Package 'ukbflow'

April 7, 2026

Title Streamlined Workflow for UK Biobank Data Extraction, Analysis, and Visualization

Version 0.3.3

Description Provides a streamlined workflow for UK Biobank cloud-based analysis on the Research Analysis Platform (RAP). Includes tools for phenotype extraction and decoding, variable derivation, survival and association analysis, genetic risk score computation, and publication-quality visualization. For details on the UK Biobank resource, see Bycroft et al. (2018) <doi:10.1038/s41586-018-0579-z>.

License MIT + file LICENSE

URL https://github.com/evanbio/ukbflow, https://evanbio.github.io/ukbflow/

BugReports https://github.com/evanbio/ukbflow/issues

Depends R (>= 4.1)

Imports broom, cli, curl, data.table, dplyr, forestploter, gt, gtsummary, jsonlite, processx, rlang, survival, tidyselect, tools

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, pagedown, pROC, rmarkdown, testthat (>= 3.0.0), mockery, webshot2, withr

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

NeedsCompilation no

Author Yibin Zhou [aut, cre] (ORCID: <https://orcid.org/0009-0009-4600-8175>)

Maintainer Yibin Zhou <evanzhou.bio@gmail.com>

Repository CRAN

Date/Publication 2026-04-07 07:40:02 UTC

Contents

`assoc_competing` *Fine-Gray competing risks association analysis*

Description

Fits a Fine-Gray subdistribution hazard model (via `survival::finegray()` + weighted `coxph()`) for each exposure variable and returns a tidy result table with subdistribution hazard ratios (SHR).

Usage

```
assoc_competing(
  data,
  outcome_col,
  time_col,
  exposure_col,
  compete_col = NULL,
  event_val = 1L,
  compete_val = 2L,
  covariates = NULL,
  base = TRUE,
  conf_level = 0.95
)

assoc_fg(
  data,
  outcome_col,
  time_col,
  exposure_col,
  compete_col = NULL,
  event_val = 1L,
  compete_val = 2L,
  covariates = NULL,
  base = TRUE,
```

```
  conf_level = 0.95
)
```

Arguments

data	(data.frame or data.table) Analysis dataset.
outcome_col	(character) Primary event column. In Mode A: a multi-value column (any integer or character codes). In Mode B: a 0/1 binary column.
time_col	(character) Follow-up time column (numeric, e.g. years).
exposure_col	(character) One or more exposure variable names.
compete_col	(character or NULL) Mode B only: name of the 0/1 competing event column. When NULL (default), Mode A is used.
event_val	Scalar value in outcome_col indicating the primary event (Mode A only). Default: 1L.
compete_val	Scalar value in outcome_col indicating the competing event (Mode A only). Default: 2L.
covariates	(character or NULL) Covariate names for the Fully adjusted model. When NULL, only Unadjusted (and Age/sex adjusted if base = TRUE) are run.
base	(logical) Whether to auto-detect age and sex columns and include an Age and sex adjusted model. Default: TRUE.
conf_level	(numeric) Confidence level for SHR intervals. Default: 0.95.

Details

Two input modes are supported depending on how the outcome is coded in your dataset:

Mode A - single multi-value column compete_col = NULL (default). outcome_col contains all event codes in one column (e.g. 0/1/2/3). Use event_val and compete_val to identify the event of interest and the competing event; all other values are treated as censored. Example: UKB censoring_type where 1 = event, 2 = death (competing), 0/3 = censored.

Mode B - dual binary columns compete_col is the name of a separate 0/1 column for the competing event. outcome_col is a 0/1 column for the primary event. When both are 1 for the same participant, the primary event takes priority. Example: outcome_col = "copd_status", compete_col = "death_status".

Internally both modes are converted to a three-level factor c("censor", "event", "compete") before being passed to finegray().

Three adjustment models are produced (where data allow):

- **Unadjusted** - always included.
- **Age and sex adjusted** - when base = TRUE and age/sex columns are detected.
- **Fully adjusted** - when covariates is non-NULL.

Value

A `data.table` with one row per exposure × term × model combination:

`exposure` Exposure variable name.

`term` Coefficient name as returned by `coxph`.

`model` Ordered factor: `Unadjusted` < `Age and sex adjusted` < `Fully adjusted`.

`n` Participants in the model (after NA removal).

`n_events` Primary events in the analysis set.

`n_compete` Competing events in the analysis set.

`SHR` Subdistribution hazard ratio.

`CI_lower` Lower CI bound.

`CI_upper` Upper CI bound.

`p_value` Robust z-test p-value from weighted Cox.

`SHR_label` Formatted string, e.g. `"1.23 (1.05-1.44)"`.

Examples

```
dt <- ops_toy(scenario = "association", n = 500)
dt <- dt[dm_timing != 1L & htn_timing != 1L]

res <- assoc_competing(
  data         = dt,
  outcome_col  = "dm_status",
  time_col     = "dm_followup_years",
  exposure_col = "p20116_i0",
  compete_col  = "htn_status",
  covariates   = c("bmi_cat", "tdi_cat")
)
```

`assoc_coxph` *Cox proportional hazards association analysis*

Description

Fits one or more Cox models for each exposure variable and returns a tidy result table suitable for downstream forest plots. By default, two standard adjustment models are always included alongside any user-specified model:

Usage

```
assoc_coxph(
  data,
  outcome_col,
  time_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  strata = NULL,
  conf_level = 0.95
)

assoc_cox(
  data,
  outcome_col,
  time_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  strata = NULL,
  conf_level = 0.95
)
```

Arguments

`data`	(data.frame or data.table) Analysis dataset. Must contain all columns referenced by `outcome_col`, `time_col`, and `exposure_col`.
`outcome_col`	(character) Name of the event indicator column. Accepts `logical` (`TRUE`/`FALSE`) or numeric/integer (`0`/`1`).
`time_col`	(character) Name of the follow-up time column (numeric, in consistent units, e.g. years).
`exposure_col`	(character) One or more exposure variable names. Each variable is analysed separately; results are stacked row-wise.
`covariates`	(character or NULL) Additional covariate column names for the **Fully adjusted** model (e.g. `c("tdi", "smoking", paste0("pc", 1:10))`). When `NULL` (default), the Fully adjusted model is not run.
`base`	(logical) If `TRUE` (default), always include the **Unadjusted** and **Age and sex adjusted** models in addition to any user-specified `covariates` model. Set to `FALSE` to run only the Fully adjusted model (requires `covariates` to be non-NULL).
`strata`	(character or NULL) Optional stratification variable. Passed to `survival::strata()` in the Cox formula.
`conf_level`	(numeric) Confidence level for hazard ratio intervals. Default: `0.95`.

Details

- **Unadjusted** - no covariates (crude).

- **Age and sex adjusted** - age + sex auto-detected from the data via UKB field IDs (21022 and 31). Skipped with a warning if either column cannot be found.
- **Fully adjusted** - the covariates supplied via the `covariates` argument. Only run when `covariates` is non-NULL.

Outcome coding: `outcome_col` may be `logical` (`TRUE`/`FALSE`) or integer/numeric (`0`/`1`). Logical values are converted to integer internally.

Exposure types supported:

- *Binary* - `0`/`1` or `TRUE`/`FALSE`; produces one `term` row per model.
- *Factor* - produces one `term` row per non-reference level.
- *Numeric* (continuous) - produces one `term` row per model.

Value

A `data.table` with one row per exposure × term × model combination, and the following columns:

`exposure` Exposure variable name.

`term` Coefficient name as returned by `coxph` (e.g. `"bmi_categoryObese"` for a factor, or the variable name itself for numeric/binary exposures).

`model` Ordered factor: `Unadjusted` < `Age and sex adjusted` < `Fully adjusted`.

`n` Number of participants included in the model (after `NA` removal).

`n_events` Number of events in the model's analysis set (after `NA` removal).

`person_years` Total person-years of follow-up in the model's analysis set (rounded, after `NA` removal).

`HR` Hazard ratio (point estimate).

`CI_lower` Lower bound of the confidence interval.

`CI_upper` Upper bound of the confidence interval.

`p_value` Wald test p-value.

`HR_label` Formatted string, e.g. `"1.23 (1.05-1.44)"`.

Examples

```
dt <- ops_toy(scenario = "association", n = 500)
dt <- dt[dm_timing != 1L]

res <- assoc_coxph(
  data         = dt,
  outcome_col  = "dm_status",
  time_col     = "dm_followup_years",
  exposure_col = "p20116_i0",
  covariates   = c("bmi_cat", "tdi_cat"),
  base         = FALSE
)

# Fully adjusted only (skip Unadjusted + Age-sex)
res <- assoc_coxph(
```

```
  data         = dt,
  outcome_col  = "dm_status",
  time_col     = "dm_followup_years",
  exposure_col = "p20116_i0",
  covariates   = c("p21022", "p31", "bmi_cat", "tdi_cat"),
  base         = FALSE
)
```

---

assoc_coxph_zph *Proportional hazards assumption test for Cox regression*

Description

Tests the proportional hazards (PH) assumption using Schoenfeld residuals via `cox.zph`. Re-fits the same models as `assoc_coxph` (same interface) and returns a tidy result table with term-level and global test statistics.

Usage

```
assoc_coxph_zph(
  data,
  outcome_col,
  time_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  strata = NULL
)

assoc_zph(
  data,
  outcome_col,
  time_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  strata = NULL
)
```

Arguments

`data`	(data.frame or data.table) Analysis dataset.
`outcome_col`	(character) Binary event indicator (`0`/`1` or `TRUE`/`FALSE`).
`time_col`	(character) Follow-up time column name.
`exposure_col`	(character) One or more exposure variable names.
`covariates`	(character or NULL) Covariates for the Fully adjusted model. Default: `NULL`.
`base`	(logical) Include Unadjusted and Age and sex adjusted models. Default: `TRUE`.
`strata`	(character or NULL) Optional stratification variable.

Details

A non-significant p-value ($p > 0.05$) indicates the PH assumption is satisfied for that term. The global test (`global_p`) reflects the overall PH assumption for the whole model.

Value

A `data.table` with one row per exposure × term × model combination, and columns:

`exposure` Exposure variable name.

`term` Coefficient name.

`model` Ordered factor: `Unadjusted` < `Age and sex adjusted` < `Fully adjusted`.

`chisq` Schoenfeld residual chi-squared statistic.

`df` Degrees of freedom.

`p_value` P-value for the PH test (term-level).

`ph_satisfied` Logical; `TRUE` if `p_value` > `0.05`.

`global_chisq` Global chi-squared for the whole model.

`global_df` Global degrees of freedom.

`global_p` Global p-value for the whole model.

Examples

```
dt <- ops_toy(scenario = "association", n = 500)
dt <- dt[dm_timing != 1L]

zph <- assoc_coxph_zph(
  data         = dt,
  outcome_col  = "dm_status",
  time_col     = "dm_followup_years",
  exposure_col = "p20116_i0",
  covariates   = c("bmi_cat", "tdi_cat"),
  base         = FALSE
)
zph[ph_satisfied == FALSE]
```

---

assoc_lag *Cox regression lag sensitivity analysis*

Description

Runs Cox proportional hazards models at one or more lag periods to assess whether associations are robust to the exclusion of early events. For each lag, participants whose follow-up time is less than `lag_years` are removed from the analysis dataset; follow-up time is kept on its original scale (not shifted). This mirrors the approach used in UK Biobank sensitivity analyses to address reverse causation and detection bias.

Usage

```
assoc_lag(
  data,
  outcome_col,
  time_col,
  exposure_col,
  lag_years = c(1, 2),
  covariates = NULL,
  base = TRUE,
  strata = NULL,
  conf_level = 0.95
)
```

Arguments

`data`	(data.frame or data.table) Analysis dataset.
`outcome_col`	(character) Event indicator column (0/1 or logical).
`time_col`	(character) Follow-up time column (numeric, e.g. years).
`exposure_col`	(character) One or more exposure variable names.
`lag_years`	(numeric) One or more lag periods in the same units as `time_col`. Default: `c(1, 2)`. Use `0` to include the unfiltered full-cohort result as a reference.
`covariates`	(character or NULL) Covariates for the Fully adjusted model. When `NULL`, only Unadjusted (and Age and sex adjusted if `base = TRUE`) are run.
`base`	(logical) Auto-detect age and sex and include an Age and sex adjusted model. Default: `TRUE`.
`strata`	(character or NULL) Optional stratification variable passed to `survival::strata()`.
`conf_level`	(numeric) Confidence level for HR intervals. Default: `0.95`.

Details

Setting `lag_years = 0` (or including `0` in the vector) runs the model on the full unfiltered cohort, providing a reference against which lagged results can be compared.

The same three adjustment models produced by `assoc_coxph` are available here (**Unadjusted**, **Age and sex adjusted**, **Fully adjusted**).

Value

A `data.table` with one row per lag × exposure × term × model combination, containing all columns produced by `assoc_coxph` plus:

`lag_years` The lag period applied (numeric).

`n_excluded` Number of participants excluded because their follow-up time was less than `lag_years`.

`lag_years` and `n_excluded` are placed immediately after `model` in the column order.

Examples

```
dt <- ops_toy(scenario = "association", n = 500)
dt <- dt[dm_timing != 1L]

res <- assoc_lag(
  data         = dt,
  outcome_col  = "dm_status",
  time_col     = "dm_followup_years",
  exposure_col = "p20116_i0",
  lag_years    = c(0, 1, 2),
  covariates   = c("bmi_cat", "tdi_cat"),
  base         = FALSE
)
res[, .(lag_years, n, n_excluded, HR, CI_lower, CI_upper, p_value)]
```

---

assoc_linear *Linear regression association analysis*

Description

Fits one or more linear regression models for each exposure variable and returns a tidy result table. By default, two standard adjustment models are always included:

Usage

```
assoc_linear(
  data,
  outcome_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  conf_level = 0.95
)

assoc_lm(
  data,
  outcome_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  conf_level = 0.95
)
```

Arguments

data	(data.frame or data.table) Analysis dataset.
outcome_col	(character) Name of the continuous numeric outcome column.

exposure_col	(character) One or more exposure variable names.
covariates	(character or NULL) Covariate column names for the **Fully adjusted** model. Default: NULL.
base	(logical) Include **Unadjusted** and **Age and sex adjusted** models. Default: TRUE.
conf_level	(numeric) Confidence level. Default: 0.95.

Details

- **Unadjusted** - no covariates (crude).
- **Age and sex adjusted** - age + sex auto-detected from the data via UKB field IDs (21022 and 31). Skipped with a warning if either column cannot be found.
- **Fully adjusted** - the covariates supplied via the covariates argument. Only run when covariates is non-NULL.

Outcome: intended for continuous numeric variables. Passing a binary (0/1) or logical column is permitted (linear probability model) but will trigger a warning recommending assoc_logistic instead.

CI method: based on the t-distribution via confint.lm(), which is exact under the normal linear model assumption. There is no ci_method argument (unlike assoc_logistic) as profile likelihood does not apply to lm.

SE column: the standard error of β is included to support downstream meta-analysis and GWAS-style summary statistics.

Value

A data.table with one row per exposure $\times$ term $\times$ model combination, and columns:

exposure Exposure variable name.

term Coefficient name (e.g. "bmi_categoryObese").

model Ordered factor: Unadjusted < Age and sex adjusted < Fully adjusted.

n Participants in model (after NA removal).

beta Regression coefficient (β).

se Standard error of β.

CI_lower Lower confidence bound.

CI_upper Upper confidence bound.

p_value t-test p-value.

beta_label Formatted string, e.g. "0.23 (0.05-0.41)".

Examples

```
dt <- ops_toy(scenario = "association", n = 500)

res <- assoc_linear(
  data         = dt,
  outcome_col  = "p21001_i0",
```

```
  exposure_col = "p20116_i0",
  covariates   = c("bmi_cat", "tdi_cat"),
  base         = FALSE
)
```

`assoc_logistic` *Logistic regression association analysis*

Description

Fits one or more logistic regression models for each exposure variable and returns a tidy result table suitable for downstream forest plots. By default, two standard adjustment models are always included:

Usage

```
assoc_logistic(
  data,
  outcome_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  ci_method = c("wald", "profile"),
  conf_level = 0.95
)

assoc_logit(
  data,
  outcome_col,
  exposure_col,
  covariates = NULL,
  base = TRUE,
  ci_method = c("wald", "profile"),
  conf_level = 0.95
)
```

Arguments

`data`	(data.frame or data.table) Analysis dataset.
`outcome_col`	(character) Binary outcome column (`0`/`1` or `TRUE`/`FALSE`).
`exposure_col`	(character) One or more exposure variable names.
`covariates`	(character or NULL) Covariate column names for the **Fully adjusted** model. Default: `NULL`.
`base`	(logical) Include **Unadjusted** and **Age and sex adjusted** models. Default: `TRUE`.
`ci_method`	(character) CI calculation method: `"wald"` (default) or `"profile"`.
`conf_level`	(numeric) Confidence level. Default: `0.95`.

Details

- **Unadjusted** - no covariates (crude).
- **Age and sex adjusted** - age + sex auto-detected from the data via UKB field IDs (21022 and 31). Skipped with a warning if either column cannot be found.
- **Fully adjusted** - the covariates supplied via the `covariates` argument. Only run when `covariates` is non-NULL.

Outcome coding: `outcome_col` may be `logical` (`TRUE`/`FALSE`) or integer/numeric (`0`/`1`). Logical values are converted to integer internally.

CI methods:

- `"wald"` (default) - fast, appropriate for large UKB samples.
- `"profile"` - profile likelihood CI via `confint.glm()`; slower but more accurate for small or sparse data.

Value

A `data.table` with one row per exposure × term × model combination, and columns:

`exposure` Exposure variable name.

`term` Coefficient name (e.g. `"bmi_categoryObese"`).

`model` Ordered factor: `Unadjusted` < `Age and sex adjusted` < `Fully adjusted`.

`n` Participants in model (after NA removal).

`n_cases` Number of cases (outcome = 1) in model.

`OR` Odds ratio (point estimate).

`CI_lower` Lower confidence bound.

`CI_upper` Upper confidence bound.

`p_value` Wald test p-value.

`OR_label` Formatted string, e.g. `"1.23 (1.05-1.44)"`.

Examples

```
dt <- ops_toy(scenario = "association", n = 500)

res <- assoc_logistic(
  data         = dt,
  outcome_col  = "dm_status",
  exposure_col = "p20116_i0",
  covariates   = c("bmi_cat", "tdi_cat"),
  base         = FALSE
)
```

assoc_subgroup *Subgroup association analysis with optional interaction test*

Description

Stratifies the dataset by a single grouping variable (by) and runs the specified association model (coxph, logistic, or linear) within each subgroup. Unlike assoc_coxph and its siblings, there is no automatic age-and-sex-adjusted model: at the subgroup level the variable that defines the stratum would have zero variance, making the auto-detected adjustment meaningless. Accordingly, this function does not accept a base argument. Instead, two models are available:

Usage

```
assoc_subgroup(
  data,
  outcome_col,
  time_col = NULL,
  exposure_col,
  by,
  method = c("coxph", "logistic", "linear"),
  covariates = NULL,
  interaction = TRUE,
  conf_level = 0.95
)

assoc_sub(
  data,
  outcome_col,
  time_col = NULL,
  exposure_col,
  by,
  method = c("coxph", "logistic", "linear"),
  covariates = NULL,
  interaction = TRUE,
  conf_level = 0.95
)
```

Arguments

data	(data.frame or data.table) Analysis dataset.
outcome_col	(character) Outcome column name.
time_col	(character or NULL) Follow-up time column (required when method = "coxph").
exposure_col	(character) One or more exposure variable names.
by	(character) Single stratification variable name. Its unique non-NA values (or factor levels, in order) define the subgroups. Should be a categorical or binary

variable with a small number of levels (e.g. sex, smoking status). Continuous variables are technically permitted and the interaction LRT will still run, but per-unique-value subgrouping is rarely meaningful in practice — use a pre-categorised version (e.g. via derive_cut) instead.

`method` (character) Regression method: `"coxph"` (default), `"logistic"`, or `"linear"`.

`covariates` (character or NULL) Covariate column names for the Fully adjusted model. When `NULL`, only the Unadjusted model is run.

`interaction` (logical) Compute the LRT p-value for the exposure × by interaction on the full dataset. Default: `TRUE`.

`conf_level` (numeric) Confidence level. Default: `0.95`.

Details

- **Unadjusted** - always run (no covariates).
- **Fully adjusted** - run when `covariates` is non-NULL. Users are responsible for excluding the by variable from `covariates` (a warning is issued if it is included).

Interaction test: when `interaction = TRUE` (default), a likelihood ratio test (LRT) for the exposure × by interaction is computed on the *full* dataset for each exposure × model combination and appended as `p_interaction`. LRT is preferred over Wald because it handles factor, binary, and continuous by variables uniformly without requiring the user to recode the by variable.

Value

A `data.table` with one row per subgroup level × exposure × term × model, containing:

`subgroup` Name of the by variable.

`subgroup_level` Factor: level of the by variable (ordered by original level / sort order).

`exposure` Exposure variable name.

`term` Coefficient name from the fitted model.

`model` Ordered factor: `Unadjusted` < `Fully adjusted`.

`n` Participants in model (after NA removal).

... Effect estimate columns: `HR`/`OR`/`beta`, `CI_lower`, `CI_upper`, `p_value`, and a formatted label column. Cox models additionally include `n_events` and `person_years`; logistic models include `n_cases`; linear models include `se`.

`p_interaction` LRT p-value for the exposure × by interaction on the full dataset. Shared across all subgroup levels for the same exposure × model. `NA` when the interaction model fails. Only present when `interaction = TRUE`.

Examples

```
dt <- ops_toy(scenario = "association", n = 500)
dt <- dt[dm_timing != 1L]

res <- assoc_subgroup(
  data         = dt,
  outcome_col  = "dm_status",
```

```
  time_col     = "dm_followup_years",
  exposure_col = "p20116_i0",
  by           = "p31",
  method       = "coxph",
  covariates   = c("bmi_cat", "tdi_cat")
)
```

`assoc_trend` *Dose-response trend analysis*

Description

Fits categorical and trend models simultaneously for each ordered-factor exposure, returning per-category effect estimates alongside a p-value for linear trend. Two models are run internally per exposure × adjustment combination:

Usage

```
assoc_trend(
  data,
  outcome_col,
  time_col = NULL,
  exposure_col,
  method = c("coxph", "logistic", "linear"),
  covariates = NULL,
  base = TRUE,
  scores = NULL,
  conf_level = 0.95
)

assoc_tr(
  data,
  outcome_col,
  time_col = NULL,
  exposure_col,
  method = c("coxph", "logistic", "linear"),
  covariates = NULL,
  base = TRUE,
  scores = NULL,
  conf_level = 0.95
)
```

Arguments

`data`	(data.frame or data.table) Analysis dataset.
`outcome_col`	(character) Outcome column name.
`time_col`	(character or NULL) Follow-up time column (required when `method = "coxph"`).

exposure_col	(character) One or more exposure column names. Each must be a factor (ordered or unordered). The first level is used as the reference group.
method	(character) Regression method: "coxph" (default), "logistic", or "linear".
covariates	(character or NULL) Covariates for the Fully adjusted model. Default: NULL.
base	(logical) Include Unadjusted and Age-and-sex-adjusted models. Default: TRUE.
scores	(numeric or NULL) Numeric scores assigned to factor levels in level order. Length must equal nlevels of every exposure. NULL (default) uses 0, 1, 2, ...
conf_level	(numeric) Confidence level. Default: 0.95.

Details

1. **Categorical model** - exposure treated as a factor; produces one row per non-reference level (HR / OR / β vs reference).
2. **Trend model** - exposure recoded as numeric scores (default: 0, 1, 2, ...); produces the per-score-unit effect estimate (*_per_score) and p_trend.

Both results are merged: the output contains a reference row (effect = 1 / 0, CI = NA) followed by non-reference rows, with *_per_score and p_trend appended as shared columns (same value within each exposure $\times$ model combination).

Scores: By default levels are scored 0, 1, 2, ... so the reference group = 0 and each step = 1 unit. Supply scores to use meaningful units (e.g. median years per category) - only p_trend and the per-score estimate change; per-category HRs are unaffected.

Adjustment models: follows the same logic as assoc_coxph - Unadjusted and Age-and-sex-adjusted models are included by default (base = TRUE); a Fully adjusted model is added when covariates is non-NULL.

Value

A data.table with one row per exposure $\times$ level $\times$ model, containing:

exposure Exposure variable name.

level Factor level (ordered factor preserving original level order).

term Coefficient name as returned by the model (reference row uses paste0(exposure, ref_level)).

model Ordered factor: Unadjusted < Age and sex adjusted < Fully adjusted.

n Participants in model (after NA removal).

... Effect estimate columns from the categorical model: HR/OR/beta, CI_lower, CI_upper, p_value, and a formatted label. Reference row has HR = 1 / beta = 0 and NA for CI and p. Cox models additionally include n_events and person_years; logistic includes n_cases; linear includes se.

HR_per_score / OR_per_score / beta_per_score Per-score-unit effect estimate from the trend model. Shared across all levels within the same exposure $\times$ model.

HR_per_score_label / OR_per_score_label / beta_per_score_label Formatted string for the per-score estimate.

p_trend P-value for linear trend from the trend model. Shared across all levels within the same exposure $\times$ model.

Examples

```
dt <- ops_toy(scenario = "association", n = 500)
dt <- dt[dm_timing != 1L]

res <- assoc_trend(
  data         = dt,
  outcome_col  = "dm_status",
  time_col     = "dm_followup_years",
  exposure_col = "bmi_cat",
  method       = "coxph",
  covariates   = c("tdi_cat", "p20116_i0"),
  base         = FALSE
)
```

---

auth_list_projects *List available DNAnexus projects*

Description

Returns a list of all projects accessible to the current user.

Usage

```
auth_list_projects()
```

Value

A character vector of project names and IDs.

Examples

```
## Not run:
auth_list_projects()

## End(Not run)
```

---

auth_login *Login to DNAnexus with a token*

Description

Authenticates with the DNAnexus Research Analysis Platform using an API token. Equivalent to running `dx login --token` on the command line.

Usage

```
auth_login(token = NULL)
```

Arguments

token	(character) DNAnexus API token. If NULL, reads from the environment variable DX_API_TOKEN.

Value

Invisible TRUE on success.

Examples

```
## Not run:
# Supply token directly
auth_login(token = "your_token_here")

# Or store token in ~/.Renviron (recommended):
# usethis::edit_r_environ()
# Add: DX_API_TOKEN=your_token_here
# Save and restart R, then call:
auth_login()

## End(Not run)
```

---

auth_logout *Logout from DNAnexus*

Description

Invalidates the current DNAnexus session on the remote platform. The local token file is not removed but becomes invalid. A new token must be generated from the DNAnexus platform before calling auth_login() again.

Usage

```
auth_logout()
```

Value

Invisible TRUE on success.

Examples

```
## Not run:
auth_logout()

## End(Not run)
```

auth_select_project *Select a DNAnexus project*

Description

Switches the active project context on the DNAnexus platform. Only project IDs (e.g. `"project-XXXXXXXXXXXX"`) are accepted. Run `auth_list_projects()` to find your project ID.

Usage

```
auth_select_project(project)
```

Arguments

`project`	(character) Project ID in the form `"project-XXXXXXXXXXXX"`.

Value

Invisible TRUE on success.

Examples

```
## Not run:
auth_select_project("project-XXXXXXXXXXXX")

## End(Not run)
```

auth_status *Check current DNAnexus authentication status*

Description

Returns the current logged-in user and selected project.

Usage

```
auth_status()
```

Value

A named list with `user` and `project`.

Examples

```
## Not run:
auth_status()

## End(Not run)
```

decode_names *Rename UKB field ID columns to human-readable snake_case names*

Description

Renames columns from the raw UKB field ID format used by `extract_pheno` (e.g. `participant.p31`) and `job_result` (e.g. `p53_i0`) to human-readable snake_case identifiers (e.g. `sex`, `date_of_attending_assessment_centr`

Usage

```
decode_names(data, max_nchar = 60L)
```

Arguments

`data`	(data.frame or data.table) Data extracted from UKB-RAP via `extract_pheno()` or `job_result()`.
`max_nchar`	(integer) Column names longer than this value are flagged. Default: `60`.

Details

Column labels are taken from the UKB field title dictionary via `extract_ls`. Both the dataset name and field list are cached after the first call, so subsequent calls to `decode_names()` involve no network requests.

When an auto-generated name exceeds `max_nchar` characters it is flagged with a warning so you can decide whether to shorten it manually with `names(data)[...] <- ...`. The function never truncates names automatically, because the right short name depends on scientific context that only you know.

Value

The input `data` with column names replaced by snake_case labels. Returns a `data.table` if the input is a `data.table`.

Examples

```
## Not run:
df <- extract_pheno(c(31, 53, 21022))
df <- decode_names(df)
# participant.eid    → eid
# participant.p31    → sex
# participant.p21022 → age_at_recruitment
# participant.p53_i0 → date_of_attending_assessment_centre_i0  (warned if > 60)

# Shorten a long name afterwards
names(df)[names(df) == "date_of_attending_assessment_centre_i0"] <- "date_baseline"

## End(Not run)
```

decode_values *Decode UKB categorical column values using Showcase metadata*

Description

Converts raw integer codes produced by extract_pheno into human-readable labels for all categorical fields (value_type 21 and 22), using the UKB Showcase encoding tables. Continuous, text, date, and already-decoded columns are left unchanged.

Usage

```
decode_values(data, metadata_dir = "data/metadata/")
```

Arguments

data	(data.frame or data.table) Data from extract_pheno(), with column names in participant.pXXXX or pXXXX_iX format.
metadata_dir	(character) Directory containing field.tsv and esimpint.tsv. Default: "data/metadata/".

Details

This function requires two metadata files downloaded from the UKB Research Analysis Platform:

- field.tsv - maps field IDs to encoding IDs and value types.
- esimpint.tsv - maps encoding ID + integer code to label.

Download them once with:

```
fetch_metadata(dest_dir = "data/metadata")
```

Both files are cached in the session after the first read.

Call order: use decode_values() *before* decode_names, so that column names still contain the numeric field ID needed to look up the encoding.

Value

The input data with categorical columns replaced by character labels. Returns a data.table if the input is a data.table.

Examples

```
## Not run:
# Download metadata once
fetch_metadata(dest_dir = "data/metadata")

# Recommended call order
df <- extract_pheno(c(31, 54, 20116, 21000))
df <- decode_values(df)                 # 0/1 → "Female"/"Male", etc.
```

```
df <- decode_names(df)                  # participant.p31 → sex

## End(Not run)
```

`derive_age` *Compute age at event for one or more UKB outcomes*

Description

For each name in `name`, adds one column `age_at_{name}` (numeric, years) computed as:

$$age\_at\_event = age\_col + (event\_date - baseline\_date)/365.25$$

Usage

```
derive_age(
  data,
  name,
  baseline_col,
  age_col,
  date_cols = NULL,
  status_cols = NULL
)
```

Arguments

`data`	(data.frame or data.table) UKB phenotype data.
`name`	(character) One or more output prefixes, e.g. `c("disease", "disease_icd10", "outcome")`. Each produces `age_at_{name}`.
`baseline_col`	(character) Name of the baseline date column (e.g. `"date_baseline"`).
`age_col`	(character) Name of the age-at-baseline column (e.g. `"age_recruitment"`).
`date_cols`	(character or NULL) Named character vector mapping each name to its event date column, e.g. `c(disease = "disease_date", outcome = "outcome_date")`. `NULL` (default) triggers auto-detection as `{name}_date`.
`status_cols`	(character or NULL) Named character vector mapping each name to its status column. `NULL` (default) triggers auto-detection.

Details

The value is `NA` for participants who did not experience the event (status is `FALSE` / `0`) or who lack an event date.

Auto-detection per name (when `date_cols` / `status_cols` are `NULL`):

- `date_col` - looked up as `{name}_date`.
- `status_col` - looked up first as `{name}_status`, then as `{name}` (logical column); if neither exists all rows with a non-`NA` date are treated as cases.

data.table pass-by-reference: new columns are added in-place.

Value

The input `data` with one new `age_at_{name}` column per entry in `name`, added in-place.

Examples

```
dt <- ops_toy(scenario = "association", n = 100)
derive_age(dt, name = "dm", baseline_col = "p53_i0", age_col = "p21022")
```

`derive_cancer_registry` *Derive a binary disease flag from UKB cancer registry*

Description

The UK Biobank cancer registry links each participant's cancer diagnoses from national cancer registries. Each diagnosis is stored as a separate instance with four parallel fields: ICD-10 code (`p40006`), histology code (`p40011`), behaviour code (`p40012`), and diagnosis date (`p40005`). Unlike HES or self-report data, each instance holds exactly one record - there is no array (`a*`) dimension.

Usage

```
derive_cancer_registry(
  data,
  name,
  icd10 = NULL,
  histology = NULL,
  behaviour = NULL,
  code_cols = NULL,
  hist_cols = NULL,
  behv_cols = NULL,
  date_cols = NULL
)
```

Arguments

`data`	(data.frame or data.table) UKB phenotype data containing cancer registry fields.
`name`	(character) Output column prefix, e.g. `"outcome_invasive"` produces `outcome_invasive_cancer` and `outcome_invasive_cancer_date`.
`icd10`	(character or NULL) Regular expression matched against the ICD-10 code column (`p40006`). `NULL` = no ICD-10 filter. Examples: `"^C44"`, `"^(C44|D04)"`.
`histology`	(integer vector or NULL) Histology codes to retain (`p40011`). `NULL` = no histology filter. Example: `c(8070:8078, 8083, 8084)`.
`behaviour`	(integer vector or NULL) Behaviour codes to retain (`p40012`). `NULL` = no behaviour filter. Typical values: `3L` (invasive / malignant), `2L` (in situ).
`code_cols`	(character or NULL) Names of ICD-10 code columns (`p40006_i*`). `NULL` = auto-detect via field 40006.

hist_cols	(character or NULL) Names of histology columns (p40011_i*). NULL = auto-detect via field 40011.
behv_cols	(character or NULL) Names of behaviour columns (p40012_i*). NULL = auto-detect via field 40012.
date_cols	(character or NULL) Names of diagnosis date columns (p40005_i*). NULL = auto-detect via field 40005.

Details

All three filter arguments (icd10, histology, behaviour) are applied with AND logic: a record must satisfy every non-NULL filter to be counted. For OR conditions (e.g.\ D04 *or* C44 with specific histology), call the function twice and combine the resulting columns downstream.

{name}_cancer Logical flag: TRUE if any cancer registry record satisfies all supplied filters.

{name}_cancer_date Earliest matching diagnosis date (IDate).

Value

The input data with two new columns added in-place: {name}_cancer (logical) and {name}_cancer_date (IDate). Always returns a data.table.

Examples

```
dt <- ops_toy(n = 100)
derive_cancer_registry(dt, name = "skin",     icd10 = "^C44")
derive_cancer_registry(dt, name = "invasive", icd10 = "^C", behaviour = 3L)
```

---

derive_case *Combine self-report and ICD-10 sources into a unified case definition*

Description

Takes the self-report flag and ICD-10 flag produced by derive_selfreport and derive_icd10 (or any pair of logical columns) and merges them into a single logical case status and earliest date.

Usage

```
derive_case(
  data,
  name,
  icd10_col = NULL,
  selfreport_col = NULL,
  icd10_date_col = NULL,
  selfreport_date_col = NULL
)
```

Arguments

data	(data.frame or data.table) UKB phenotype data.
name	(character) Column prefix used both to locate the default input columns and to name the output columns. Defaults: {name}_icd10, {name}_selfreport, {name}_icd10_date, {name}_selfreport_date.
icd10_col	(character or NULL) Name of the ICD-10 status column. NULL = paste0(name, "_icd10").
selfreport_col	(character or NULL) Name of the self-report status column. NULL = paste0(name, "_selfreport").
icd10_date_col	(character or NULL) Name of the ICD-10 date column. NULL = paste0(name, "_icd10_date").
selfreport_date_col	(character or NULL) Name of the self-report date column. NULL = paste0(name, "_selfreport_date").

Details

{name}_status Logical: TRUE if positive in at least one source.

{name}_date Earliest diagnosis/report date across both sources (IDate).

Value

The input data with two new columns added in-place: {name}_status (logical) and {name}_date (IDate). Always returns a data.table.

Examples

```
dt <- ops_toy(n = 100)
derive_selfreport(dt, name = "htn", regex = "hypertension",
                  field        = "noncancer",
                  disease_cols = paste0("p20002_i0_a", 0:4),
                  date_cols    = paste0("p20008_i0_a", 0:4),
                  visit_cols   = "p53_i0")
derive_icd10(dt, name = "htn", icd10 = "I10", source = c("hes", "death"))
derive_case(dt, name = "htn")
```

---

derive_covariate *Prepare UKB covariates for analysis*

Description

Converts decoded UKB columns to analysis-ready types: character-encoded numeric fields to numeric, and categorical fields to factor. Prints a concise summary for each converted column - mean / median / SD / missing rate for numeric columns, and level counts for factor columns - so you can verify distributions without leaving the pipeline.

Usage

```
derive_covariate(
  data,
  as_numeric = NULL,
  as_factor = NULL,
  factor_levels = NULL,
  max_levels = 5L
)
```

Arguments

data	(data.frame or data.table) UKB data, typically output of derive_missing.
as_numeric	(character or NULL) Column names to convert to numeric. Values that cannot be coerced (e.g. residual text) become NA with a warning. Default: NULL.
as_factor	(character or NULL) Column names to convert to factor. Default levels are the sorted unique non-NA values unless overridden by factor_levels. Default: NULL.
factor_levels	(named list or NULL) Custom level ordering for specific factor columns. Names must match entries in as_factor; values are character vectors of levels in the desired order (first level = reference group in regression). Columns not listed use default ordering. Default: NULL.
max_levels	(integer) Factor columns with more levels than this threshold trigger a warning suggesting the user consider collapsing categories. Default: 5L.

Details

data.table pass-by-reference: when the input is a data.table, modifications are made in-place. Pass data.table::copy(data) to preserve the original.

Value

The input data with converted columns. Always returns a data.table.

Examples

```
dt <- ops_toy(n = 100)
dt <- derive_missing(dt)
# messy_label is a character column with numeric-looking values ("-1", "999")
# mixed with non-parseable strings — demonstrates coercion and NA warnings
derive_covariate(dt,
  as_numeric = "messy_label",
  as_factor  = c("p31", "p20116_i0"),
  factor_levels = list(
    p20116_i0 = c("Never", "Previous", "Current")
  )
)
```

`derive_cut` *Cut a continuous UKB variable into quantile-based or custom groups*

Description

Creates a new factor column by binning a continuous variable into `n` groups. When `breaks` is omitted, group boundaries are derived from quantiles of the observed data (equal-frequency binning). When `breaks` is supplied, those values are used as interior cut points.

Usage

```
derive_cut(data, col, n, breaks = NULL, labels = NULL, name = NULL)
```

Arguments

`data`	(data.frame or data.table) UKB data.
`col`	(character) Name of the source numeric column.
`n`	(integer) Number of groups. Any integer `>= 2` is accepted. Commonly used values are `2`, `3`, `4`, `5`.
`breaks`	(numeric vector or NULL) Interior cut points; length must equal `n - 1`. When `NULL` (default), quantile-based equal- frequency boundaries are computed automatically.
`labels`	(character vector or NULL) Group labels of length `n`. Defaults to `"Q1"`, `"Q2"`, ..., `"Qn"`.
`name`	(character or NULL) Name for the new column. Defaults to `"{col}_bi"` / `"{col}_tri"` / `"{col}_quad"` / `"{col}_quin"` for `n` = 2 / 3 / 4 / 5; other values of `n` produce `"{col}_g{n}"`.

Details

Before binning, a numeric summary (mean, median, SD, Q1, Q3, missing rate) is printed for the source column. After binning, the group distribution is printed via an internal summary helper.

Only one column can be processed per call; loop over columns explicitly when binning multiple variables.

data.table pass-by-reference: when the input is a `data.table`, the new column is added in-place. Pass `data.table::copy(data)` to preserve the original.

Value

The input `data` with one new factor column appended. Always returns a `data.table`.

Examples

```
dt <- ops_toy(n = 100)
derive_cut(dt, col = "p21001_i0", n = 3)

# Custom breaks and labels
derive_cut(dt, col = "p21001_i0", n = 3,
           breaks = c(25, 30),
           labels = c("<25", "25-30", "30+"),
           name   = "bmi_group")
```

`derive_death_registry` *Derive a binary disease flag from UKB death registry*

Description

Death registry records store the underlying (primary) cause of death in field `p40001` and contributory (secondary) causes in field `p40002`, both coded in ICD-10. The date of death is in field `p40000`. All three fields have an instance dimension (`i0`, `i1`) reflecting potential amendments; `p40002` additionally has an array dimension (`a0`, `a1`, ...).

Usage

```
derive_death_registry(
  data,
  name,
  icd10,
  match = c("prefix", "exact", "regex"),
  primary_cols = NULL,
  secondary_cols = NULL,
  date_cols = NULL
)
```

Arguments

`data`	(data.frame or data.table) UKB phenotype data containing death registry fields.
`name`	(character) Output column prefix, e.g. `"disease"` produces `disease_death` and `disease_death_date`.
`icd10`	(character) ICD-10 code(s) to match. For `"prefix"` and `"exact"`, supply a vector such as `c("L20", "L21")`. For `"regex"`, supply a single regex string.
`match`	(character) Matching strategy: `"prefix"` (default), `"exact"`, or `"regex"`.
`primary_cols`	(character or NULL) Names of primary cause columns (`p40001_i*`). `NULL` = auto-detect via field 40001.
`secondary_cols`	(character or NULL) Names of secondary cause columns (`p40002_i*_a*`). `NULL` = auto-detect via field 40002.
`date_cols`	(character or NULL) Names of death date columns (`p40000_i*`). `NULL` = auto-detect via field 40000.

Details

{name}_death Logical flag: TRUE if any death registry record (primary or secondary cause) contains a matching ICD-10 code.

{name}_death_date Earliest death date across matching instances (IDate). Note: this is the *date of death*, not onset date.

Value

The input data with two new columns added in-place: {name}_death (logical) and {name}_death_date (IDate). Always returns a data.table.

Examples

```
dt <- ops_toy(n = 100)
derive_death_registry(dt, name = "mi",   icd10 = "I21")
derive_death_registry(dt, name = "lung", icd10 = "C34")
```

---

derive_first_occurrence

Derive a binary disease flag from UKB First Occurrence fields

---

Description

UKB pre-computes the earliest recorded date for hundreds of ICD-10 chapters and categories as *First Occurrence* fields (p131xxx). Each field contains a single date per participant; no array or instance depth is involved. This function reads that date, converts it to IDate, and writes two analysis-ready columns:

Usage

```
derive_first_occurrence(data, name, field, col = NULL)
```

Arguments

data	(data.frame or data.table) UKB phenotype data.
name	(character) Output column prefix, e.g. "disease" produces disease_fo and disease_fo_date.
field	(integer or character) UKB field ID of the First Occurrence field, e.g. 131666 for E11 (type 2 diabetes).
col	(character or NULL) Name of the source column in data. When NULL (default) the column is detected automatically from field.

Details

{name}_fo_date Earliest First Occurrence date (IDate). Values that cannot be coerced to a valid date (e.g. UKB error codes) are silently set to NA.

{name}_fo Logical flag derived from {name}_fo_date: TRUE if and only if a valid date exists. This guarantees that every positive case has a usable date - essential for time-to-event and prevalent/incident classification.

Column detection: the function locates the source column automatically from field, handling both the raw format used by extract_pheno (participant.p131666) and the snake_case format produced by decode_names (date_e11_first_reported_type_2_diabetes). Supply col to override auto-detection.

data.table pass-by-reference: when the input is a data.table, new columns are added in-place via :=. The returned object and the original variable point to the same memory.

Value

The input data (invisibly) with two new columns added in-place: {name}_fo (logical) and {name}_fo_date (IDate).

Examples

```
dt <- ops_toy(n = 100)
derive_first_occurrence(dt, name = "outcome", field = 131742L, col = "p131742")
```

---

derive_followup *Compute follow-up end date and follow-up time for survival analysis*

Description

Adds two columns to data:

- {name}_followup_end (IDate) - the earliest of the outcome event date, death date, lost-to-follow-up date, and the administrative censoring date.
- {name}_followup_years (numeric) - time in years from baseline_col to {name}_followup_end.

Usage

```
derive_followup(
  data,
  name,
  event_col,
  baseline_col,
  censor_date,
  death_col = NULL,
  lost_col = NULL
)
```

Arguments

data	(data.frame or data.table) UKB phenotype data.
name	(character) Output column prefix, e.g. "outcome" produces outcome_followup_end and outcome_followup_years.
event_col	(character) Name of the outcome event date column (e.g. "outcome_date").
baseline_col	(character) Name of the baseline date column (e.g. "date_baseline").
censor_date	(Date or character) Scalar administrative censoring date, e.g. as.Date("2022-06-01"). A character string in "YYYY-MM-DD" format is also accepted.
death_col	(character or NULL) Name of the death date column (UKB field 40000). NULL (default) triggers auto-detection via the extract_ls cache; pass FALSE to explicitly disable death as a competing end-point.
lost_col	(character or NULL) Name of the lost-to-follow-up date column (UKB field 191). NULL (default) triggers auto-detection; pass FALSE to explicitly disable.

Details

data.table pass-by-reference: when the input is a data.table, new columns are added in-place via :=.

Value

The input data with two new columns added in-place: {name}_followup_end (IDate) and {name}_followup_years (numeric).

Examples

```
dt <- ops_toy(n = 100)
derive_hes(dt, name = "htn", icd10 = "I10")
derive_followup(dt,
  name         = "htn_hes",
  event_col    = "htn_hes_date",
  baseline_col = "p53_i0",
  censor_date  = as.Date("2022-06-01"),
  death_col    = "p40000_i0",
  lost_col     = FALSE)
```

---

derive_hes *Derive a binary disease flag from UKB HES inpatient diagnoses*

Description

Hospital Episode Statistics (HES) inpatient records store ICD-10 diagnosis codes in field p41270 (single JSON-array column on UKB RAP) and corresponding first-diagnosis dates in field p41280 (p41280_a0, p41280_a1, ...). The array index in p41270 and p41280 are aligned: the N-th code in the JSON array corresponds to p41280_aN (date of first in-patient diagnosis for that code).

Usage

```
derive_hes(
  data,
  name,
  icd10,
  match = c("prefix", "exact", "regex"),
  disease_cols = NULL,
  date_cols = NULL
)
```

Arguments

data	(data.frame or data.table) UKB phenotype data containing HES fields (p41270 and p41280_a*).
name	(character) Output column prefix, e.g. "disease" produces disease_hes and disease_hes_date.
icd10	(character) ICD-10 code(s) to match. For "prefix" and "exact", supply a vector such as c("L20", "L21"). For "regex", supply a single regex string.
match	(character) Matching strategy: "prefix" (default) matches any code starting with the supplied string; "exact" requires a full match; "regex" uses icd10 directly.
disease_cols	(character or NULL) Name of the p41270 column. NULL = auto-detect.
date_cols	(character or NULL) Names of p41280_a* columns. NULL = auto-detect.

Details

{name}_hes Logical flag: TRUE if any HES record contains a matching ICD-10 code.

{name}_hes_date Earliest first-diagnosis date across all matching codes (IDate). NA if no date is available.

Value

The input data with two new columns added in-place: {name}_hes (logical) and {name}_hes_date (IDate). Always returns a data.table.

Examples

```
dt <- ops_toy(n = 100)
derive_hes(dt, name = "htn",      icd10 = "I10")
derive_hes(dt, name = "diabetes", icd10 = c("E10", "E11"))
derive_hes(dt, name = "asthma",   icd10 = "^J4", match = "regex")
```

`derive_icd10` *Derive a unified ICD-10 disease flag across multiple UKB data sources*

Description

A high-level wrapper that calls one or more of `derive_hes`, `derive_death_registry`, `derive_first_occurrence`, and `derive_cancer_registry` according to the `source` argument, then combines their results into a single status flag and earliest-date column.

Usage

```
derive_icd10(
  data,
  name,
  icd10,
  source = c("hes", "death", "first_occurrence", "cancer_registry"),
  match = c("prefix", "exact", "regex"),
  fo_field = NULL,
  fo_col = NULL,
  histology = NULL,
  behaviour = NULL,
  hes_code_col = NULL,
  hes_date_cols = NULL,
  primary_cols = NULL,
  secondary_cols = NULL,
  death_date_cols = NULL,
  cr_code_cols = NULL,
  cr_hist_cols = NULL,
  cr_behv_cols = NULL,
  cr_date_cols = NULL
)
```

Arguments

`data`	(data.frame or data.table) UKB phenotype data.
`name`	(character) Output column prefix, e.g. `"disease"` produces `disease_icd10` and `disease_icd10_date`, plus intermediate columns such as `disease_hes`, `disease_hes_date`, etc.
`icd10`	(character) ICD-10 code(s) to match. For `"prefix"` and `"exact"`, supply a vector such as `c("L20", "L21")`. For `"regex"`, supply a single regex string. When `"cancer_registry"` is included in `source`, `icd10` and `match` are automatically converted to a regex and passed to `derive_cancer_registry`.
`source`	(character) One or more of `"hes"`, `"death"`, `"first_occurrence"`, `"cancer_registry"`. Defaults to all four.

match	(character) Matching strategy passed to derive_hes and derive_death_registry: "prefix" (default), "exact", or "regex".
fo_field	(integer or character or NULL) UKB field ID for the First Occurrence column (e.g. 131666L for E11). Required when "first_occurrence" is in source and fo_col is NULL.
fo_col	(character or NULL) Column name of the First Occurrence field in data. Alternative to fo_field.
histology	(integer vector or NULL) Passed to derive_cancer_registry. Ignored for other sources.
behaviour	(integer vector or NULL) Passed to derive_cancer_registry. Ignored for other sources.
hes_code_col	(character or NULL) Passed as disease_cols to derive_hes.
hes_date_cols	(character or NULL) Passed as date_cols to derive_hes.
primary_cols	(character or NULL) Passed to derive_death_registry.
secondary_cols	(character or NULL) Passed to derive_death_registry.
death_date_cols	(character or NULL) Passed as date_cols to derive_death_registry.
cr_code_cols	(character or NULL) Passed as code_cols to derive_cancer_registry.
cr_hist_cols	(character or NULL) Passed as hist_cols to derive_cancer_registry.
cr_behv_cols	(character or NULL) Passed as behv_cols to derive_cancer_registry.
cr_date_cols	(character or NULL) Passed as date_cols to derive_cancer_registry.

Details

All intermediate source columns ({name}_hes, {name}_death, {name}_fo, {name}_cancer and their _date counterparts) are retained in data so that per-source contributions remain traceable.

{name}_icd10 Logical flag: TRUE if any selected source contains a matching record.

{name}_icd10_date Earliest matching date across all selected sources (IDate).

Value

The input data with {name}_icd10 (logical) and {name}_icd10_date (IDate) added in-place, plus all intermediate source columns. Always returns a data.table.

Examples

```
dt <- ops_toy(n = 100)
derive_icd10(dt, name = "htn",
             icd10  = "I10",
             source = c("hes", "death"))
derive_icd10(dt, name = "mi",
             icd10  = "I21",
             source = c("hes", "death", "first_occurrence"),
             fo_col = "p131742")
```

`derive_missing` *Handle informative missing labels in UKB decoded data*

Description

After `decode_values` converts categorical codes to character labels, some values represent meaningful non-response rather than true data: `"Do not know"`, `"Prefer not to answer"`, and `"Prefer not to say"`. This function either converts them to `NA` (for complete-case analysis) or retains them as `"Unknown"` (to preserve the informative missingness as a model category).

Usage

```
derive_missing(
  data,
  cols = tidyselect::everything(),
  action = c("na", "unknown"),
  extra_labels = NULL
)
```

Arguments

`data`	(data.frame or data.table) Decoded UKB data, typically output of `decode_values` followed by `decode_names`.
`cols`	(tidyselect) Columns to process. Default: `everything()` (all columns). Non-character columns in the selection are silently skipped.
`action`	(character) One of `"na"` (default) or `"unknown"`. • `"na"`: convert all informative missing labels to `NA`. • `"unknown"`: convert informative missing labels to `"Unknown"`, preserving them as a distinct model category. Empty strings are always converted to `NA` regardless of this parameter.
`extra_labels`	(character or NULL) Additional labels to treat as informative missing, appended to the built-in list. Default: `NULL`.

Details

Empty strings (`""`) are always converted to `NA` regardless of `action`, as they carry no informational content.

Only character columns are modified; numeric, integer, Date, and logical columns are silently skipped.

data.table pass-by-reference: when the input is a `data.table`, modifications are made in-place via `set`. The returned object and the original variable point to the same memory. If you need to preserve the original, pass `data.table::copy(data)` instead.

Value

The input `data` with missing labels replaced in-place. Always returns a `data.table`.

Examples

```
dt <- ops_toy(n = 100)
derive_missing(dt)

# Retain informative non-response as a model category
derive_missing(dt, action = "unknown")

# Add a custom label
derive_missing(dt, extra_labels = "Not applicable")
```

---

derive_selfreport *Define a self-reported phenotype from UKB touchscreen data*

Description

Searches UKB self-reported illness fields across all instances and arrays, matches records against a user-supplied regex, parses the associated year-month date, and appends two columns to the data: `{name}_selfreport` (logical) and `{name}_selfreport_date` (`IDate`, earliest matching instance).

Usage

```
derive_selfreport(
  data,
  name,
  regex,
  field = c("noncancer", "cancer"),
  ignore_case = TRUE,
  disease_cols = NULL,
  date_cols = NULL,
  visit_cols = NULL
)
```

Arguments

`data`	(data.frame or data.table) UKB data containing self-report fields.
`name`	(character) Output column name prefix, e.g. `"disease"` or `"outcome"`.
`regex`	(character) Regular expression matched against disease text values (after `tolower()`), e.g. `"^diabetes$"`.
`field`	(character) Self-report field type: `"noncancer"` (p20002 / p20008) or `"cancer"` (p20001 / p20006).
`ignore_case`	(logical) Should regex matching ignore case? Default: `TRUE`.
`disease_cols`	(character or NULL) Column name(s) containing disease text (p20002 or p20001). `NULL` = auto-detect via `extract_ls()` cache.
`date_cols`	(character or NULL) Column name(s) containing the self-report year-month date (p20008 or p20006). `NULL` = auto-detect.
`visit_cols`	(character or NULL) Column name(s) containing the baseline assessment date (p53). `NULL` = auto-detect.

Details

The function auto-detects the relevant columns using the `extract_ls()` field dictionary cache (populated by `extract_ls()` or `extract_pheno()`). The three `_cols` parameters let you override auto-detection when column names have been customised (e.g. after `decode_names()`).

Field mapping by `field`:

- `"noncancer"`: disease text = p20002, date = p20008.
- `"cancer"`: disease text = p20001, date = p20006.

Baseline visit date (p53) is used as a fallback when no specific diagnosis date is recorded.

data.table pass-by-reference: new columns are added in-place. Pass `data.table::copy(data)` to preserve the original.

Value

The input `data` with two new columns appended in-place: `{name}_selfreport` (logical) and `{name}_selfreport_date` (IDate). Always returns a `data.table`.

Examples

```
dt <- ops_toy(n = 100)
# disease_cols / date_cols / visit_cols can be omitted — auto-detected from
# column names matching p20002_* / p20008_* / p53_* respectively
derive_selfreport(dt, name = "htn", regex = "hypertension",
                  field = "noncancer")
```

`derive_timing` *Classify disease timing relative to UKB baseline assessment*

Description

Assigns each participant an integer timing category based on whether their disease date falls before or after the baseline visit date:

Usage

```
derive_timing(data, name, baseline_col, status_col = NULL, date_col = NULL)
```

Arguments

`data`	(data.frame or data.table) UKB phenotype data.
`name`	(character) Output column prefix. The new column is named `{name}_timing`. Also used to derive default `status_col` and `date_col` when those are `NULL`.
`baseline_col`	(character) Name of the baseline date column in `data` (e.g. `"date_baseline"` or `"p53_i0"`).

status_col (character or NULL) Name of the logical disease flag. NULL = paste0(name, "_status").

date_col (character or NULL) Name of the disease date column (IDate or Date). NULL = paste0(name, "_date").

Details

0 No disease (status_col is FALSE).

1 Prevalent - disease date on or before baseline.

2 Incident - disease date strictly after baseline.

NA Case with missing date; timing cannot be determined.

Call once per timing variable needed (e.g. once for the combined case, once per individual source).

Value

The input data with one new integer column {name}_timing (0/1/2/NA) added in-place. Always returns a data.table.

Examples

```
dt <- ops_toy(n = 100)
derive_hes(dt, name = "htn", icd10 = "I10")
derive_timing(dt, name = "htn_hes",
              status_col   = "htn_hes",
              date_col     = "htn_hes_date",
              baseline_col = "p53_i0")
```

---

extract_batch *Submit a large-scale phenotype extraction job via table-exporter*

Description

Submits an asynchronous table-exporter job on the DNAnexus Research Analysis Platform for large-scale phenotype extraction. Use this instead of extract_pheno() when extracting many fields (e.g. 50+).

Usage

```
extract_batch(
  field_id,
  dataset = NULL,
  file = NULL,
  instance_type = NULL,
  priority = c("low", "high")
)
```

Arguments

field_id	(integer) Vector of UKB Field IDs to extract. eid is always included automatically.
dataset	(character) Dataset file name. Default: NULL (auto-detect from project root).
file	(character) Output file name on the cloud (without extension), e.g. "ad_cscc_pheno". Default: NULL (auto-generate as "ukb_pheno_YYYYMMDD_HHMMSS" to avoid same-day collisions).
instance_type	(character) DNAnexus instance type, e.g. "mem1_ssd1_v2_x16". Default: NULL (auto-select: x4 for up to 20 cols, x8 for up to 100 cols, x16 for up to 500 cols, x36 for more than 500 cols).
priority	(character) Job scheduling priority. "low" (recommended, cheaper) or "high" (faster queue). Default: "low".

Details

The job runs on the cloud and typically completes in 20-40 minutes. Monitor progress and retrieve results using the job_ series.

Value

Invisibly returns the job ID string (e.g. "job-XXXX").

Examples

```
## Not run:
job_id <- extract_batch(core_field_ids)
job_id <- extract_batch(core_field_ids, file = "ad_cscc_pheno")
job_id <- extract_batch(core_field_ids, priority = "high")
# Monitor: job_status(job_id)
# Download: job_result(job_id, dest = "data/pheno.csv")

## End(Not run)
```

---

extract_ls *List all approved fields in the UKB dataset*

Description

Returns a data.frame of all fields available for extraction in the current UKB project dataset. Fields reflect what has been approved for your project — not all UKB fields are present.

Usage

```
extract_ls(dataset = NULL, pattern = NULL, refresh = FALSE)
```

Arguments

dataset	(character) Dataset file name, e.g. "app12345_20260101.dataset". Default: NULL (auto-detect).
pattern	(character) Optional regex to filter results by field_name or title. Default: NULL.
refresh	(logical) Force re-fetch from cloud, ignoring cache. Default: FALSE.

Details

Results are cached in the session after the first call. Subsequent calls return instantly from cache. Use refresh = TRUE to force a new network request (e.g. after switching projects).

Value

A data.frame with columns:

field_name Full field name as used in extraction, e.g. "participant.p31", "participant.p53_i0".

title Human-readable field description, e.g. "Sex", "Date of attending assessment centre | Instance 0".

Examples

```
## Not run:
# List all approved fields
extract_ls()

# Search by keyword
extract_ls(pattern = "cancer")
extract_ls(pattern = "p31|p53|p22009")

# Force refresh after switching projects
extract_ls(refresh = TRUE)

## End(Not run)
```

---

extract_pheno *Extract phenotype data from a UKB dataset*

Description

Extracts phenotypic fields from the UKB Research Analysis Platform dataset and returns a data.table. All instances and arrays are returned for each requested field. Column names are kept as-is (e.g. participant.p53_i0); use the clean_ series for renaming.

Usage

```
extract_pheno(field_id, dataset = NULL, timeout = 300)
```

Arguments

field_id	(integer) Vector of UKB Field IDs to extract, e.g. c(31, 53, 22189). eid is always included automatically.
dataset	(character) Dataset file name. Default: NULL (auto-detect from project root).
timeout	(integer) Extraction timeout in seconds. Default: 300.

Value

A data.table with one row per participant. Column names follow the participant.p<id>_i<n>_a<m> convention. Fields not found are skipped with a warning.

Examples

```
## Not run:
df <- extract_pheno(c(31, 53, 21022))
df <- extract_pheno(c(31, 53, 20002), dataset = "app12345_20260101.dataset")

## End(Not run)
```

---

fetch_field *Download the UKB field dictionary file*

Description

Downloads field.tsv from the Showcase metadata/ folder on the DNAnexus Research Analysis Platform. This file contains the complete UKB data dictionary: field IDs, titles, value types, and encoding references.

Usage

```
fetch_field(dest_dir, overwrite = FALSE, resume = FALSE, verbose = TRUE)
```

Arguments

dest_dir	(character) Destination directory. Created automatically if it does not exist.
overwrite	(logical) Overwrite existing local file. Default: FALSE.
resume	(logical) Resume an interrupted download. Default: FALSE.
verbose	(logical) Show download progress. Default: TRUE.

Value

Invisibly returns the local file path as a character string.

Examples

```
## Not run:
fetch_field(dest_dir = "metadata")
fetch_field(dest_dir = "metadata", overwrite = TRUE)

## End(Not run)
```

`fetch_file` *Download a file from RAP project storage*

Description

Downloads one file or all files within a folder from RAP project storage to the current directory or a specified destination within the RAP environment. This function must be called from within RAP.

Usage

```
fetch_file(path, dest_dir, overwrite = FALSE, resume = FALSE, verbose = TRUE)
```

Arguments

`path`	(character) Remote file or folder path.
`dest_dir`	(character) Destination directory. Created automatically if it does not exist.
`overwrite`	(logical) Overwrite existing local files. Default: `FALSE`.
`resume`	(logical) Resume an interrupted download. Useful for large files (e.g. `.bed`, `.bgen`). Default: `FALSE`.
`verbose`	(logical) Show download progress. Default: `TRUE`.

Value

Invisibly returns the local file path(s) as a character vector.

Examples

```
## Not run:
# Download a single metadata file
fetch_file("Showcase metadata/field.tsv", dest_dir = "data/")

# Download an entire folder
fetch_file("Showcase metadata/", dest_dir = "data/metadata/")

# Resume an interrupted download
fetch_file("results/summary_stats.csv", dest_dir = "data/", resume = TRUE)

## End(Not run)
```

`fetch_ls` *List files and folders at a remote RAP path*

Description

Returns a structured data.frame describing the contents of a remote DNAnexus Research Analysis Platform (RAP) directory. Analogous to `file_info()` but for remote project storage.

Usage

```
fetch_ls(path = ".", type = "all", pattern = NULL)
```

Arguments

`path`	(character) Remote path to list. Default: `"."` (project root). Both `"Bulk/"` and `"/Bulk/"` are accepted.
`type`	(character) Filter results by entry type: `"all"` (default), `"file"`, or `"folder"`.
`pattern`	(character) Optional regex to filter by name, e.g. `"\.bed$"`. Default: `NULL`.

Value

A data.frame with columns:

name Entry name (no trailing slash for folders).

type `"file"` or `"folder"`.

size File size string (e.g. `"120.94 GB"`), `NA` for folders or non-file objects.

modified Last modified time (`POSIXct`), `NA` for folders.

Examples

```
## Not run:
fetch_ls()
fetch_ls("Showcase metadata/", type = "file")
fetch_ls("results/", pattern = "\\.csv$")

## End(Not run)
```

fetch_metadata *Download the Showcase metadata folder*

Description

Downloads the entire Showcase `metadata/` folder from the DNAnexus Research Analysis Platform to a local directory. This includes `field.tsv`, `encoding.tsv`, and all associated encoding tables.

Usage

```
fetch_metadata(dest_dir, overwrite = FALSE, resume = FALSE, verbose = TRUE)
```

Arguments

`dest_dir`	(character) Destination directory. Created automatically if it does not exist.
`overwrite`	(logical) Overwrite existing local files. Default: `FALSE`.
`resume`	(logical) Resume interrupted downloads. Default: `FALSE`.
`verbose`	(logical) Show download progress. Default: `TRUE`.

Value

Invisibly returns the local file paths as a character vector.

Examples

```
## Not run:
fetch_metadata(dest_dir = "metadata")
fetch_metadata(dest_dir = "metadata", overwrite = TRUE)

## End(Not run)
```

fetch_tree *Print a remote RAP directory tree*

Description

Displays the remote directory structure in a tree-like format by recursively listing sub-folders up to `max_depth`. Analogous to `file_tree()` but for remote project storage.

Usage

```
fetch_tree(path = ".", max_depth = 2, verbose = TRUE)
```

Arguments

path	(character) Remote root path. Default: "." (project root). Both "Bulk/" and "/Bulk/" are accepted.
max_depth	(integer) Maximum recursion depth. Default: 2.
verbose	(logical) Whether to print the tree to the console. Default: TRUE.

Value

Invisibly returns a character vector of tree lines.

Warning

Each level of recursion triggers one HTTPS API call per folder. Deep trees (e.g. max_depth > 3) on large UKB projects may issue 100+ network requests, causing the console to hang for tens of seconds or time out. Keep max_depth at 2-3 for interactive use.

Examples

```
## Not run:
fetch_tree()
fetch_tree("Bulk/", max_depth = 2)
fetch_tree(verbose = FALSE)

## End(Not run)
```

---

fetch_url *Get pre-authenticated download URL(s) for a remote RAP file or folder*

Description

Generates temporary HTTPS URLs for files on the DNAnexus Research Analysis Platform. For a single file, returns one URL. For a folder, lists all files inside and returns a named character vector of URLs.

Usage

```
fetch_url(path, duration = "1d")
```

Arguments

path	(character) Remote file path or folder path, e.g. "Showcase metadata/field.tsv" or "Showcase metadata/".
duration	(character) How long the URLs remain valid. Accepts suffixes: s, m, h, d, w, M, y. Default: "1d" (one day).

Value

A named character vector of pre-authenticated HTTPS URLs. Names are the file names.

Examples

```
## Not run:
# Single file
fetch_url("Showcase metadata/field.tsv")

# Entire folder
fetch_url("Showcase metadata/", duration = "7d")

## End(Not run)
```

---

grs_bgen2pgen *Convert UKB imputed BGEN files to PGEN on RAP*

Description

Submits one Swiss Army Knife job per chromosome to the DNAnexus Research Analysis Platform, each converting a UKB imputed BGEN file to PGEN format with a MAF > 0.01 filter applied via plink2. Jobs run in parallel across chromosomes.

Usage

```
grs_bgen2pgen(
  chr = 1:22,
  dest = NULL,
  maf = 0.01,
  instance = "standard",
  priority = "low"
)
```

Arguments

chr	Integer vector. Chromosomes to process. Default: 1:22.
dest	Character scalar. RAP destination path for output PGEN files (e.g. "/pgen/"). Must be specified explicitly.
maf	Numeric scalar. Minor allele frequency filter passed to plink2 --maf. Variants with MAF below this threshold are excluded. Default: 0.01. Must be in (0, 0.5).
instance	Character scalar. Instance type preset: "standard" or "large". See Details. Default: "standard".
priority	Character scalar. Job priority: "low" or "high". Default: "low".

Details

The function auto-generates the plink2 driver script, uploads it once to the RAP project root (/) on RAP, then loops over chr submitting one job per chromosome. A 500 ms pause between submissions prevents API rate-limiting.

Output path is critical. The driver script writes plink2 output to /home/dnanexus/out/out/ - the fixed path that Swiss Army Knife auto-uploads to dest on completion. Output files per chromosome: chr{N}_maf001.pgen/.pvar/.psam/.log.

Instance types:

"standard" mem2_ssd1_v2_x4: 4 cores, 12 GB RAM. Suitable for smaller chromosomes (roughly chr 17–22).

"large" mem2_ssd2_v2_x8: 8 cores, 28 GB RAM, 640 GB SSD. Required for large chromosomes (roughly chr 1–16) where standard storage is insufficient.

Value

A character vector of job IDs (one per chromosome), returned invisibly. Failed submissions are NA. Use job_ls, job_status, or job_wait to monitor progress.

Examples

```
## Not run:
# Test with chr22 first (smallest chromosome)
ids <- grs_bgen2pgen(chr = 22, dest = "/pgen/", priority = "high")

# Small chromosomes - standard instance
ids_small <- grs_bgen2pgen(chr = 15:22, dest = "/pgen/")

# Large chromosomes - upgrade instance to handle storage
ids_large <- grs_bgen2pgen(chr = 1:16, dest = "/pgen/", instance = "large")

# Monitor
job_ls()

## End(Not run)
```

---

grs_check *Check and export a GRS weights file*

Description

Reads a SNP weights file, validates its content, and writes a plink2-compatible space-delimited output ready for upload to UKB RAP.

Usage

```
grs_check(file, dest = NULL)
```

Arguments

file	Character scalar. Path to the input weights file. Read via data.table::fread (format auto-detected; handles CSV, TSV, space-delimited, etc.).
dest	Character scalar or NULL. Output path for the validated, space-delimited weights file. When NULL (default), no file is written and the validated data.table is returned invisibly only.

Details

The input file must contain at least the three columns below (additional columns are ignored):

snp SNP identifier, expected in rs + digits format.

effect_allele Effect allele; must be one of A / T / C / G.

beta Effect size (log-OR or beta coefficient); must be numeric.

Checks performed:

- Required columns present.
- No NA values in the three required columns.
- No duplicate snp identifiers.
- snp matches rs[0-9]+ pattern (warning if not).
- effect_allele contains only A / T / C / G (warning if not).
- beta is numeric (error if not).

Value

A data.table with columns snp, effect_allele, and beta, returned invisibly.

Examples

```
tmp_in <- tempfile(fileext = ".csv")
weights <- data.frame(
  snp           = c("rs1234567", "rs2345678", "rs3456789"),
  effect_allele = c("A", "T", "G"),
  beta          = c(0.12, -0.05, 0.23)
)
write.csv(weights, tmp_in, row.names = FALSE)

w <- grs_check(tmp_in)
w

# Save validated weights to a file
tmp_out <- tempfile(fileext = ".txt")
grs_check(tmp_in, dest = tmp_out)
```

grs_score *Calculate genetic risk scores from PGEN files on RAP*

Description

Uploads local SNP weight files to the RAP project root, then submits one Swiss Army Knife job per GRS. Each job runs plink2 `--score` across all 22 chromosomes and saves a single CSV to `dest` on completion. Jobs run in parallel; use `job_ls` to monitor progress.

Usage

```
grs_score(
  file,
  pgen_dir = NULL,
  dest = NULL,
  maf = 0.01,
  instance = "standard",
  priority = "low"
)
```

Arguments

`file`	Named character vector of local weight file paths. Names become the GRS identifiers (output column = `GRS_<name>`). Example: `c(grs_a = "weights_a.txt")`.
`pgen_dir`	Character scalar. Path to PGEN files on RAP (e.g. `"/mnt/project/pgen"`). Must be specified explicitly.
`dest`	Character scalar. RAP destination path for output CSV files (e.g. `"/grs/"`). Must be specified explicitly.
`maf`	Numeric scalar. MAF filter threshold used when locating PGEN files. Must match the value used in `grs_bgen2pgen`. Default: `0.01`.
`instance`	Character scalar. Instance type preset: `"standard"` or `"large"`. Default: `"standard"`.
`priority`	Character scalar. Job priority: `"low"` or `"high"`. Default: `"low"`.

Details

Weight files should have three columns (any delimiter, header required):

Column 1 Variant ID (e.g. `rs` IDs).

Column 2 Effect allele (A1).

Column 3 Effect weight (beta / log-OR).

This matches the output format of `grs_check`.

Output per job: `dest/<score_name>_scores.csv` with columns `IID` and the GRS score (named `GRS_<name>`).

Instance types:

`"standard"` mem2_ssd1_v2_x4: 4 cores, 12 GB RAM.

`"large"` mem2_ssd2_v2_x8: 8 cores, 28 GB RAM.

Value

A named character vector of job IDs (one per GRS), returned invisibly. Failed submissions are `NA`. Use `job_ls` to monitor progress.

Examples

```
## Not run:
ids <- grs_score(
  file = c(
    grs_a = "weights/grs_a_weights.txt",
    grs_b = "weights/grs_b_weights.txt"
  ),
  pgen_dir = "/mnt/project/pgen",
  dest     = "/grs/",
  priority = "high"
)

job_ls()

## End(Not run)
```

`grs_standardize` *Standardise GRS columns by Z-score transformation*

Description

Adds a `_z` column for every selected GRS column: `z = (x - mean(x)) / sd(x)`. The original columns are kept unchanged. `grs_zscore` is an alias for this function.

Usage

```
grs_standardize(data, grs_cols = NULL)

grs_zscore(data, grs_cols = NULL)
```

Arguments

`data`	A `data.frame` or `data.table` containing at least one GRS column.
`grs_cols`	Character vector of column names to standardise. If `NULL` (default), all columns whose names contain `"grs"` (case-insensitive) are selected automatically.

Value

The input `data` as a `data.table` with one additional `_z` column per GRS column appended after its source column.

Examples

```
dt <- data.frame(
  IID   = 1:5,
  GRS_a = c(0.12, 0.34, 0.56, 0.23, 0.45),
  GRS_b = c(1.1,  0.9,  1.3,  0.8,  1.0)
)
grs_standardize(dt)
grs_zscore(dt)   # identical
```

`grs_validate` *Validate GRS predictive performance*

Description

For each GRS column, computes four sets of validation metrics:

1. **Per SD** - OR (logistic) or HR (Cox) per 1-SD increase.
2. **High vs Low** - OR / HR comparing top 20\ (extreme tertile grouping: Low / Mid / High).
3. **Trend test** - P-trend across quartiles (Q1–Q4).
4. **Discrimination** - AUC (logistic) or C-index (Cox).

Usage

```
grs_validate(
  data,
  grs_cols = NULL,
  outcome_col,
  time_col = NULL,
  covariates = NULL
)
```

Arguments

`data`	A `data.frame` or `data.table`.
`grs_cols`	Character vector of GRS column names to validate. If `NULL` (default), all columns whose names contain `"grs"` (case-insensitive) are selected automatically. All specified columns must be numeric.
`outcome_col`	Character scalar. Name of the outcome column (`0`/`1` or `TRUE`/`FALSE`).
`time_col`	Character scalar or `NULL`. Name of the follow-up time column. When `NULL` (default), logistic regression is used; when supplied, Cox regression is used.
`covariates`	Character vector or `NULL`. Covariates for the fully adjusted model. When `NULL`, only unadjusted and age-sex adjusted models are run.

Details

GRS grouping columns are created internally via derive_cut and are not added to the user's data. When time_col is NULL, logistic regression is used throughout; when supplied, Cox proportional hazards models are used.

Models follow the same adjustment logic as assoc_logistic / assoc_coxph: unadjusted and age-sex adjusted models are always included; a fully adjusted model is added when covariates is non-NULL.

Value

A named list with four data.table elements:

- per_sd: OR / HR per 1-SD increase in GRS.
- high_vs_low: OR / HR for High vs Low extreme tertile.
- trend: P-trend across Q1–Q4 quartiles.
- discrimination: AUC (logistic) or C-index (Cox) with 95\

Examples

```
dt <- ops_toy(scenario = "association", n = 500)
dt <- grs_standardize(dt, grs_cols = "grs_bmi")

res <- grs_validate(
  data        = dt,
  grs_cols    = "grs_bmi_z",
  outcome_col = "dm_status",
  time_col    = "dm_followup_years"
)
res$per_sd

if (requireNamespace("pROC", quietly = TRUE)) {
  res_logit <- grs_validate(
    data        = dt,
    grs_cols    = "grs_bmi_z",
    outcome_col = "dm_status"
  )
  res_logit$discrimination
}
```

---

job_ls *List recent DNAnexus jobs in the current project*

Description

Returns a summary of the most recent jobs, optionally filtered by state. Useful for quickly reviewing which jobs have completed, failed, or are still running.

Usage

```
job_ls(n = 20, state = NULL)
```

Arguments

n	(integer) Maximum number of recent jobs to return. Must be a single positive integer. Default: 20.
state	(character) Filter by state(s). Must be NULL or a character vector of valid states: "idle", "runnable", "running", "done", "failed", "terminated". Default: NULL (return all).

Value

A data.frame with columns:

job_id Job ID string, e.g. "job-XXXX".

name Job name (typically "Table exporter").

state Current job state.

created Job creation time (POSIXct).

runtime Runtime string (e.g. "0:04:36"), NA if still running.

Examples

```
## Not run:
job_ls()
job_ls(n = 5)
job_ls(state = "failed")
job_ls(state = c("done", "failed"))

## End(Not run)
```

---

job_path *Get the RAP file path of a completed DNAnexus job output*

Description

Returns the absolute /mnt/project/ path of the CSV produced by extract_batch(). Use this to read the file directly on the RAP without downloading.

Usage

```
job_path(job_id)
```

Arguments

job_id	(character) Job ID returned by extract_batch().

Value

A character string — the absolute path to the output CSV under `/mnt/project/`.

Examples

```
## Not run:
path <- job_path(job_id)
df   <- data.table::fread(path)

## End(Not run)
```

`job_result` *Load the result of a completed DNAnexus job into R*

Description

Reads the output CSV produced by `extract_batch()` directly from RAP project storage and returns a `data.table`. Must be run inside the RAP environment.

Usage

```
job_result(job_id)
```

Arguments

`job_id`	(character) Job ID returned by `extract_batch()`.

Value

A `data.table` with one row per participant.

Examples

```
## Not run:
job_id <- extract_batch(c(31, 53, 21022))
job_wait(job_id)
df <- job_result(job_id)

## End(Not run)
```

job_status *Check the current state of a DNAnexus job*

Description

Returns the current state of a job submitted by `extract_batch()`. For failed jobs, the failure message is attached as an attribute.

Usage

```
job_status(job_id)
```

Arguments

`job_id`	(character) Job ID returned by `extract_batch()`, e.g. `"job-XXXX"`.

Value

A named character string — the job state. Possible values:

`"idle"` Queued, waiting to be scheduled.

`"runnable"` Resources being allocated.

`"running"` Actively executing.

`"done"` Completed successfully.

`"failed"` Failed; see `attr(result, "failure_message")`.

`"terminated"` Manually terminated.

Examples

```
## Not run:
job_id <- extract_batch(c(31, 53, 21022))
job_status(job_id)

s <- job_status(job_id)
if (s == "failed") cli::cli_inform(attr(s, "failure_message"))

## End(Not run)
```

job_wait *Wait for a DNAnexus job to finish*

Description

Polls job_status() at regular intervals until the job reaches a terminal state ("done", "failed", or "terminated"). Stops with an error if the job fails, is terminated, or times out.

Usage

```
job_wait(job_id, interval = 30, timeout = Inf, verbose = TRUE)
```

Arguments

job_id	(character) Job ID returned by extract_batch().
interval	(integer) Polling interval in seconds. Default: 30.
timeout	(numeric) Maximum wait time in seconds. Default: Inf (wait indefinitely). On UKB RAP, jobs can stay in "runnable" for several hours during peak times — set a finite value (e.g. 7200) only if you need a hard deadline.
verbose	(logical) Print state and elapsed time at each poll. Default: TRUE.

Value

Invisibly returns the final state string ("done").

Examples

```
## Not run:
job_id <- extract_batch(c(31, 53, 21022))
job_wait(job_id)

# Read result immediately after completion (RAP only)
job_wait(job_id)
df <- job_result(job_id)

## End(Not run)
```

ops_na *Summarise missing values by column*

Description

Scans each column of a data.frame or data.table and returns the count and percentage of missing values. Results are sorted by missingness in descending order. Columns above 10\ and 10\ threshold.

Usage

```
ops_na(data, threshold = 0, verbose = TRUE)
```

Arguments

data	A data.frame or data.table to scan.
threshold	(numeric) Columns with pct_na <= threshold are silenced from the per-column CLI output. The summary block is always shown. Default 0: every column with any missing value is listed.
verbose	(logical) Print the CLI report. Default TRUE.

Details

Missing value definition: a value is counted as missing if it is NA *or* an empty string (""). Empty strings are treated as missing because UKB exports frequently use "" as a placeholder for absent text values. This means n_na and pct_na reflect *effective* missingness, not just is.na(). Numeric and logical columns are not affected (they cannot hold "").

Value

An invisible data.table with columns column, n_na, and pct_na, sorted by pct_na descending. n_na counts both NA and "". Always contains all columns regardless of threshold (which only affects CLI output).

Examples

```
dt <- ops_toy(n = 100)

# Show all columns with any missing value
ops_na(dt)

# Only list columns with > 10% missing in the CLI output
ops_na(dt, threshold = 10)

# Programmatic use — retrieve result silently
result <- ops_na(dt, verbose = FALSE)
result[pct_na > 50]
```

---

ops_setup *Check the ukbflow operating environment*

Description

Runs a four-block health check covering the dx CLI, dxpy (Python), RAP authentication, and R package dependencies. Designed to be the first function a new user runs after installation.

Usage

```
ops_setup(
  check_dx = TRUE,
  check_auth = TRUE,
  check_deps = TRUE,
  verbose = TRUE
)
```

Arguments

check_dx	(logical) Check dx CLI installation (dxpy is implied by dx).
check_auth	(logical) Check RAP login status.
check_deps	(logical) Check R package dependencies.
verbose	(logical) Print the formatted report. Set to FALSE for programmatic use (results are still returned invisibly).

Details

The function is **read-only**: it never modifies system state, installs packages, or authenticates. Auth failures are reported as warnings, not errors, because the check itself does not require a live RAP connection.

Value

An invisible named list with elements dx, dxpy, auth, deps, and summary. Each element reflects the result of its respective check block and can be inspected programmatically.

Examples

```
ops_setup(check_dx = FALSE, check_auth = FALSE)

result <- ops_setup(check_dx = FALSE, check_auth = FALSE, verbose = FALSE)
result$summary$fail == 0
```

---

ops_set_safe_cols *Register additional safe columns protected from snapshot-based drops*

Description

Adds column names to the session-level safe list. Columns in this list are automatically excluded when ops_snapshot_cols is used to build a drop vector, in addition to the built-in protected columns ("eid", "sex", "age", "age_at_recruitment").

Usage

```
ops_set_safe_cols(cols = NULL, reset = FALSE)
```

Arguments

cols	(character) One or more column names to protect.
reset	(logical) If TRUE, clear the current user-registered safe list before adding. Default FALSE.

Value

Invisibly returns the updated safe cols vector.

Examples

```
ops_set_safe_cols(c("date_baseline", "townsend_index"))
ops_set_safe_cols(reset = TRUE)  # clear user-registered safe cols
```

---

ops_snapshot *Record and review dataset pipeline snapshots*

Description

Captures a lightweight summary of a data.frame at a given pipeline stage and stores it in the session cache. Subsequent calls automatically compute deltas against the previous snapshot, making it easy to track how data changes through derive_*, assoc_*, and other processing steps.

Usage

```
ops_snapshot(
  data = NULL,
  label = NULL,
  reset = FALSE,
  verbose = TRUE,
  check_na = TRUE
)
```

Arguments

data	A data.frame or data.table to snapshot. Pass NULL (or omit) to print the full snapshot history without recording a new entry.
label	(character) A short name for this pipeline stage, e.g. "raw", "after_derive_missing". Defaults to "snapshot_N" where N is the sequential index.
reset	(logical) If TRUE, clears the entire snapshot history and returns invisibly. Default FALSE.
verbose	(logical) Print the CLI report. Default TRUE.
check_na	(logical) Whether to count columns with any NA or blank string values and include the delta in the report. Set to FALSE to skip the NA scan (useful for large datasets or when NA tracking is not needed). Default TRUE.

Value

When `data` is supplied, returns the new snapshot row invisibly (a one-row data.table). When called with no `data`, returns the full history data.table invisibly.

Examples

```
dt <- ops_toy(n = 100)
ops_snapshot(dt, label = "raw")

dt <- derive_missing(dt)
ops_snapshot(dt, label = "after_derive_missing")

# View full history
ops_snapshot()

# Reset history
ops_snapshot(reset = TRUE)
```

---

ops_snapshot_cols *Retrieve column names recorded at a snapshot*

Description

Returns the column names stored by a previous `ops_snapshot` call, optionally excluding columns you wish to keep.

Usage

```
ops_snapshot_cols(label, keep = NULL)
```

Arguments

`label`	(character) Snapshot label passed to `ops_snapshot()`.
`keep`	(character or NULL) Column names to exclude from the returned vector (i.e. columns to retain in the data even if they were present at that snapshot). Default `NULL`.

Value

A character vector of column names.

Examples

```
dt <- ops_toy(n = 100)
ops_snapshot(dt, label = "raw")
ops_snapshot_cols("raw")
ops_snapshot_cols("raw", keep = "eid")
```

ops_snapshot_diff *Compare column names between two snapshots*

Description

Returns lists of columns added and removed between two recorded snapshots.

Usage

```
ops_snapshot_diff(label1, label2)
```

Arguments

label1	(character) Label of the earlier snapshot.
label2	(character) Label of the later snapshot.

Value

A named list with two character vectors: added (columns present in label2 but not label1) and removed (columns present in label1 but not label2).

Examples

```
dt <- ops_toy(n = 100)
ops_snapshot(dt, label = "raw")
dt <- derive_missing(dt)
ops_snapshot(dt, label = "derived")
ops_snapshot_diff("raw", "derived")
# $added   — newly derived columns
# $removed — columns dropped between snapshots
```

ops_snapshot_remove *Remove raw source columns recorded at a snapshot*

Description

Drops columns that were present at snapshot from from data, while automatically protecting built-in safe columns ("eid", "sex", "age", "age_at_recruitment") and any user-registered safe columns set via ops_set_safe_cols. Columns that no longer exist in data are silently skipped.

Usage

```
ops_snapshot_remove(data, from, keep = NULL, verbose = TRUE)
```

Arguments

data	A data.frame or data.table.
from	(character) Label of the snapshot whose columns should be dropped (typically "raw").
keep	(character or NULL) Additional column names to protect beyond the built-in and user-registered safe cols. Default NULL.
verbose	(logical) Print a summary of dropped columns. Default TRUE.

Value

A data.table with the specified columns removed. For data.table input the operation is performed by reference (in-place); for data.frame input the data is first converted to a new data.table — the original data.frame is not modified.

Examples

```
dt <- ops_toy(n = 100)
ops_snapshot(dt, label = "raw")
dt <- derive_missing(dt)
ops_snapshot(dt, label = "derived")
ops_snapshot_diff("raw", "derived")
dt <- ops_snapshot_remove(dt, from = "raw")
```

---

ops_toy *Generate toy UKB-like data for testing and development*

Description

Creates a small, synthetic dataset that mimics the structure of UK Biobank phenotype data on the RAP. Useful for developing and testing derive_*, assoc_*, and plot_* functions without requiring real UKB data access.

Usage

```
ops_toy(scenario = "cohort", n = 1000L, seed = 42L)
```

Arguments

scenario	(character) Data structure to generate: • "cohort": wide participant-level table with raw UKB field columns for the full derive_* → assoc_* → plot_* pipeline. • "association": analysis-ready table with covariates already as factors, BMI/TDI binned, and two pre-derived disease outcomes (dm_*, htn_*) including status, date, timing, and follow-up columns. Use this for assoc_* examples and testing without running the derive pipeline.

- `"forest"`: association results table matching `assoc_coxph()` output, for testing `plot_forest()`. `n` = number of exposures (default 8).

`n` (integer) Number of participants (or exposures for `"forest"`). Default `1000L` for `"cohort"`, `2000L` for `"association"`, `8L` for `"forest"`.

`seed` (integer or NULL) Random seed for reproducibility. Pass `NULL` for a different dataset on every call. Default `42L`.

Details

This dataset is entirely synthetic. Column names follow RAP conventions (e.g. `p41270`, `p20002_i0_a0`).

scenario = "cohort":

Includes the following column groups:

- **Demographics**: `eid`, `p31`, `p34`, `p53_i0`, `p21022`
- **Covariates**: `p21001_i0`, `p20116_i0`, `p1558_i0`, `p21000_i0`, `p22189`, `p54_i0`
- **Genetic PCs**: `p22009_a1` – `p22009_a10`
- **Self-report disease**: `p20002_i0_a0-a4`, `p20008_i0_a0-a4`
- **Self-report cancer**: `p20001_i0_a0-a4`, `p20006_i0_a0-a4`
- **HES**: `p41270` (JSON array), `p41280_a0-a8`
- **Cancer registry**: `p40006_i0-i2`, `p40011_i0-i2`, `p40012_i0-i2`, `p40005_i0-i2`
- **Death registry**: `p40001_i0`, `p40002_i0_a0-a2`, `p40000_i0`
- **First occurrence**: `p131742`
- **GRS**: `grs_bmi`, `grs_raw`, `grs_finngen`
- **Messy columns**: `messy_allna`, `messy_empty`, `messy_label`

scenario = "association":

Analysis-ready table. All derive inputs (raw arrays, HES JSON, registry fields) are omitted; derive outputs are pre-computed with internally consistent relationships:

- **Demographics**: `eid`, `p31` (factor), `p53_i0` (IDate), `p21022`
- **Covariates**: `p21001_i0`, `bmi_cat` (factor, derived from `p21001_i0`), `p20116_i0` (factor), `p1558_i0` (factor), `p21000_i0` (factor), `p22189`, `tdi_cat` (factor, derived from `p22189` quartiles), `p54_i0` (factor)
- **Genetic PCs**: `p22009_a1` – `p22009_a10`
- **GRS**: `grs_bmi` (continuous exposure)
- **DM outcome**: `dm_status`, `dm_date`, `dm_timing`, `dm_followup_end`, `dm_followup_years` (type 2 diabetes, ~12% prevalence)
- **HTN outcome**: `htn_status`, `htn_date`, `htn_timing`, `htn_followup_end`, `htn_followup_years` (hypertension, ~28% prevalence)

Internal relationships guaranteed:

- `bmi_cat` is cut from `p21001_i0` (breaks 18.5 / 25 / 30)
- `tdi_cat` is cut from `p22189` quartiles
- `dm_date` is `NA` iff `dm_status` = `FALSE`
- `dm_timing`: 0 = no disease, 1 = prevalent, 2 = incident, `NA` = no date
- `dm_followup_years` is `NA` for prevalent cases (`dm_timing == 1`)

Value

A data.table with UKB-style column names. See Details for the columns included in each scenario.

Examples

```
# cohort: raw UKB-style columns, feed into derive pipeline
dt <- ops_toy(n = 100)
dt <- derive_missing(dt)

# association: analysis-ready, feed directly into assoc_* functions
dt <- ops_toy(scenario = "association", n = 500)
dt <- dt[dm_timing != 1L]   # exclude prevalent cases

# forest: results table for plot_forest()
dt <- ops_toy(scenario = "forest")
```

---

ops_withdraw *Exclude withdrawn participants from a dataset*

Description

Reads a UK Biobank withdrawal list (a headerless single-column CSV of anonymised participant IDs) and removes the corresponding rows from data. A pair of ops_snapshot() calls is made automatically so the before/after row counts are recorded in the session snapshot history.

Usage

```
ops_withdraw(data, file, eid_col = "eid", verbose = TRUE)
```

Arguments

data	A data.frame or data.table containing a participant ID column.
file	(character) Path to the UKB withdrawal CSV file. The file must be a single-column, **header-free** CSV as supplied by UK Biobank (e.g. w854944_20260310.csv).
eid_col	(character) Name of the participant ID column in data. Default "eid".
verbose	(logical) Print the CLI report. Default TRUE.

Value

A data.table with withdrawn participants removed.

Examples

```
dt <- ops_toy(n = 100)
withdraw_file <- tempfile(fileext = ".csv")
writeLines(as.character(dt$eid[1:5]), withdraw_file)
dt <- ops_withdraw(dt, file = withdraw_file)
```

plot_forest *Publication-ready forest plot*

Description

Produces a publication-ready forest plot with UKB-standard styling. The user supplies a data frame whose first column is the row label (`item`), plus any additional display columns (e.g. `Cases/N`). The gap column and the auto-formatted `OR (95% CI)` text column are inserted automatically at `ci_column`. Numeric p-value columns declared via `p_cols` are formatted in-place.

Usage

```
plot_forest(
  data,
  est,
  lower,
  upper,
  ci_column = 2L,
  ref_line = 1,
  xlim = NULL,
  ticks_at = NULL,
  arrow_lab = c("Lower risk", "Higher risk"),
  header = NULL,
  indent = NULL,
  bold_label = NULL,
  ci_col = "black",
  ci_sizes = 0.6,
  ci_Theight = 0.2,
  ci_digits = 2L,
  ci_sep = ", ",
  p_cols = NULL,
  p_digits = 3L,
  bold_p = TRUE,
  p_threshold = 0.05,
  align = NULL,
  background = "zebra",
  bg_col = "#F0F0F0",
  border = "three_line",
  border_width = 3,
  row_height = NULL,
  col_width = NULL,
  save = FALSE,
  dest = NULL,
  save_width = 20,
  save_height = NULL,
  theme = "default"
)
```

Arguments

data	data.frame. First column must be the label column (item). Additional columns are displayed as-is (character) or formatted if named in p_cols (must be numeric). Column order is preserved.
est	Numeric vector. Point estimates (NA = no CI drawn).
lower	Numeric vector. Lower CI bounds (same length as est).
upper	Numeric vector. Upper CI bounds (same length as est).
ci_column	Integer. Column position in the final rendered table where the gap/CI graphic is placed. Must be between 2 and ncol(data) + 1 (inclusive). Default: 2L.
ref_line	Numeric. Reference line. Default: 1 (HR/OR). Use 0 for beta coefficients.
xlim	Numeric vector of length 2. X-axis limits. NULL (default) uses c(0, 2).
ticks_at	Numeric vector. Tick positions. NULL (default) = 5 evenly spaced ticks across xlim.
arrow_lab	Character vector of length 2. Directional labels. Default: c("Lower risk", "Higher risk"). NULL = none.
header	Character vector of length ncol(data) + 2. Column header labels for the final rendered table (original columns + gap_ci + OR label). NULL (default) = use column names from data plus "gap_ci" and "OR (95% CI)". Pass "" for the gap column position.
indent	Integer vector (length = nrow(data)). Indentation level of the label column: each unit adds two leading spaces. Default: all zeros.
bold_label	Logical vector (length = nrow(data)). Which rows to bold in the label column. NULL (default) = auto-derive from indent: rows where indent == 0 are bolded (parent rows), indented sub-rows are plain.
ci_col	Character scalar or vector (length = nrow(data)). CI colour(s). NA rows are skipped automatically. Default: "black".
ci_sizes	Numeric. Point size. Default: 0.6.
ci_Theight	Numeric. CI cap height. Default: 0.2.
ci_digits	Integer. Decimal places for the auto-generated OR (95% CI) column. Default: 2L.
ci_sep	Character. Separator between lower and upper CI in the label, e.g. ", " or " - ". Default: ", ".
p_cols	Character vector. Names of numeric p-value columns in data. These are formatted to p_digits decimal places with "<0.001"-style clipping. NULL = none.
p_digits	Integer. Decimal places for p-value formatting. Default: 3L.
bold_p	TRUE (bold all non-NA p below p_threshold), FALSE (no bolding), or a logical vector (per-row control). Default: TRUE.
p_threshold	Numeric. P-value threshold for bolding when bold_p = TRUE. Default: 0.05.
align	Integer vector of length ncol(data) + 2. Alignment per column: -1 left, 0 centre, 1 right. Must cover all final columns (original + gap_ci + OR label). NULL = auto (column 1 left, all others centre).

background	Character. Row background style: "zebra", "bold_label" (shade rows where bold_label = TRUE), or "none". Default: "zebra".
bg_col	Character. Shading colour for backgrounds (scalar), or a per-row vector of length nrow(data) (overrides style). Default: "#F0F0F0".
border	Character. Border style: "three_line" or "none". Default: "three_line".
border_width	Numeric. Border line width(s). Scalar = all three lines same width; length-3 vector = top-of-header, bottom-of-header, bottom-of-body. Default: 3.
row_height	NULL (auto), numeric scalar, or numeric vector (length = total gtable rows including margins). Auto sets 8 / 12 / 10 / 15 mm for top / header / data / bottom respectively.
col_width	NULL (auto), numeric scalar, or numeric vector (length = total gtable columns). Auto rounds each default width up so the adjustment is in $[5, 10)$ mm.
save	Logical. Save output to files? Default: FALSE.
dest	Character. Destination file path (extension ignored; all four formats are saved). Required when save = TRUE.
save_width	Numeric. Output width in cm. Default: 20.
save_height	Numeric or NULL. Output height in cm. NULL = nrow(data) * 0.9 + 3.
theme	Character preset ("default") or a forestploter::forest_theme object. Default: "default".

Value

A **forestploter** plot object (gtable), returned invisibly. Display with plot() or grid::grid.draw().

Examples

```
df <- data.frame(
  item      = c("Exposure vs. control", "Unadjusted", "Fully adjusted"),
  `Cases/N` = c("", "89/4521", "89/4521"),
  p_value   = c(NA_real_, 0.001, 0.006),
  check.names = FALSE
)

p <- plot_forest(
  data       = df,
  est        = c(NA, 1.52, 1.43),
  lower      = c(NA, 1.18, 1.11),
  upper      = c(NA, 1.96, 1.85),
  ci_column  = 2L,
  indent     = c(0L, 1L, 1L),
  bold_label = c(TRUE, FALSE, FALSE),
  p_cols     = "p_value",
  xlim       = c(0.5, 3.0)
)
plot(p)
```

plot_tableone *Publication-ready Table 1 (Baseline Characteristics)*

Description

Generates a publication-quality baseline-characteristics table (Table 1) using **gtsummary**, with optional SMD column, *Lancet*-style theming, and automatic export to four formats.

Usage

```
plot_tableone(
  data,
  vars,
  strata = NULL,
  type = NULL,
  label = NULL,
  statistic = NULL,
  digits = NULL,
  percent = "column",
  missing = "no",
  add_p = TRUE,
  add_smd = FALSE,
  overall = FALSE,
  exclude_labels = NULL,
  theme = "lancet",
  label_width = 200,
  stat_width = 140,
  pvalue_width = 100,
  row_height = 8,
  save = FALSE,
  dest = NULL,
  png_scale = 2,
  pdf_width = NULL,
  pdf_height = NULL
)
```

Arguments

data	data.frame. Input data containing all variables.
vars	Character vector. Variable names to display in the table.
strata	Character scalar. Column name for the grouping/stratification variable. NULL produces an unstratified overall summary.
type	Named list or NULL. Variable type overrides passed directly to tbl_summary. E.g. list(age = "continuous2").
label	Named list / formula list or NULL. Label overrides. E.g. list(age ~ "Age (years)", sex ~ "Sex").

statistic	Named list or NULL. Statistic format strings. E.g. list(all_continuous() ~ "{mean} ({sd})").
digits	Named list or NULL. Decimal place overrides. E.g. list(age ~ 1).
percent	Character. Percentage base for categorical variables: "column" (default), "row", or "cell".
missing	Character. Display of missing counts: "no" (default), "ifany", or "always".
add_p	Logical. Add a p-value column. Default TRUE; disabled with a warning if strata is NULL.
add_smd	Logical. Add a standardised mean difference (SMD) column. Continuous variables use Cohen's *d*; categorical variables use root-mean-square deviation (RMSD) of group proportions. SMD is shown only on label rows. Default FALSE; disabled with a warning if strata is NULL.
overall	Logical. Add an Overall column when strata is set. Default FALSE.
exclude_labels	Character vector or NULL. Row labels to remove from the rendered table, matched exactly against the label column in the gtsummary table body. E.g. c("Unknown", "Missing").
theme	Character. Visual theme preset. Only "lancet" (default) is currently supported. Controls alternating row shading (#f8e8e7 / white), three-line borders, and 15 px font size.
label_width	Integer. Label column width in pixels. Default 200.
stat_width	Integer. Statistics column(s) width in pixels. Default 140.
pvalue_width	Integer. P-value and SMD column width in pixels. Default 100.
row_height	Integer. Data row padding (top + bottom) in pixels. Default 8.
save	Logical. Export the table to files. Default FALSE.
dest	Character or NULL. File path without extension. When save = TRUE, four files are written: <dest>.docx, <dest>.html, <dest>.pdf, <dest>.png. Required when save = TRUE.
png_scale	Numeric. Zoom factor for PNG export via **webshot2**. Higher values produce larger, higher-resolution images. Default: 2.
pdf_width	Numeric or NULL. PDF paper width in inches passed to pagedown::chrome_print. A larger value increases the page size so more content fits on a single page. Default: NULL (Chrome default).
pdf_height	Numeric or NULL. PDF paper height in inches. Increase if the table is cut off across pages. Default: NULL.

Details

The following behaviours are fixed (not exposed as parameters):

- Variable labels are **bold** (bold_labels()).
- P-values are formatted as <0.001 or to 3 decimal places.
- Significant p-values ($p < 0.05$) are **bold**.
- The p-value column header is rendered as *P*-value.
- The table uses a three-line (booktabs) border.
- Saving always exports **word**, **html**, **pdf**, and **png**.

Value

A **gt** table object, returned invisibly.

Examples

```
library(gtsummary)
data(trial)

# Basic stratified table
plot_tableone(
  data   = trial,
  vars   = c("age", "marker", "grade"),
  strata = "trt",
  save   = FALSE
)

# With SMD, custom labels, exclude Unknown level
plot_tableone(
  data           = trial,
  vars           = c("age", "marker", "grade", "stage"),
  strata         = "trt",
  label          = list(age ~ "Age (years)", marker ~ "Marker level"),
  add_smd        = TRUE,
  exclude_labels = "Unknown",
  save           = FALSE
)
```

Index