Publishing the same bytes again for the same dataset and date is idempotent: it returns the existing release rather than creating a revision. If that release was withdrawn, it remains withdrawn; publication does not reactivate the bytes under a new release identity.

The release file moves into place before the catalog is replaced. If the catalog write fails, the file remains as an unregistered orphan. No study can discover it. Retry the same call and publication registers the matching orphan without rewriting its bytes. Different bytes at an expected release filename are an integrity error and are never overwritten.

Value

Invisibly, a named list containing `release_id`, `sequence`, `file`, `extract_date`, `revision`, `published_at`, `sha256`, `n_rows`, `n_cols`, `status`, and `source` when supplied.

Examples

```
published <- tempfile("published-")
dir.create(published)
draft <- tempfile(fileext = ".csv")
write.csv(
  data.frame(synthetic_id = c("SYN001", "SYN002"), value = c(1, 2)),
  draft,
  row.names = FALSE
)
release <- publish_dataset(
  draft,
  dataset_id = "example_cohort",
  datasets_dir = published,
  extract_date = "2026-09-21",
  source = "Synthetic documentation example"
)
release$release_id
unlink(c(draft, published), recursive = TRUE)
```

Description

Reads the analysis set name written by `write_analysis_set()`, after checking that it is current. A set is stale when the built dataset has a newer registered version than the one it was cut from, or when its declaration in `_study.yml` has changed. A stale set is never rebuilt silently: its exclusions are decisions, and a changed attrition should be looked at. In a draft render it is read, with a message of class `hvtiRutilities_stale_analysis_set` giving the `write_analysis_set()` call that updates it. In a final render (HVTI_TEMPLATE_STRICT set, as `hvtiRtemplates::render_job(final = TRUE)` sets it) it stops with the same text. A parquet that no longer matches its manifest entry always stops.

When the built dataset is registered as a dated parquet, "changed" means a newer version has been registered with `hvtiRutilities::update_manifest()`. Rebuilding the source alone does not make a set stale, because nothing reads the rebuild until it is registered.

Usage

```
read_analysis_set(name, cfg = hvtiRutilities::study_config())
```

Arguments

<code>name</code>	Character(1). The set's name in <code>_study.yml</code> .
<code>cfg</code>	List. A study manifest from <code>hvtiRutilities::study_config()</code> .

Value

A data frame of the set's columns and rows, with the per-rule attrition table attached as `attr(x, "attrition")`.

See Also

`write_analysis_set()`

Examples

```
if (requireNamespace("arrow", quietly = TRUE)) {
  # A throwaway study in a temporary directory. Values are invented; no PHI.
  root <- tempfile("study")
  dir.create(root)
  suppressMessages(hvtiRutilities::study_setup(
    root, study = "Synthetic", study_tracker_id = 1L, adopt = TRUE,
    cleanup = FALSE, renv = FALSE
  ))
  built <- data.frame(
    ccfid = 1:6, age = c(15, 42, 55, 61, 70, 48),
    dead = c(0L, 1L, 0L, 1L, 0L, 0L)
  )
  utils::write.csv(
    built,
    file.path(hvtiRutilities::study_dir("datasets", root), "built.csv"),
    row.names = FALSE
  )
}
```

```

suppressMessages(hvtiRutilities::register_data(root, built = "built.csv"))
yaml <- file.path(root, "_study.yaml")
y <- yaml::read_yaml(yaml)
y$analysis_sets <- list(primary = list(
  id = "ccfid", vars = c("ccfid", "age", "dead"), event = "dead",
  exclude = list(list(reason = "Under 18", when = "age < 18"))
))
yaml::write_yaml(y, yaml)
cfg <- hvtiRutilities::study_config(root)
write_analysis_set("primary", cfg)
d <- read_analysis_set("primary", cfg)
print(dim(d))
print(attr(d, "attrition"))
unlink(root, recursive = TRUE)
}

```

read_master_config	<i>Read a master dataset's configuration</i>
--------------------	--

Description

A master dataset is a clinical dataset that other builds read, such as the cardiac-surgery master or the mitral master built on top of it. Each is described by a master.yaml kept outside the package, because it names paths on a shared volume. The file declares the master's view name, its one primary key, any alternate keys, its parent master, and where its SAS snapshots and build program live.

Usage

```
read_master_config(path)
```

Arguments

path	Path to the master.yaml file.
------	-------------------------------

Details

The primary key is what corrections match on and what the view joins on. Alternate keys are bridges to other systems: each is checked unique where it is non-null, and its columns are carried on every correction, but nothing matches on them.

Value

An object of class master_config: a list with name, key, alt_keys (a named list, possibly empty), parent (NULL or a list with master and libref), parent_release (NULL or a string), snapshots, current, history (NULL or a regular expression), build_program, and file.

See Also

`snapshot_master()`, `lift_master()`, `backfill_corrections()`

Examples

```
cfg <- read_master_config(system.file("extdata", "master-example.yml",
                                     package = "hvtiRdatabuild"))
cfg$key
```

read_study_config	<i>Read and validate a study configuration</i>
-------------------	--

Description

Loads the `study.yaml` that declares which warehouse modules a study pulls and which derivation steps it runs. The SAS templates expressed this as a catalogue of optional blocks, commented in or out; this makes it an explicit, diffable, committed file.

Usage

```
read_study_config(path)
```

Arguments

`path` Path to a `study.yaml` file.

Details

Validation is strict by design. An unknown key is an error rather than a warning, because a typo'd module name must not silently disable a module; that failure mode produces a quietly incomplete dataset with no signal. For the same reason, `modules` must name at least one module: an empty or bare `modules` key would otherwise pull zero data with no error, the same failure mode the SAS-era workflow of commenting out the last enabled block produces.

Value

An object of class `study_config`: a list with elements `study`, `cohort_table`, `warehouse`, `view_schema`, `pull_date` (a Date), `modules` (a non-empty character vector), `varsets`, and `derive` (a named logical vector).

Examples

```
path <- tempfile(fileext = ".yaml")
yaml::write_yaml(list(
  study = "st1234", cohort_table = "db.schema.st1234_cohort",
  warehouse = "warehouse", view_schema = "dbo",
  pull_date = "2026-08-04", modules = list("base"),
  varsets = list("core"), derive = list(missing = TRUE)
), path)
read_study_config(path)
```

snapshot_master	<i>Snapshot a master dataset to parquet, with its lineage</i>
-----------------	---

Description

Freezes a master's current SAS build, or its historical builds, as parquet with [snapshot_oracle\(\)](#), then checks its keys and records which release of its parent master it was built from.

Usage

```
snapshot_master(
  config,
  out_dir,
  which = c("current", "history"),
  chunk_rows = 1e+05,
  expect = NULL
)
```

Arguments

config	A master_config from read_master_config() .
out_dir	Directory to write the parquet snapshots and their sidecars.
which	"current", "history", or both.
chunk_rows	Rows per chunk, passed to snapshot_oracle() .
expect	Optional validation for the current build, passed to snapshot_oracle() . Historical builds are not validated, because their logs are rarely retained.

Details

The parent release is read from evidence of what ran, in this order: the log of the run that produced the dataset (a .log whose timestamp falls within six hours after the dataset's), where SAS records every dataset it read; then, for the current build only, the build program's set statements; then parent_release in the configuration. The log comes first because a program can be edited after the run. A current build whose parent cannot be decided stops; a historical one with no detected parent records "unknown". parent_release in the configuration, and the disagreement check against a

detected parent, apply only to the current build: a declared release describes the build as configured today, not necessarily what an older historical dataset actually read.

Only NOTE: lines naming datasets are read from a log, never data lines. Nothing printed carries a key or a value.

Value

Invisibly, a data frame with one row per dataset: file, status ("written", "skipped" or "failed"), n_rows, n_cols, sha256, source_sha256, parent_release, parent_source ("log", "program", "declared", "unknown" or "none") and key_verdict.

See Also

`read_master_config()`, `lift_master()`

Examples

```
if (requireNamespace("arrow", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE) &&
    requireNamespace("tidyselect", quietly = TRUE)) {
  dir <- tempfile("master")
  dir.create(dir)
  file.copy(system.file("extdata", "oracle_small.sas7bdat",
                        package = "hvtiRdatabuild"),
            file.path(dir, "built.sas7bdat"))
  writeLines("data m; run;", file.path(dir, "bd.data.sas"))
  cfg_path <- file.path(dir, "master.yml")
  writeLines(c("name: master_demo", "key: [ccfidu]",
              paste0("snapshots: ", dir), "current: built.sas7bdat",
              paste0("build_program: ", file.path(dir, "bd.data.sas"))),
            cfg_path)
  snapshot_master(read_master_config(cfg_path), tempfile("out"),
                 which = "current")
}
```

snapshot_oracle

Freeze a SAS-built dataset as a checksummed parquet oracle

Description

Reads a SAS dataset once with **haven**, writes it as parquet, and returns a record of what was written including a SHA-256 checksum of the parquet file.

Usage

```
snapshot_oracle(
  sas_path,
  out_path,
  expect = NULL,
  manifest = NULL,
  chunk_rows = NULL
)
```

Arguments

<code>sas_path</code>	Path to the SAS dataset (.sas7bdat).
<code>out_path</code>	Path to write the parquet file. Must not already exist.
<code>expect</code>	Optional named list validating the conversion, with any of <code>n_rows</code> , <code>n_cols</code> , and <code>variables</code> . Supply values read from SAS-side PROC CONTENTS. A mismatch is an error.
<code>manifest</code>	Optional path to a manifest YAML. When supplied, the snapshot is recorded with <code>hvtiRutilities::update_manifest()</code> so that <code>hvtiRutilities::verify_manifest()</code> can later detect a drifted oracle.
<code>chunk_rows</code>	Optional single positive number. When supplied, the SAS dataset is read and written this many rows at a time, as parquet row groups in one file, so a dataset too large to hold in memory can be snapshotted. A chunk whose schema differs from the first chunk's is an error, and the partial file is removed. The chunked file reads back identical to an unchunked one, but its bytes and checksum differ, because its row groups differ.

Details

The snapshot exists so that equivalence testing has a fixed reference. A SAS-built dataset on a shared volume can be regenerated at any time; if it changes mid-migration, every previously passing comparison silently becomes meaningless. A checksummed snapshot is a citable fixed point.

Note that this does not remove **haven** from the chain of custody; it confines it to a single audited step. A misread is faithfully preserved in the parquet file. Supply `expect` to validate the conversion against SAS-side PROC CONTENTS output.

Writing the sidecar needs **jsonlite**.

Value

Invisibly, a list with elements `path`, `n_rows`, `n_cols`, `variables`, `sha256` (of the parquet file), `source_sha256` (of the SAS dataset) and `meta_path`. The metadata sidecar at `meta_path` records both checksums, the shape, and each column's label, SAS format, SAS type and R class, so the metadata survives into systems that cannot read R's attributes.

See Also

[compare_built\(\)](#)

Examples

```
if (requireNamespace("arrow", quietly = TRUE)) {
  src <- system.file("extdata", "oracle_small.sas7bdat",
                    package = "hvtiRdatabuild")
  snapshot_oracle(src, tempfile(fileext = ".parquet"))
}
```

str.build_result	<i>Safely inspect a built dataset result</i>
------------------	--

Description

Safely inspect a built dataset result

Usage

```
## S3 method for class 'build_result'
str(object, ...)
```

Arguments

object	A build_result.
...	Ignored.

Value

NULL, invisibly.

str.built_comparison	<i>Structure of a dataset comparison, without identifiers</i>
----------------------	---

Description

str() dispatches on the object rather than routing through [print.built_comparison\(\)](#), so the default method would print the rows attribute verbatim, including the identifier vectors. In this group's data an identifier is a medical record number concatenated with a date of surgery.

Usage

```
## S3 method for class 'built_comparison'
str(object, ...)
```

Arguments

object	An object of class <code>built_comparison</code> .
...	Passed to the underlying <code>utils::str()</code> call.

Details

The hazard here is *incidental* disclosure, not retrieval. `str()` is what people type reflexively to inspect an object, and its output is routinely pasted into issues, chat, and logs, so PHI arrives without anyone having asked for it. Deliberate retrieval stays available and unchanged: `attr(x, "rows")$only_oracle` and `$only_r`.

Value

NULL, invisibly. Called for its side effect.

See Also

`print.built_comparison()`

str.cohort_record	<i>Safely inspect a cohort record</i>
-------------------	---------------------------------------

Description

Safely inspect a cohort record

Usage

```
## S3 method for class 'cohort_record'
str(object, ...)
```

Arguments

object	A <code>cohort_record</code> .
...	Ignored.

Value

NULL, invisibly.

str.derive_result	<i>Safely inspect a derived dataset result</i>
-------------------	--

Description

Safely inspect a derived dataset result

Usage

```
## S3 method for class 'derive_result'
str(object, ...)
```

Arguments

object	A derive_result.
...	Ignored.

Value

NULL, invisibly.

withdraw_dataset_release	<i>Withdraw a published dataset release</i>
--------------------------	---

Description

Withdrawal changes the release's catalog status and records why it should no longer be adopted. It never changes or deletes the release file. A study already pinned to that release can therefore reproduce historical work with the explicit `allow_withdrawn = TRUE` override in `hvtiRutilities::read_built()`, while normal reads stop and report the withdrawal.

Usage

```
withdraw_dataset_release(
  dataset_id,
  release_id,
  datasets_dir,
  reason,
  replacement_release_id = NULL
)
```

Arguments

<code>dataset_id</code>	Character. Stable logical dataset identifier.
<code>release_id</code>	Character. Exact release identifier to withdraw.
<code>datasets_dir</code>	Character. Existing directory that owns the published files and <code>dataset-catalog.yml</code> .
<code>reason</code>	Character. Non-empty reason for withdrawal.
<code>replacement_release_id</code>	Optional character release identifier from the same logical dataset.

Details

A replacement is optional, but it must name another release under the same logical dataset. Catalog locking serializes withdrawal with publication, so neither operation can discard the other's catalog change.

Value

Invisibly, the release record with `status = "withdrawn"`, `withdrawal_reason`, and `replacement_release_id` when supplied.

Examples

```
published <- tempfile("published-")
dir.create(published)
draft <- tempfile(fileext = ".csv")
write.csv(data.frame(synthetic_id = "SYN001", value = 1), draft,
          row.names = FALSE)
release <- publish_dataset(
  draft,
  "example_cohort",
  published,
  extract_date = "2026-09-21"
)
withdrawn <- withdraw_dataset_release(
  "example_cohort",
  release$release_id,
  published,
  reason = "Synthetic example correction"
)
withdrawn$status
file.exists(file.path(published, release$file))
unlink(c(draft, published), recursive = TRUE)
```

write_analysis_set	<i>Write an analysis set</i>
--------------------	------------------------------

Description

Cuts the analysis set name, declared under `analysis_sets`: in the study's `_study.yml`, from the built dataset: keeps its vars, applies its exclude rules in order (first match wins), checks any expect counts, and writes `<name>.parquet` and a `<name>.set.yml` sidecar in the study's logical datasets directory, plus a `manifest.yaml` entry. The sidecar records the parent dataset and the attrition.

Usage

```
write_analysis_set(name, cfg = hvtiRutilities::study_config())
```

Arguments

name	Character(1). The set's name in <code>_study.yml</code> .
cfg	List. A study manifest from <code>hvtiRutilities::study_config()</code> .

Details

Nothing is written unless every check passes. Each when is R code evaluated against the data with only base R visible; a missing value counts as not excluded, as SAS if `<missing>` then delete does. The identifier column named by `id` is used to track exclusions and is never written to the sidecar or the manifest. A set that names an event column, one of its vars, also counts `n_events` (rows where it equals 1) and `n_censored`; an expect on either count needs that event key.

Value

The sidecar contents, invisibly: a list with `set`, `parent`, `declaration_sha256`, `written`, `counts`, `n_cols`, `attrition` and `packages`.

Examples

```
if (requireNamespace("arrow", quietly = TRUE)) {
  # A throwaway study in a temporary directory. Values are invented; no PHI.
  root <- tempfile("study")
  dir.create(root)
  suppressMessages(hvtiRutilities::study_setup(
    root, study = "Synthetic", study_tracker_id = 1L, adopt = TRUE,
    cleanup = FALSE, renv = FALSE
  ))
  built <- data.frame(
    ccfid = 1:6, age = c(15, 42, 55, 61, 70, 48),
    dead = c(0L, 1L, 0L, 1L, 0L, 0L)
  )
  utils::write.csv(
```

```
    built,
    file.path(hvtiRutilities::study_dir("datasets", root), "built.csv"),
    row.names = FALSE
  )
  suppressMessages(hvtiRutilities::register_data(root, built = "built.csv"))
  yml <- file.path(root, "_study.yml")
  y <- yaml::read_yaml(yml)
  y$analysis_sets <- list(primary = list(
    id = "ccfid", vars = c("ccfid", "age", "dead"), event = "dead",
    exclude = list(list(reason = "Under 18", when = "age < 18"))
  ))
  yaml::write_yaml(y, yml)
  cfg <- hvtiRutilities::study_config(root)
  side <- write_analysis_set("primary", cfg)
  print(side$counts)
  unlink(root, recursive = TRUE)
}
```

Index

`append_cohort_stage`, [2](#)
`as.logical()`, [12](#)

`backfill_corrections`, [3](#)
`backfill_corrections()`, [9](#), [15](#), [20](#), [25](#)
`build_dataset`, [5](#)
`build_dataset()`, [10](#), [17](#)

`cohort_summary`, [6](#)
`compare_built`, [7](#)
`compare_built()`, [14](#), [17](#), [28](#)

`DBI::dbConnect()`, [12](#)
`DBI::DBIConnection`, [12](#), [14](#)
`decide_correction`, [8](#)
`decide_correction()`, [4](#), [19](#), [20](#)
`derive_vars`, [10](#)
`derive_vars()`, [18](#)
`dw_connect`, [11](#)
`dw_connect()`, [14](#), [15](#)
`dw_modules`, [13](#)
`dw_modules()`, [14](#)
`dw_pull`, [13](#)
`dw_pull()`, [5](#), [19](#)

`hvtiRutilities::adopt_data_update()`,
[21](#)
`hvtiRutilities::read_built()`, [31](#)
`hvtiRutilities::read_clinical_data()`,
[22](#)
`hvtiRutilities::study_config()`, [23](#), [33](#)
`hvtiRutilities::update_manifest()`, [23](#),
[28](#)
`hvtiRutilities::verify_manifest()`, [28](#)

`lift_master`, [14](#)
`lift_master()`, [4](#), [25](#), [27](#)

`new_cohort_record`, [16](#)
`new_cohort_record()`, [3](#)

`print.build_result`, [17](#)
`print.built_comparison`, [17](#)
`print.built_comparison()`, [29](#), [30](#)
`print.derive_result`, [18](#)
`print.pull_result`, [18](#)
`propose_correction`, [19](#)
`propose_correction()`, [4](#), [9](#)
`publish_dataset`, [21](#)

`read_analysis_set`, [22](#)
`read_master_config`, [24](#)
`read_master_config()`, [4](#), [8](#), [15](#), [19](#), [26](#), [27](#)
`read_study_config`, [25](#)
`read_study_config()`, [5](#), [10](#), [14](#)

`snapshot_master`, [26](#)
`snapshot_master()`, [15](#), [25](#)
`snapshot_oracle`, [27](#)
`snapshot_oracle()`, [7](#), [8](#), [26](#)
`str.build_result`, [29](#)
`str.built_comparison`, [29](#)
`str.cohort_record`, [30](#)
`str.derive_result`, [31](#)

`utils::str()`, [30](#)

`withdraw_dataset_release`, [31](#)
`write_analysis_set`, [33](#)
`write_analysis_set()`, [23](#)