

Package: UKBAlytica (via r-universe)

October 6, 2026

Title UK Biobank Data Processing and Survival Analysis Toolkit

Version 1.1.0

Author Nan He [aut, cre] (<<https://orcid.org/0009-0008-6932-3867>>)

Maintainer Nan He <hinna01@163.com>

Description Provides an integrated workflow for UK Biobank Research Analysis Platform (RAP) hosted and RAP-generated analysis tables. The package supports RAP phenotype extraction planning, predefined variable sets and disease definitions, standardized baseline preprocessing, multi-source endpoint ascertainment, prevalent and incident case classification, survival-ready cohort construction, GWAS/PheWAS phenotype preparation, published and PLINK2-computed polygenic risk scores, regression, multiple imputation, propensity score analysis, mediation analysis, subgroup and sensitivity analyses, machine learning, proteomics enrichment and protein-protein interaction analysis, and publication-oriented visualization. The package workflow is described in He et al. (2026) <[doi:10.64898/2026.06.19.26356057](https://doi.org/10.64898/2026.06.19.26356057)>.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL <https://github.com/Hinna0818/UKBAlytica>,
<https://hinna0818.github.io/UKBAlytica/>

BugReports <https://github.com/Hinna0818/UKBAlytica/issues>

Imports data.table, stringi, ggplot2, rlang, scales, survival, pROC, tableone, mice, mitools, sandwich, lmttest, MASS, mgcv, xml2, igraph, withr

Suggests testthat, rms, gbm, cobalt, survminer, MatchIt, regmedint, kernelshap, fastshap, ranger, xgboost, glmnet, e1071, nnet, rpart, Boruta, rBayesianOptimization, randomForestSRC, clusterProfiler, AnnotationDbi, org.Hs.eg.db, qs2, PheWAS, DBI, sparklyr

Remotes PheWAS/PheWAS

Config/testthat/edition 3

Config/Needs/bioc TRUE

Depends R (>= 4.0)

LazyData true

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libxml2-dev libx11-dev zlib1g-dev

Repository <https://hinna0818.r-universe.dev>

Date/Publication 2026-08-07 10:58:57 UTC

RemoteUrl <https://github.com/hinna0818/ukbanalytica>

RemoteRef HEAD

RemoteSha 40258f9f0a6d59dcdb86da6b79265b71cbf406f7

Contents

assess_balance	7
assess_gp_observability	8
build_gwas_phewas_phenotype	9
build_survival_dataset	11
calculate_air_pollution	13
calculate_blood_pressure	14
calculate_diet_score	14
calculate_weights	15
classify_metabolites	16
coef.mediation_result	16
combine_disease_definitions	17
combine_prs_weights	17
compare_data_sources	18
compute_protein_ppi_metrics	19
confint.mediation_result	19
convert_gwas_datatype	20
create_baseline_table	21
create_disease_definition	22
create_imputation_list	24
create_medication_definition	25
estimate_propensity_score	26
extract_cases_by_source	26
extract_diabetes_subtype_baseline	28
extract_disease_diagnosis	29
extract_disease_history	30
extract_disease_history_sensitivity	31
extract_medications	32
extract_self_report_medications	32
fit_mi_models	34
get_death_dates	34

get_disease_catalog	35
get_field_info	35
get_field_metadata	36
get_medication_catalog	37
get_pomegranate_diseases	38
get_pomegranate_source_manifest	39
get_predefined_diseases	39
get_predefined_medications	41
get_protein_ppi	41
get_ukb_demo_colnames	42
get_variable_info	43
get_variable_set	43
get_variable_sets	44
harmonize_prs_weights	45
integrate_gp_results	46
load_pomegranate_portal_coding	47
load_ukb_medication_coding	47
load_ukb_metabolite_panel	48
load_ukb_prs	48
match_propensity	49
metabolite_to_metaboanalyst_name	50
parse_cancer_registry	51
parse_death_records	52
parse_gp_clinical	53
parse_gp_query	53
parse_gp_registrations	54
parse_icd10_diagnoses	55
parse_icd9_diagnoses	56
parse_opcs4_procedures	56
parse_self_reported_illnesses	57
plot.ukb_ml_flow	58
plot.ukb_ml_flow_compare	59
plot_balance	59
plot_calibration	60
plot_correlation	61
plot_cox_loghr_correlation	62
plot_cox_sensitivity_correlation	62
plot_enrichment_lollipop	63
plot_forest	64
plot_go_ora_bar	65
plot_heatmap	66
plot_km_curve	67
plot_mediation	68
plot_mediation_forest	69
plot_metabolite_ora_barplot	70
plot_metabolite_ora_dotplot	70
plot_mi_diagnostics	71
plot_mi_pooled	72

plot_ml_calibration	73
plot_ml_compare	73
plot_ml_confusion	74
plot_ml_dca	75
plot_ml_gain	75
plot_ml_importance	76
plot_ml_ks	76
plot_ml_lift	77
plot_ml_pr	77
plot_ml_roc	78
plot_ml_roc_compare	78
plot_participant_flow	79
plot_ps_distribution	80
plot_rcs	81
plot_regression_volcano	82
plot_scatter	83
plot_shap_beeswarm	84
plot_shap_dependence	86
plot_shap_force	86
plot_shap_summary	87
plot_stacked_bar	88
plot_top_hr_bars	89
plot_violin	89
pool_custom_estimates	90
pool_mi_models	91
prepare_prs_weights	92
preprocess_baseline	93
print.mediation_result	94
protein_to_gene_symbol	95
rank_protein_ppi_nodes	96
rap_extract_pheno	97
rap_find_dataset	98
rap_list_fields	98
rap_plan_extract	99
rap_plan_gp_query	100
rap_run_gp_query	100
rap_submit_extract	101
read_prs_scores	102
run_correlation	103
run_gp_workflow	103
run_imputation	105
run_mediation	107
run_metabolite_ora	109
run_multi_mediator	110
run_multi_subgroup	111
run_protein_kegg_ora	112
run_protein_ora	113
run_protein_ppi_clustering	115

run_protein_ppi_robustness	116
run_rcs	117
run_regression	119
run_subgroup_analysis	120
run_weighted_analysis	121
runmulti_competing	122
runmulti_cox	123
runmulti_cox_lag	124
runmulti_cox_zph	124
runmulti_gam	125
runmulti_glm	126
runmulti_lm	127
runmulti_logit	128
runmulti_negbin	129
runmulti_trend	129
score_protein_ppi_clusters	130
select_incident_by_years	131
sensitivity_exclude_early_events	132
sensitivity_exclude_missing_covariates	133
split_prs_weights	134
subset_protein_ppi	135
summarise_gp_coverage	136
summarise_gp_diagnoses	137
summary.mediation_result	137
tidy.mi_pooled_result	138
ukb_check_rap_env	138
ukb_clean_missing	139
ukb_compare_cox_results	140
ukb_compare_sensitivity_cox	141
ukb_cox_diagnostics	142
ukb_create_extraction_manifest	143
ukb_decode	144
ukb_decode_column_names	145
ukb_decode_values	146
ukb_demo	147
ukb_dictionary_zh	147
ukb_download_rap_dictionary	148
ukb_extract_fields	149
ukb_field_info	150
ukb_gwas_phewas_tools	150
ukb_gwas_qc_profile	151
ukb_metadata_setup	152
ukb_ml_as_split	153
ukb_ml_calibration	154
ukb_ml_compare	154
ukb_ml_compare_feature_sets	155
ukb_ml_compare_flows	157
ukb_ml_confusion	158

ukb_ml_cv	159
ukb_ml_dca	160
ukb_ml_evaluate_test	160
ukb_ml_feature_select	161
ukb_ml_fit_final	162
ukb_ml_flow	163
ukb_ml_gain_lift	165
ukb_ml_importance	166
ukb_ml_ks	166
ukb_ml_metrics	167
ukb_ml_model	168
ukb_ml_nested_cv	169
ukb_ml_pr	171
ukb_ml_predict	171
ukb_ml_roc	172
ukb_ml_roc_data	173
ukb_ml_split_data	174
ukb_ml_supported_models	175
ukb_ml_survival	175
ukb_ml_survival_as_split	176
ukb_ml_survival_evaluate_test	177
ukb_ml_survival_feature_select	178
ukb_ml_survival_fit_final	179
ukb_ml_survival_importance	180
ukb_ml_survival_predict	180
ukb_ml_survival_shap	181
ukb_ml_survival_split_data	182
ukb_ml_survival_tune	183
ukb_ml_survival_workflow	184
ukb_ml_threshold	185
ukb_ml_tune	186
ukb_ml_workflow	187
ukb_participant_flow	188
ukb_plan_plink2	190
ukb_plan_plink2_dosage	190
ukb_plan_prs	191
ukb_plan_regenie	193
ukb_plan_regenie_command	195
ukb_protein_annotation	195
ukb_prs_catalog	196
ukb_query_dictionary	197
ukb_run_phewas	198
ukb_run_plink2	199
ukb_run_prs	200
ukb_run_regenie	201
ukb_scale_with_parameters	202
ukb_search_fields	202
ukb_sensitivity_suite	203

ukb_shap	204
ukb_shap_dependence	205
ukb_shap_force	206
ukb_shap_summary	207
ukb_snapshot	207
ukb_standardize_by_train	208
ukb_time_skeleton	209
ukb_top_hr_results	210
ukb_train_validation_cox	211
ukb_validate_columns	212
ukb_write_extraction_manifest	213
ukb_write_gwas_phewas_phenotype	213
UKBAlytica	214
Index	216

assess_balance	<i>Assess Covariate Balance</i>
----------------	---------------------------------

Description

Assess balance of covariates between treatment groups before and after matching or weighting.

Usage

```
assess_balance(  
  data,  
  treatment,  
  covariates,  
  method = c("unmatched", "matched", "weighted"),  
  weight_col = NULL,  
  threshold = 0.1  
)
```

Arguments

- | | |
|------------|---|
| data | A data.frame or data.table. |
| treatment | Character string specifying the treatment variable name. |
| covariates | Character vector of covariate names to assess. |
| method | Character string specifying the data type: "unmatched", "matched", or "weighted". |
| weight_col | Character string specifying the weight column name (for weighted method). |
| threshold | Numeric threshold for SMD to determine balance. Default 0.1. |

Value

A data.frame with balance statistics:

variable Variable name

mean_treated Mean in treatment group

mean_control Mean in control group

smd Standardized mean difference

variance_ratio Variance ratio (treated/control)

balanced Whether SMD < threshold

assess_gp_observability

Assess GP Observability in a Participant-Specific Study Window

Description

Uses parsed gp_registrations intervals to determine whether each participant has sufficient primary-care observation for classification as a non-case. Overlapping and adjacent registration intervals are merged before calculating coverage, so duplicated registrations do not inflate observed days. Open registrations are usable only when observation_end is supplied.

Usage

```
assess_gp_observability(
  registrations,
  participants = NULL,
  window_start,
  window_end,
  index_date = NULL,
  observation_end = NULL,
  min_lookback_days = 0L,
  min_followup_days = 0L,
  min_coverage_fraction = 1
)
```

Arguments

registrations	Output from parse_gp_registrations() .
participants	Optional participant vector or data.frame containing eid. A data.frame may also contain participant-specific study dates.
window_start	Study-window start. Supply one date, a column name in participants, or one value per participant.
window_end	Study-window end, using the same forms as window_start.

index_date	Optional index/baseline date. When supplied, the participant must be covered on this date and continuous lookback/follow-up around the date can be required.
observation_end	Optional GP release/provider cut-off used to close open registrations and cap later deduction dates.
min_lookback_days	Minimum continuous covered days immediately before index_date.
min_followup_days	Minimum continuous covered days immediately after index_date.
min_coverage_fraction	Minimum fraction of the complete study window covered by valid registration intervals.

Value

A participant-level data.table with observed days, coverage fraction, continuous lookback/follow-up, control_eligible, and exclusion reason.

build_gwas_phewas_phenotype

Build a GWAS and PheWAS Phenotype Object

Description

Prepare aligned, in-memory phenotype inputs for a UK Biobank GWAS followed by PheWAS. The function produces a REGENIE-compatible wide table with FID/IID, one or more phenotypes, and covariates, plus the four-column long diagnosis table used by the PheWAS R package (id, vocabulary_id, code, count).

No files are written and no participant data are cached by the package. Use [ukb_write_gwas_phewas_phenotype](#) explicitly inside RAP to materialize analysis files in RAP-controlled storage.

Usage

```
build_gwas_phewas_phenotype(
  data,
  phenotype_cols,
  diagnoses,
  covariates = character(),
  categorical_covariates = character(),
  id_col = "eid",
  family_id_col = NULL,
  diagnosis_id_col = id_col,
  diagnosis_code_col = "icd10_code",
  diagnosis_date_col = NULL,
  diagnosis_count_col = NULL,
  vocabulary_id = "ICD10",
```

```

genotype_ids = NULL,
sample_qc = ukb_gwas_qc_profile("none"),
trait_types = NULL,
covariate_missing = c("keep", "complete_case", "mean_mode"),
sex_col = NULL,
sex_map = c(`0` = "F", `1` = "M", F = "F", M = "M"),
min_code_count = 1L
)

```

Arguments

<code>data</code>	Participant-level data.frame or data.table with one row per participant.
<code>phenotype_cols</code>	One or more phenotype columns. Binary phenotypes must use 0/1/NA coding; quantitative phenotypes must be numeric.
<code>diagnoses</code>	Long diagnosis table. It should normally be the result of parse_icd10_diagnoses or an equivalent table containing a participant ID and diagnosis code.
<code>covariates</code>	Covariate columns to retain in the GWAS/PheWAS inputs.
<code>categorical_covariates</code>	Subset of <code>covariates</code> that should be treated as categorical by downstream GWAS tools.
<code>id_col</code>	Participant identifier column in data.
<code>family_id_col</code>	Optional family identifier column. If NULL, FID is set equal to IID, as in the RAP tutorial.
<code>diagnosis_id_col</code>	Participant identifier column in diagnoses.
<code>diagnosis_code_col</code>	Diagnosis-code column in diagnoses.
<code>diagnosis_date_col</code>	Optional diagnosis-date column. When supplied, distinct non-missing dates are counted per participant-code pair.
<code>diagnosis_count_col</code>	Optional existing numeric count column. It cannot be used together with <code>diagnosis_date_col</code> .
<code>vocabulary_id</code>	Vocabulary label expected by PheWAS. For UKB field 41270 use "ICD10".
<code>genotype_ids</code>	Optional vector of participant IDs present in the genotype files. When supplied, phenotype participants are intersected with this vector.
<code>sample_qc</code>	A profile returned by ukb_gwas_qc_profile , or a compatible custom list.
<code>trait_types</code>	Optional named character vector with one value per phenotype: "binary" or "quantitative". If NULL, types are inferred conservatively.
<code>covariate_missing</code>	How to handle missing covariates: "keep" preserves missing values, "complete_case" removes affected participants, and "mean_mode" performs explicit mean/mode imputation.
<code>sex_col</code>	Optional sex column used to construct PheWAS sex restrictions.
<code>sex_map</code>	Named character vector mapping values in <code>sex_col</code> to "F" or "M". The default supports UKB field 31 coding and already formatted values.

`min_code_count` Minimum retained diagnosis count per participant-code pair. This filters the long code table only; the downstream PheWAS case threshold remains independently configurable.

Value

An object of class `ukb_gwas_phewas_phenotype`.

`build_survival_dataset`

Build Survival Analysis Dataset

Description

Integrates diagnosis data from multiple sources (ICD-10, ICD-9, self-report, death, OPCS4 procedures, cancer registry records, First Occurrence fields, algorithm) to generate a survival dataset. By default, returns a wide table that retains all participants and adds disease history/incident indicators plus follow-up time for a primary disease.

Usage

```
build_survival_dataset(
  dt,
  disease_definitions,
  prevalent_sources = c("ICD10", "ICD9", "Self-report", "Death"),
  outcome_sources = c("ICD10", "ICD9", "Death"),
  censor_date = as.Date("2023-10-31"),
  baseline_col = "p53_i0",
  time_skeleton = NULL,
  primary_disease = NULL,
  output = c("wide", "long"),
  include_all = TRUE,
  show_flow = TRUE,
  dt_threads = NULL
)
```

Arguments

`dt` A `data.table` or `data.frame` containing complete UKB data.

`disease_definitions` Named list of disease definitions (see [create_disease_definition](#)).

`prevalent_sources` Character vector specifying data sources for identifying prevalent (baseline) cases. Self-report is recommended here since participants reporting a disease at baseline clearly had it before enrollment. Default includes all core sources: "ICD10", "ICD9", "Self-report", "Death". "OPCS4" can be added for surgical phenotypes when `opcs4_pattern` is supplied in the disease definition. Also

	supports "CancerRegistry" for UKB cancer registry outcomes, "FirstOccurrence" for UKB First Occurrence fields, and "Algorithm" for UK Biobank algorithmically-defined outcomes.
outcome_sources	Character vector specifying data sources for defining incident outcomes. Self-report is typically excluded here because self-reported diagnosis dates are imprecise (year only) and less reliable for prospective endpoint ascertainment. Default: "ICD10", "ICD9", "Death". "CancerRegistry" can be added for cancer outcomes; "FirstOccurrence" can be added when the extracted dataset includes UKB First Occurrence fields for the disease definition. "OPCS4" can be included when the event of interest is a surgery or procedure-based phenotype.
censor_date	Administrative censoring date (default: "2023-10-31").
baseline_col	Column name for baseline assessment date (default: "p53_i0").
time_skeleton	Optional output from ukb_time_skeleton . When supplied, its baseline_date is used for prevalent/incident classification and its participant-specific followup_end_date is used to calculate default follow-up time for non-cases. The censor_date argument remains the common administrative censoring date used by endpoint extraction.
primary_disease	One or more disease keys used to compute follow-up time and event status (all must be in disease_definitions). If NULL, the first disease in the list is used. For wide output, a single key preserves the legacy outcome_status and outcome_surv_time columns. Multiple keys produce independent <Disease>_status and <Disease>_surv_time columns in one call.
output	Output format: "wide" (default) returns the original data with disease indicator columns; "long" returns case-level records.
include_all	Logical; when output = "long", if TRUE includes the full cohort with non-cases coded as status = 0.
show_flow	Logical; if TRUE and output = "wide", prints a step-by-step participant attrition table in the terminal, including counts before/after each filter and retention rates.
dt_threads	Optional integer. If provided, temporarily sets data.table thread count via setDTthreads() for this function call, and restores the previous thread setting on exit.

Details

This function supports separate source definitions for prevalent case exclusion and outcome ascertainment. This is important because:

- Self-reported conditions at baseline clearly indicate pre-existing disease and should be used for prevalent case identification.
- However, self-reported incident events during follow-up have imprecise dates (year only) and lower validity, making them unsuitable for outcome definition.
- OPCS4 procedure dates are often useful for procedure-defined endpoints or surgical history, but may occur later than the true biological disease onset.

Case classification logic:

- **Prevalent case:** Earliest diagnosis date (from prevalent_sources) $\leq$ baseline date. These participants have outcome_status = NA and outcome_surv_time = NA because they are not at risk for incident disease.
- **Incident case:** Earliest diagnosis date (from outcome_sources) $>$ baseline date
- **Censored:** No diagnosis by end of follow-up (status = 0)

Follow-up time calculation (independently for every selected primary_disease):

- Prevalent case (primary disease): NA (not at risk)
- Incident case: $(\text{diagnosis_date} - \text{baseline_date}) / 365.25$
- Censored: $(\min(\text{death_date}, \text{censor_date}) - \text{baseline_date}) / 365.25$

Value

A data.table with columns:

eid Participant identifier

<Disease>_history 1 if prevalent case (from prevalent_sources), 0 otherwise

<Disease>_incident 1 if incident case (from outcome_sources), 0 otherwise

outcome_status For one primary disease, event indicator (1=event, 0=censored, NA=prevalent case).

outcome_surv_time For one primary disease, follow-up time in years (NA for prevalent cases).

<Disease>_status For multiple primary diseases, an independent event indicator for each disease.

<Disease>_surv_time For multiple primary diseases, independent follow-up time in years for each disease.

calculate_air_pollution

Calculate air pollution exposure averages

Description

Computes averaged air pollution exposures from multiple time points.

Usage

```
calculate_air_pollution(df, pollutants = c("NO2", "PM10", "PM2.5", "NOx"))
```

Arguments

df A data.table containing air pollution columns

pollutants Character vector of pollutants to calculate. Available: "NO2", "PM10", "PM2.5", "NOx"

Value

A data.table with averaged pollution columns

`calculate_blood_pressure`*Calculate blood pressure from multiple readings*

Description

Combines automated and manual BP readings using UK Biobank collection logic: automated readings are primary and manual readings are used as fallback when automated measurements are unavailable. Returns the mean of the two available readings.

Usage

```
calculate_blood_pressure(  
  df,  
  type = c("sbp", "dbp"),  
  prefer = c("auto", "manual")  
)
```

Arguments

<code>df</code>	A data.table containing BP columns
<code>type</code>	Character: "sbp" or "dbp"
<code>prefer</code>	Character: "auto" (default) or "manual", controlling which measurement source is treated as primary when both are available.

Value

A data.table with calculated sbp or dbp column added

`calculate_diet_score` *Calculate diet score*

Description

Computes a simplified healthy diet score based on food frequency questionnaire.

Usage

```
calculate_diet_score(  
  df,  
  components = c("fruit", "vegetable", "fish", "meat", "cereal", "milk"),  
  na_handling = c("strict", "partial")  
)
```

Arguments

df	A data.table containing diet-related columns
components	Character vector of diet components to include. Available: "fruit", "vegetable", "fish", "meat", "cereal", "milk"
na_handling	Character: "strict" (NA if any component missing) or "partial" (calculate from available components, NA only if insufficient data)

Value

A data.table with diet_score column (0-7 scale)

calculate_weights	<i>Calculate IPTW Weights</i>
-------------------	-------------------------------

Description

Calculate inverse probability of treatment weights (IPTW) for causal inference.

Usage

```
calculate_weights(
  data,
  ps_col = "ps",
  treatment,
  weight_type = c("ATE", "ATT", "ATC"),
  stabilized = TRUE,
  truncate = c(0.01, 0.99)
)
```

Arguments

data	A data.table containing propensity scores.
ps_col	Character string specifying the propensity score column name. Default "ps".
treatment	Character string specifying the treatment variable name.
weight_type	Character string specifying weight type: "ATE", "ATT", or "ATC".
stabilized	Logical; whether to use stabilized weights. Default TRUE.
truncate	Numeric vector of length 2 specifying quantiles for weight truncation. Default c(0.01, 0.99).

Details

Weight formulas:

- ATE: $T/PS + (1-T)/(1-PS)$
- ATT: $T + (1-T) * PS/(1-PS)$
- ATC: $T * (1-PS)/PS + (1-T)$

Stabilized weights multiply by the marginal probability of treatment.

Value

A data.table with the original data plus:

weight IPTW weight

classify_metabolites *Classify UK Biobank metabolite names*

Description

Classify metabolite-like names into broad groups used by the metabolomics ORA workflow. Small molecules can be mapped to MetaboAnalyst-compatible names, whereas lipoprotein subclass measures, lipid aggregate measures, and proteins are retained in the mapping table but are not passed to small-molecule ORA by default.

Usage

```
classify_metabolites(metabolites)
```

Arguments

metabolites Character vector of metabolite names.

Value

A data.frame with metabolite, category, and metaboanalyst_name.

Examples

```
classify_metabolites(c("Alanine", "LDL Cholesterol", "Apolipoprotein B"))
```

coef.mediation_result *Extract Coefficients from Mediation Results*

Description

Extract effect estimates from mediation analysis results.

Usage

```
## S3 method for class 'mediation_result'  
coef(object, ...)
```

Arguments

object An object of class "mediation_result".
 ... Additional arguments (unused).

Value

A data.frame with effect estimates.

combine_disease_definitions *Combine Multiple Disease Definitions*

Description

Merges multiple disease definitions into a single composite endpoint definition. Useful for creating MACE (Major Adverse Cardiovascular Events) or similar composite outcomes.

Usage

```
combine_disease_definitions(..., name = "Combined")
```

Arguments

... Disease definition objects to combine.
 name Name for the composite outcome.

Value

A combined disease definition object.

combine_prs_weights *Combine Harmonized PRS Weights into One PLINK2 Score Matrix*

Description

Combine several harmonized PRS definitions into a wide score file. PLINK2 can then calculate all scores during one pass over each genotype file with `--score-col-nums`, instead of rereading the same genotypes for every PRS. Missing variant/score combinations are assigned weight zero.

Usage

```
combine_prs_weights(
  weights,
  score_names = names(weights),
  output = NULL,
  overwrite = FALSE
)
```

Arguments

- weights A named list of ukb_prs_harmonized objects.
- score_names Optional output score names. Defaults to names(weights).
- output Optional tab-delimited output path.
- overwrite Overwrite an existing output.

Value

A ukb_prs_weight_matrix data.frame with ID, A1, one column per score, and target variant meta-data.

compare_data_sources	<i>Compare Case Counts Across Data Sources</i>
----------------------	--

Description

Generates a summary table comparing case counts from different data sources. Useful for methods sections and sensitivity analysis planning.

Usage

```
compare_data_sources(dt, disease_definitions, baseline_col = "p53_i0")
```

Arguments

- dt A data.table containing complete UKB data.
- disease_definitions Named list of disease definitions.
- baseline_col Column name for baseline date.

Value

A data.table with case counts by source and combination.

compute_protein_ppi_metrics

Compute topological metrics for a PPI network

Description

A thin wrapper around TCMDATA::compute_nodeinfo().

Usage

```
compute_protein_ppi_metrics(
  ppi,
  weight_attr = "score",
  normalize = FALSE,
  seed = 42
)
```

Arguments

ppi	An igraph object, a list returned by get_protein_ppi(), or a list containing a graph element.
weight_attr	Character. Edge attribute used as weight. Default is "score".
normalize	Logical. Whether to normalize betweenness and closeness. Default is FALSE.
seed	Numeric random seed used by the EPC calculation. Default is 42.

Value

An igraph object with additional vertex attributes.

confint.mediation_result

Confidence Intervals for Mediation Results

Description

Extract confidence intervals from mediation analysis results.

Usage

```
## S3 method for class 'mediation_result'
confint(object, parm = NULL, level = 0.95, ...)
```

Arguments

object	An object of class "mediation_result".
parm	Character vector of effect names. If NULL, returns all effects.
level	Confidence level. Default 0.95.
...	Additional arguments (unused).

Value

A matrix with lower and upper confidence limits.

convert_gwas_datatype *Convert Common GWAS Genotype File Formats*

Description

Build or execute a PLINK2 command that converts between common GWAS genotype formats. Supported high-level formats are PLINK 1 binary (bed/bim/fam), PLINK 2 binary (pgen/pvar/psam), BGEN, VCF, and BCF.

The function is a dry run by default. It does not install PLINK2, submit a DNAnexus job, or copy participant data into the package. With `execute = TRUE`, it calls an existing PLINK2 binary through [ukb_run_plink2](#).

Usage

```
convert_gwas_datatype(
  input,
  to,
  output_prefix,
  from = c("auto", "bed", "pgen", "bgen", "vcf", "bcf"),
  sample_file = NULL,
  bgen_ref = c("ref-first", "ref-last", "ref-unknown"),
  vcf_dosage = NULL,
  bgen_version = c("1.2", "1.3", "1.1"),
  bgen_bits = 16L,
  export_ref_first = TRUE,
  vcf_bgz = TRUE,
  vcf_dosage_export = NULL,
  plink2_args = character(),
  execute = FALSE,
  executable = "plink2",
  require_rap = TRUE,
  overwrite = FALSE,
  timeout = 86400
)
```

Arguments

input	Input path. For BED and PGEN filesets this may be the fileset prefix or any primary component path; for BGEN, VCF, and BCF it is the data-file path.
to	Output format: "bed", "pgen", "bgen", "vcf", or "bcf".
output_prefix	Output prefix without a genotype-format extension.
from	Input format or "auto". Automatic detection uses the supplied extension and, for a prefix, companion files visible in the current filesystem.
sample_file	Optional BGEN .sample file or VCF/BCF .psam file.
bgen_ref	Reference-allele interpretation for BGEN input.
vcf_dosage	Optional VCF/BCF dosage field name, for example "DS".
bgen_version	BGEN export version.
bgen_bits	Integer probability precision for BGEN export. PLINK2 defaults to 16 bits.
export_ref_first	Put REF first when exporting BGEN.
vcf_bgz	Block-gzip VCF output.
vcf_dosage_export	Optional VCF/BCF dosage export mode: "GP", "DS", "DS-force", "DS-only", "HDS", or "HDS-force".
plink2_args	Additional PLINK2 argument tokens, such as c("--maf", "0.01", "--geno", "0.05", "--threads", "16"). Input/output flags managed by this function are rejected. Use ukb_plan_plink2 for a fully custom command.
execute	Execute the conversion. Defaults to FALSE.
executable	PLINK2 executable path or command name.
require_rap	Require a RAP-like environment before execution.
overwrite	Permit PLINK2 to replace an existing expected output.
timeout	Command timeout in seconds.

Value

With execute = FALSE, a `ukb_gwas_conversion_plan`; otherwise a `ukb_plink2_run`.

`create_baseline_table` *Create a baseline table comparing cases and controls under different conditions.*

Description

Create a baseline table comparing cases and controls under different conditions.

Usage

```
create_baseline_table(
  data,
  case_col,
  factor_cols = NULL,
  continuous_cols = NULL,
  test = FALSE
)
```

Arguments

<code>data</code>	a <code>data.table</code> containing the baseline characteristics and case/control status.
<code>case_col</code>	the name of the column indicating case/control status (1 for cases, 0 for controls).
<code>factor_cols</code>	a vector of column names that are factors (categorical variables)
<code>continuous_cols</code>	a vector of column names that are continuous variables.
<code>test</code>	whether to perform statistical tests comparing cases and controls for each variable (default: FALSE).

Value

a list containing table one information.

References

<https://github.com/kaz-yos/tableone>

create_disease_definition

Create Disease Definition Object

Description

Helper function to create a standardized disease definition object containing ICD-10/ICD-9 patterns, self-report codes, UK Biobank First Occurrence fields, and optionally a UK Biobank algorithmically-defined outcome date field.

Usage

```
create_disease_definition(
  name = NULL,
  icd10_pattern = NULL,
  icd9_pattern = NULL,
  sr_codes = NULL,
  death_icd10 = NULL,
```

```

    opcs4_pattern = NULL,
    first_occurrence_fields = NULL,
    first_occurrence_source_fields = NULL,
    cancer_icd10_pattern = NULL,
    cancer_histology = NULL,
    cancer_behaviour = NULL,
    algo_date_field = NULL,
    algo_source_field = NULL,
    icd10 = NULL,
    icd9 = NULL,
    self_report = NULL
)

```

Arguments

name	Full disease name (e.g., "Aortic Aneurysm"). If NULL, defaults to "Custom disease".
icd10_pattern	Regular expression pattern for ICD-10 codes (optional).
icd9_pattern	Regular expression pattern for ICD-9 codes (optional).
sr_codes	Integer vector of UKB self-report illness codes (optional).
death_icd10	Optional regular expression pattern (or code vector) for death-cause ICD-10 matching. If NULL, defaults to icd10_pattern.
opcs4_pattern	Optional regular expression pattern (or code vector) for OPCS4 operative procedure matching. If NULL, operative procedures are not used in case ascertainment.
first_occurrence_fields	Optional integer vector of UK Biobank First Occurrence date field IDs. These fields are generated for 3-character ICD-10 codes in Category 1712, e.g. 131298 for I21 (acute myocardial infarction) and 130708 for E11 (type 2 diabetes). The source field is normally the next field ID and is inferred automatically.
first_occurrence_source_fields	Optional integer vector of First Occurrence source field IDs. If NULL, uses first_occurrence_fields + 1.
cancer_icd10_pattern	Optional regular expression pattern for UKB cancer registry ICD-10 codes (Field 40006).
cancer_histology	Optional integer vector of tumour histology codes (Field 40011) to retain.
cancer_behaviour	Optional integer vector of tumour behaviour codes (Field 40012) to retain. Use 3L for malignant tumours.
algo_date_field	Integer vector. UKB field ID(s) for algorithmically-defined outcome dates (Category 42). For example, 42016 for COPD and 42014 for Asthma. The corresponding data column can be p{field}_i0 or p{field}. Records with date 1900-01-01 are treated as unknown and excluded.

algo_source_field	Optional integer vector paired with algo_date_field. For example, 42017 for COPD source and 42015 for Asthma source. Use NA when a date field has no source field.
icd10	Deprecated alias of icd10_pattern.
icd9	Deprecated alias of icd9_pattern.
self_report	Deprecated alias of sr_codes.

Value

A list containing the disease definition parameters.

`create_imputation_list`*Create an imputationList Object*

Description

Converts a list of data.frames to a imputationList object for use with mitools functions.

Usage

```
create_imputation_list(datasets, validate = TRUE)
```

Arguments

datasets	A list of data.frames (imputed datasets).
validate	Logical; whether to validate that all datasets have the same structure. Default TRUE.

Value

An imputationList object.

`create_medication_definition`*Create a medication definition object*

Description

Helper for defining medication code sets from UK Biobank self-reported treatment/medication fields. The first implementation focuses on field 20003 arrays and intentionally stores only medication codes and classes, not copied source codelist descriptions.

Usage

```
create_medication_definition(  
  name,  
  codes,  
  source = "Self-report 20003",  
  field_id = 20003L,  
  medication_class = NULL,  
  match_type = "exact"  
)
```

Arguments

<code>name</code>	Medication definition name.
<code>codes</code>	Character or numeric medication codes.
<code>source</code>	Source label. Defaults to "Self-report 20003".
<code>field_id</code>	UK Biobank field ID. Defaults to 20003.
<code>medication_class</code>	Optional medication class label.
<code>match_type</code>	Matching mode. Defaults to "exact".

Value

A list describing the medication definition.

Examples

```
bp <- create_medication_definition("Any BP medication", c(1, 2, 3))
```

```
estimate_propensity_score
```

Estimate Propensity Score

Description

Calculate propensity scores using logistic regression or gradient boosting.

Usage

```
estimate_propensity_score(
  data,
  treatment,
  covariates,
  method = c("logistic", "gbm"),
  formula = NULL
)
```

Arguments

<code>data</code>	A data.frame or data.table containing all variables.
<code>treatment</code>	Character string specifying the treatment variable name (binary 0/1).
<code>covariates</code>	Character vector of covariate names used to estimate propensity scores.
<code>method</code>	Character string specifying the estimation method: "logistic" (default) or "gbm".
<code>formula</code>	Optional custom formula. If NULL, formula is built from treatment and covariates.

Value

A data.table with the original data plus:

ps Propensity score (probability of treatment)

```
extract_cases_by_source
```

Extract Cases by Specified Data Sources

Description

Flexibly extracts disease cases using user-specified data sources. Enables main analysis with strict case definitions (e.g., ICD-10 only) and sensitivity analyses with broader definitions (e.g., all sources).

Usage

```
extract_cases_by_source(
  dt,
  disease_definitions,
  sources = c("ICD10", "ICD9", "Self-report", "Death"),
  censor_date = as.Date("2023-10-31"),
  baseline_col = "p53_i0"
)
```

Arguments

<code>dt</code>	A data.table or data.frame containing complete UKB data.
<code>disease_definitions</code>	Named list of disease definitions.
<code>sources</code>	Character vector specifying data sources to include. Valid options: "ICD10", "ICD9", "Self-report", "Death", "OPCS4", "CancerRegistry", "FirstOccurrence", "Algorithm". "OPCS4" uses hospital inpatient summary operations (p41272 + p41282_a*) and requires opcs4_pattern in the disease definition. "Algorithm" uses UK Biobank algorithmically-defined outcomes (Category 42) which combine multiple data sources with high positive predictive value. Requires algo_date_field in the disease definition. If algo_source_field is also provided, output diagnosis_source is refined as "Algorithm_<source_code>". "FirstOccurrence" uses UK Biobank First Occurrence date fields (Category 1712, p13xxxx) and requires first_occurrence_fields in the disease definition. "CancerRegistry" uses UK Biobank cancer register records (p40006_i* + p40005_i*) and requires cancer_icd10_pattern in the disease definition.
<code>censor_date</code>	Administrative censoring date.
<code>baseline_col</code>	Column name for baseline assessment date.

Details

This function is designed for epidemiological studies requiring:

- Main analysis with hospital-confirmed diagnoses only
- Sensitivity analyses including self-reported conditions
- Procedure-augmented definitions for surgical phenotypes using OPCS4
- Cancer registry ascertainment for malignant neoplasm endpoints
- First Occurrence fields for UKB's pre-mapped 3-character ICD-10 outcomes
- Source-specific case counts for methods reporting
- UK Biobank algorithmically-defined outcomes for validated case ascertainment

The "Algorithm" source reads date fields from UK Biobank Category 42 (Algorithmically-defined outcomes). These are pre-computed by the UK Biobank outcome adjudication group, combining self-report, hospital admissions, and death records with high positive predictive value. Records with date 1900-01-01 are excluded (date unknown). If a source field is available in the definition, it is propagated into diagnosis_source as "Algorithm_<source_code>".

The "FirstOccurrence" source reads singular UKB fields such as p131298_i0 or p131298 for I21 first reported. Values with UKB special date coding 819 (1900-01-01, 1901-01-01, 1902-02-02, 1903-03-03, 1909-09-09, and 2037-07-07) are excluded.

Value

A data.table with case-level survival data from specified sources.

extract_diabetes_subtype_baseline
<i>Extract Baseline Diabetes Subtypes (T1DM/T2DM) with HbA1c Support</i>

Description

Extracts baseline prevalent Type 1 and Type 2 diabetes using existing source-based disease history logic, and optionally augments Type 2 classification using baseline HbA1c.

Usage

```
extract_diabetes_subtype_baseline(  
  dt,  
  disease_definitions = NULL,  
  sources = c("ICD10", "ICD9", "Self-report"),  
  baseline_col = "p53_i0",  
  hba1c_col = "p30750_i0",  
  hba1c_threshold = 48,  
  include_hba1c = TRUE  
)
```

Arguments

dt	A data.table or data.frame containing UKB data.
disease_definitions	Named list of disease definitions. If NULL, uses get_predefined_diseases .
sources	Character vector specifying sources for baseline history. Options: "ICD10", "ICD9", "Self-report", "Death", "CancerRegistry", "FirstOccurrence", "Algorithm".
baseline_col	Column name for baseline date. Default: "p53_i0".
hba1c_col	Column name for baseline HbA1c (mmol/mol). Default: "p30750_i0".
hba1c_threshold	Numeric threshold for diabetes by HbA1c. Default: 48 mmol/mol (equivalent to 6.5 percent).
include_hba1c	Logical. If TRUE (default), HbA1c is used to augment T2DM classification.

Details

This is a baseline classification helper and does not redefine incident event logic. Type 1 has priority when both T1 and T2 evidence are present.

Value

A data.table with columns:

eid Participant identifier

T1DM_history Baseline prevalent T1DM from selected sources (0/1)

T2DM_history Baseline prevalent T2DM from selected sources (0/1)

diabetes_hba1c Baseline HbA1c diabetes flag (0/1/NA)

T2DM_history_enhanced T2DM from source history OR HbA1c criterion (0/1)

Diabetes_history Any baseline diabetes (T1DM or enhanced T2DM) (0/1)

diabetes_subtype "Type1", "Type2", or "No_diabetes"

extract_disease_diagnosis

Extract participant-level disease diagnosis status

Description

Defines whether each participant has a selected disease using one or more UK Biobank evidence sources. This is the recommended public helper when the goal is disease ascertainment rather than construction of a full survival cohort. For survival-ready endpoints, use [build_survival_dataset](#).

Usage

```
extract_disease_diagnosis(
  dt,
  disease,
  disease_definitions = NULL,
  sources = c("ICD10", "ICD9", "Self-report", "Death"),
  censor_date = as.Date("2023-10-31"),
  baseline_col = "p53_i0",
  include_all = TRUE
)
```

Arguments

dt A data.table or data.frame containing UKB data.

disease Character vector of disease keys or disease names.

disease_definitions Optional named list of disease definitions. If NULL, [get_predefined_diseases](#) is used.

<code>sources</code>	Character vector of evidence sources. Valid options are "ICD10", "ICD9", "Self-report", "Death", "OPCS4", "CancerRegistry", "FirstOccurrence", and "Algorithm".
<code>censor_date</code>	Administrative censoring date.
<code>baseline_col</code>	Column name for baseline assessment date.
<code>include_all</code>	Logical. If TRUE, return one row per participant per disease, including non-cases. If FALSE, return diagnosed participants only.

Value

A data.table with participant-level diagnosis status, first diagnosis date, diagnosis source, prevalent and incident indicators, and survival fields returned by [extract_cases_by_source](#) where available.

`extract_disease_history`

Extract Disease History (Prevalent Cases) for Covariates

Description

Extracts prevalent case status (diagnosed before baseline) for specified diseases. Designed for use as covariates in sensitivity analyses or covariate adjustment. Returns a wide-format table with one binary column per disease.

Usage

```
extract_disease_history(
  dt,
  diseases,
  disease_definitions = NULL,
  sources = "ICD10",
  baseline_col = "p53_i0"
)
```

Arguments

<code>dt</code>	A data.table or data.frame containing complete UKB data.
<code>diseases</code>	Character vector of disease names to extract. Must match keys in <code>disease_definitions</code> or predefined disease names.
<code>disease_definitions</code>	Named list of disease definitions. If NULL, uses get_predefined_diseases .
<code>sources</code>	Character vector specifying data sources. Default: "ICD10". Options: "ICD10", "ICD9", "Self-report", "Death", "OPCS4", "CancerRegistry", "FirstOccurrence", "Algorithm".
<code>baseline_col</code>	Column name for baseline assessment date.

Details

This function is specifically designed for extracting covariate data in epidemiological studies. Common use cases:

- Adjusting for baseline comorbidities in Cox regression
- Sensitivity analyses with different case definitions
- Creating propensity score matching variables

The function only returns history (prevalent) columns, not incident columns, to clearly separate covariate extraction from outcome definition.

Value

A data.table with columns:

eid Participant identifier

Disease_history 1 if prevalent case, 0 otherwise (one column per disease)

extract_disease_history_sensitivity

Extract Disease History with Multiple Source Comparisons

Description

Extracts prevalent case status from multiple data source combinations simultaneously for sensitivity analysis comparison. Returns a table with separate columns for each source definition.

Usage

```
extract_disease_history_sensitivity(
  dt,
  diseases,
  disease_definitions = NULL,
  baseline_col = "p53_i0"
)
```

Arguments

dt	A data.table or data.frame containing complete UKB data.
diseases	Character vector of disease names to extract.
disease_definitions	Named list of disease definitions.
baseline_col	Column name for baseline date.

Value

A data.table with columns:

eid Participant identifier

Disease_history_ICD10 Prevalent case using ICD-10 only

Disease_history_hospital Prevalent case using ICD-10 + ICD-9

Disease_history_all Prevalent case using all sources

extract_medications	<i>Extract medication use from UKB drug fields</i>
---------------------	--

Description

Processes medication fields (6177 for male, 6153 for female) to extract specific medication categories.

Usage

```
extract_medications(
  df,
  medications = c("cholesterol", "blood_pressure", "insulin")
)
```

Arguments

df	A data.table containing medication columns (p6177_i0, p6153_i0)
medications	Character vector of medications to extract. Available: "cholesterol", "blood_pressure", "insulin"

Value

A data.table with binary medication columns added (1=Yes, 0=No, NA=Missing)

extract_self_report_medications	<i>Extract self-reported medication indicators from field 20003</i>
---------------------------------	---

Description

Matches UK Biobank treatment/medication code arrays (p20003_i*_a*) against predefined or user-supplied medication definitions and appends binary participant-level medication indicators.

Usage

```
extract_self_report_medications(
  data,
  medications = NULL,
  medication_definitions = get_predefined_medications(),
  id_col = "eid",
  instance = 0,
  prefix = "med20003",
  missing_as_zero = TRUE,
  return_long = FALSE
)
```

Arguments

<code>data</code>	A data.frame or data.table containing field 20003 array columns.
<code>medications</code>	Optional medication definition names to extract. If NULL, all predefined definitions are used.
<code>medication_definitions</code>	Named list of medication definitions. Defaults to <code>get_predefined_medications()</code> .
<code>id_col</code>	Participant identifier column.
<code>instance</code>	Optional UKB assessment instance. If NULL, all available instances are searched.
<code>prefix</code>	Prefix for output variable names.
<code>missing_as_zero</code>	Logical. If TRUE, participants with no valid 20003 entries are coded as 0; otherwise they are coded as NA.
<code>return_long</code>	Logical. If TRUE, return one row per participant and medication definition instead of appending wide columns.

Value

A data.table.

Examples

```
dat <- data.frame(
  eid = 1:3,
  p20003_i0_a0 = c("1140883066", "1140874686", NA),
  p20003_i0_a1 = c(NA, "1140851690", NA)
)
extract_self_report_medications(dat, medications = c("Insulin", "Metformin"))
```

fit_mi_models

Fit Regression Models on Multiple Imputed Datasets

Description

Fits the specified regression model on each imputed dataset.

Usage

```
fit_mi_models(
  datasets,
  formula,
  model_type = c("lm", "logistic", "poisson", "cox", "negbin"),
  family = NULL,
  ...
)
```

Arguments

datasets	A list of data.frames or an imputationList object.
formula	A formula specifying the model.
model_type	Character string specifying the model type.
family	A family object for GLM (optional).
...	Additional arguments passed to the model fitting function.

Value

A list of fitted model objects.

get_death_dates

Extract Death Dates for All Deceased Participants

Description

Returns death dates for all deceased participants, used for censoring in survival analysis.

Usage

```
get_death_dates(dt)
```

Arguments

dt	A data.table or data.frame containing UKB data.
----	---

Value

A data.table with columns: eid, death_date.

get_disease_catalog	<i>Query the built-in disease code catalog</i>
---------------------	--

Description

Returns a source-aware disease code catalog containing curated UKBAnalytica disease definitions and Pomegranate-derived UK Biobank phenotype coding definitions. This function returns tabular code metadata; it does not change the default behavior of `get_predefined_diseases()`.

Usage

```
get_disease_catalog(  
  source = c("all", "curated", "pomegranate"),  
  disease = NULL,  
  code_system = NULL,  
  supported_only = FALSE  
)
```

Arguments

source	Character. One of "all", "curated", or "pomegranate".
disease	Optional disease name or definition ID pattern.
code_system	Optional code system filter, such as "ICD-10" or "self-report illness".
supported_only	Logical. If TRUE, keep only catalog rows currently supported by UKBAnalytica disease parsers.

Value

A data.frame.

Examples

```
copd_codes <- get_disease_catalog(disease = "copd")  
head(copd_codes)
```

get_field_info	<i>Get one UK Biobank field's metadata</i>
----------------	--

Description

Convenience wrapper around `get_field_metadata()` for a single UKB `field_id`. This is the simplest way to ask "what does field 4080 correspond to?" and get a one-row metadata table back in R.

Usage

```
get_field_info(
  field_id,
  ukb_data_dict = NULL,
  dataset = NULL,
  fields_df = NULL,
  entity = "participant",
  live = FALSE,
  timeout = 30
)
```

Arguments

field_id	A single UKB numeric field ID.
ukb_data_dict	Optional path to a Data_Dictionary_Showcase.tsv file or equivalent UKB metadata export available in the current session.
dataset	Optional RAP .dataset file name. Used only when fields_df is NULL and RAP field metadata should be retrieved live.
fields_df	Optional data.frame returned by rap_list_fields(). This is useful for offline testing or when you already cached the RAP field list.
entity	RAP dataset entity. Defaults to "participant".
live	Logical. If TRUE, fetch the field page from the public UK Biobank Showcase website and parse the displayed metadata for this field_id.
timeout	Timeout in seconds used for the live web request.

Value

A one-row data.frame when the field is found.

get_field_metadata	<i>Get structured UK Biobank field metadata</i>
--------------------	---

Description

Returns a structured data.frame of UK Biobank field metadata. When ukb_data_dict is supplied, the function reads a UK Biobank data dictionary metadata file available in the current session and standardizes common metadata columns. When fields_df or a RAP dataset is supplied, the function also records the approved RAP field names available in the current project.

This is intended to be a simple entry point for users who want to inspect UKB field metadata in R before planning an extraction.

Usage

```

get_field_metadata(
  field_id = NULL,
  query = NULL,
  ukb_data_dict = NULL,
  dataset = NULL,
  fields_df = NULL,
  entity = "participant"
)

```

Arguments

field_id	Optional UKB numeric field IDs to keep.
query	Optional keyword used to filter the metadata table. The keyword is matched against the title, description, category, and RAP field names.
ukb_data_dict	Optional path to a Data_Dictionary_Showcase.tsv file or equivalent UKB metadata export available in the current session.
dataset	Optional RAP .dataset file name. Used only when fields_df is NULL and RAP field metadata should be retrieved live.
fields_df	Optional data.frame returned by rap_list_fields(). This is useful for offline testing or when you already cached the RAP field list.
entity	RAP dataset entity. Defaults to "participant".

Value

A data.frame with one row per UKB field and standardized metadata columns. When RAP field metadata is available, the result also includes the matching RAP column names and the number of approved RAP columns per field.

```
get_medication_catalog
```

Query the built-in medication code catalog

Description

Returns a medication code catalog containing UKBAnalytica curated medication definitions and UK Biobank official coding 4 entries for field 20003.

Usage

```
get_medication_catalog(medication = NULL, medication_class = NULL)
```

Arguments

medication	Optional medication name, ID, or code pattern.
medication_class	Optional medication class filter.

Value

A data.frame.

Examples

```
metformin <- get_medication_catalog("metformin")
head(metformin)
```

```
get_pomegranate_diseases
```

Get Pomegranate-derived disease definitions

Description

Converts the Pomegranate-derived rows in the disease catalog into UKBAnalytica disease definition objects. Only code systems currently supported by the package parsers are used by default; GP and medication rows remain available through `get_disease_catalog()`.

Usage

```
get_pomegranate_diseases(disease = NULL, supported_only = TRUE)
```

Arguments

`disease` Optional disease name or definition ID pattern.

`supported_only` Logical. If TRUE, use only rows supported by current UKBAnalytica disease parsers.

Value

A named list of disease definition objects.

Examples

```
pom <- get_pomegranate_diseases("asthma")
names(pom)
```

`get_pomegranate_source_manifest`*Get the Pomegranate source manifest*

Description

Returns source provenance for the built-in Pomegranate resources, including the GitHub YAML commit used for the canonical disease catalog and the portal CSV retained for audit.

Usage

```
get_pomegranate_source_manifest()
```

Value

A data.frame.

Examples

```
get_pomegranate_source_manifest()
```

`get_predefined_diseases`*Get Predefined Disease Definitions*

Description

Returns a list of commonly used cardiovascular and metabolic disease definitions with validated ICD-10, ICD-9, and self-report code mappings.

Usage

```
get_predefined_diseases(  
  source = c("curated", "pomegranate", "both"),  
  merge_type = c("intersection", "union"),  
  disease = NULL,  
  supported_only = TRUE  
)
```

Arguments

source	Definition source. "curated" returns the original manually curated UKBAnalytics definitions. "pomegranate" returns definitions converted from the built-in Pomegranate-derived disease catalog. "both" returns diseases that can be matched between both sources, with standardized curated names and either intersected or unioned source definitions depending on merge_type.
merge_type	Merge strategy for source = "both". "intersection" keeps codes supported by both curated and Pomegranate definitions. "union" combines codes from both definitions.
disease	Optional disease key or name pattern used to subset the returned definition list.
supported_only	Logical. For Pomegranate-derived definitions, keep only code systems currently supported by UKBAnalytics parsers.

Details

Included diseases:

AA Aortic Aneurysm (I71, 441)

TAA Thoracic Aortic Aneurysm

AAA Abdominal Aortic Aneurysm

CVD Cardiovascular Disease

MI Myocardial Infarction

HF Heart Failure

Stroke Stroke (ischemic and hemorrhagic)

Hypertension Essential and secondary hypertension

Diabetes Diabetes Mellitus (all types)

T1DM Type 1 Diabetes Mellitus

T2DM Type 2 Diabetes Mellitus

Vascular_Disease Peripheral vascular disease

Arrhythmia Broad cardiac arrhythmia endpoint including OPCS4 procedures

Atrial_Fibrillation Atrial arrhythmia / atrial fibrillation-flutter

Ventricular_Arrhythmia Ventricular arrhythmia endpoint

AV_Block Atrioventricular conduction block

Intraventricular_Block Intraventricular conduction block

SVT Supraventricular tachycardia

Lung_Cancer Lung cancer using ICD-10/death and cancer registry

Additional chronic diseases Common respiratory, renal, gastrointestinal, neurologic, psychiatric, eye, skin, musculoskeletal, and cancer endpoints used in UKB epidemiology workflows

Value

A named list of disease definition objects.

`get_predefined_medications`*Get predefined UK Biobank medication definitions*

Description

Returns curated field-20003 medication code sets for common self-reported treatment groups. These definitions are designed for baseline covariate derivation and sensitivity analyses, and are separate from disease endpoint definitions returned by [get_predefined_diseases\(\)](#).

Usage

```
get_predefined_medications()
```

Value

A named list of medication definition objects.

Examples

```
meds <- get_predefined_medications()
names(meds)
```

`get_protein_ppi`*Retrieve a STRING PPI network for proteomics hits*

Description

Convert protein identifiers to gene symbols and retrieve a protein-protein interaction network from STRING via `clusterProfiler::getPPI()`.

Usage

```
get_protein_ppi(
  proteins,
  protein_col = NULL,
  from_type = "SYMBOL",
  mapping_table = NULL,
  mapping_protein_col = "protein",
  mapping_symbol_col = "gene_symbol",
  organism_db = "org.Hs.eg.db",
  taxID = 9606,
  required_score = NULL,
  network_type = "functional",
  add_nodes = 0,
  show_query_node_labels = 0,
```

```

    output = c("igraph", "data.frame"),
    input_format = c("auto", "rap_olink", "symbol", "raw")
)

```

Arguments

proteins	A character vector of protein identifiers, or a data.frame containing a protein identifier column.
protein_col	Optional column name when proteins is a data.frame.
from_type	Character string describing the input identifier type for Bioconductor-based mapping. Default is "SYMBOL".
mapping_table	Optional data.frame containing custom protein-to-symbol mappings.
mapping_protein_col	Column name in mapping_table containing protein identifiers. Default is "protein".
mapping_symbol_col	Column name in mapping_table containing gene symbols. Default is "gene_symbol".
organism_db	Character string naming the OrgDb package. Default is "org.Hs.eg.db".
taxID	NCBI taxon identifier passed to clusterProfiler::getPPI(). Default is 9606.
required_score	Optional STRING score cutoff passed to clusterProfiler::getPPI().
network_type	STRING network type. One of "functional" or "physical". Default is "functional".
add_nodes	Number of partner nodes to add in STRING. Default is 0.
show_query_node_labels	Passed to clusterProfiler::getPPI(). Default is 0.
output	One of "igraph" or "data.frame". Default is "igraph".
input_format	Input naming convention passed to protein_to_gene_symbol() .

Value

A list with components source, gene_symbols, mapping, and ppi.

get_ukb_demo_colnames *Get column names of the synthetic UK Biobank-style demo dataset*

Description

Returns the column names generated by `ukb_demo()`. This is useful for documentation examples that need RAP-style toy column names.

Usage

```
get_ukb_demo_colnames()
```

Value

A character vector of original demo-data column names.

Examples

```
get_ukb_demo_colnames()
```

get_variable_info	<i>Get information about available variables</i>
-------------------	--

Description

Returns a data.frame describing all predefined variables available for preprocessing.

Usage

```
get_variable_info(category = "all")
```

Arguments

category	Character. Filter by category: • "all": All variables (default) • "demographics", "anthropometrics", "lifestyle", "socioeconomic", "blood_pressure", "medications", "biomarkers", "pollution", "diet"
----------	---

Value

A data.frame with variable information

get_variable_set	<i>Get one curated UK Biobank variable set</i>
------------------	--

Description

Get one curated UK Biobank variable set

Usage

```
get_variable_set(set, output = c("data.frame", "field_id", "ukb_col"))
```

Arguments

set	Set name.
output	Output format. "data.frame" returns the full manifest, "field_id" returns unique UKB field IDs, and "ukb_col" returns UKB column stems such as p31 or p4080_i0_a0.

Value

A data.frame or character/integer vector.

Examples

```
get_variable_set("clinical_core")
get_variable_set("air_pollution", output = "field_id")
```

get_variable_sets	<i>Curated UK Biobank variable sets for extraction</i>
-------------------	--

Description

Returns curated UKB field groups for common analysis domains. These sets are intended for field discovery and RAP extraction, not for automatic preprocessing. Use [preprocess_baseline\(\)](#) only for variables documented by [get_variable_info\(\)](#).

Usage

```
get_variable_sets(set = NULL, category = NULL)
```

Arguments

set	Optional set name, such as "clinical_core", "air_pollution", or "family_history". If NULL, returns all rows.
category	Optional broad category filter.

Value

A data.frame with one row per curated variable.

Examples

```
vars <- get_variable_sets("air_pollution")
unique(vars$field_id)
```

harmonize_prs_weights *Harmonize PRS Weights to a Target Variant Map*

Description

Align prepared effect alleles to target genotype alleles before PLINK2 scoring. Matching can use variant ID, chromosome/position, or ID followed by a position fallback. Direct, swapped, strand, and strand-swapped SNPs are supported. Indels are retained only when their alleles match directly or in swapped order; this function does not normalize or left-align indels.

Input and target genome builds must be declared and must match. This function deliberately does not perform liftOver. Palindromic SNPs are dropped by default; frequency inference requires both input effect-allele frequency and target allele-1 frequency.

Usage

```
harmonize_prs_weights(
  weights,
  variant_map,
  target_id_col = "ID",
  target_chr_col = "CHR",
  target_pos_col = "POS",
  target_allele1_col = "ALT",
  target_allele2_col = "REF",
  target_allele1_freq_col = NULL,
  target_build,
  match_by = c("auto", "id", "position"),
  palindromic = c("drop", "infer_by_af", "keep"),
  af_tolerance = 0.1,
  output = NULL,
  overwrite = FALSE
)
```

Arguments

weights	A <code>ukb_prs_weights</code> object from <code>prepare_prs_weights()</code> .
variant_map	A <code>data.frame</code> or delimited target variant-map path.
target_id_col, target_chr_col, target_pos_col	Target variant columns.
target_allele1_col, target_allele2_col	Target allele columns. For a REF/ALT map, allele 1 would commonly be ALT and allele 2 REF.
target_allele1_freq_col	Optional frequency column for target allele 1.
target_build	Target genome build, "GRCh37" or "GRCh38".
match_by	"auto" uses ID first and position as a fallback; "id" or "position" restrict matching to one key.

palindromic	"drop", "infer_by_af", or "keep". The last option trusts literal allele orientation and is not recommended without external justification.
af_tolerance	Maximum absolute frequency difference for palindromic inference. Exactly one orientation must pass this threshold.
output	Optional normalized PLINK2 weight-file path.
overwrite	Overwrite an existing output.

Value

A `ukb_prs_harmonized` data.frame. Attributes `harmonization_qc` and `harmonization_exclusions` record decisions.

`integrate_gp_results` *Integrate GP Diagnoses and Registration Coverage*

Description

Creates one row per participant and disease. A matched GP code is assigned `gp_case = 1`; a non-match is assigned `0` only when registration coverage is confirmed; otherwise it remains NA.

Usage

```
integrate_gp_results(
  gp_diagnoses,
  gp_coverage,
  participants = NULL,
  diseases = NULL,
  gp_observability = NULL
)
```

Arguments

<code>gp_diagnoses</code>	Output from summarise_gp_diagnoses() .
<code>gp_coverage</code>	Output from summarise_gp_coverage() .
<code>participants</code>	Optional participant vector or data.frame with <code>eid</code> .
<code>diseases</code>	Optional disease names. Required when no diagnosis rows are present.
<code>gp_observability</code>	Optional output from assess_gp_observability() . When supplied, unmatched participants receive <code>gp_case = 0</code> only when <code>control_eligible</code> is TRUE.

Value

A participant-by-disease GP phenotype data.table.

`load_pomegranate_portal_coding`*Load the Pomegranate portal coding evidence table*

Description

Loads a long-form Pomegranate portal extraction from a user-supplied local CSV or CSV.GZ file for audit and traceability. The canonical Pomegranate disease catalog used by `get_disease_catalog(source = "pomegranate")` is built into the package from the public GitHub YAML algorithms; the portal audit table is not required for endpoint construction and is not shipped in the CRAN build.

Usage

```
load_pomegranate_portal_coding(path = NULL)
```

Arguments

`path` Path to a local Pomegranate portal CSV or CSV.GZ file.

Value

A data.frame.

`load_ukb_medication_coding`*Load UK Biobank field 20003 medication coding*

Description

Loads the UK Biobank coding 4 table used by field 20003 (treatment/medication code). This table is included as a lightweight reference so users can inspect the meaning of medication codes used by `get_predefined_medications()`.

Usage

```
load_ukb_medication_coding(path = NULL)
```

Arguments

`path` Optional path to a local coding 4 TSV file. If NULL, the package copy in `inst/extdata/ukb_coding4_20003.tsv` is used.

Value

A data.frame with columns coding and meaning.

Examples

```
coding4 <- load_ukb_medication_coding()
head(coding4)
```

```
load_ukb_metabolite_panel
```

Load the bundled UK Biobank non-ratio metabolite panel

Description

Read the metabolite metadata table bundled in inst/extdata. The current file contains UK Biobank Nightingale non-ratio metabolite names, field IDs, and RAP-style column names. It is mainly intended as a helper for examples, tests, and metabolite-name checking.

Usage

```
load_ukb_metabolite_panel(file = NULL, file_encoding = "UTF-16LE")
```

Arguments

file	Optional path to a metabolite panel file. If NULL, the bundled metabolites_non_ratio.txt file is used.
file_encoding	Character file encoding. Default is "UTF-16LE" because the bundled table is UTF-16 little-endian encoded.

Value

A data.frame with columns such as Description, UKB_ID, and meta_ID.

Examples

```
panel <- load_ukb_metabolite_panel()
head(panel)
```

```
load_ukb_prs
```

Load Published UK Biobank PRS Fields

Description

Select already-dispensated PRS fields from a participant table, or extract them from the current RAP dataset through [rap_extract_pheno\(\)](#). Enhanced requests also include field 26200 by default. No genotype data are read and no participant-level data are stored in the package.

Usage

```
load_ukb_prs(
  data = NULL,
  trait = NULL,
  field_id = NULL,
  type = c("standard", "enhanced", "all"),
  id_col = "eid",
  dataset = NULL,
  include_testing = TRUE,
  eligibility = c("set_na", "require", "annotate"),
  standardize = FALSE,
  dry_run = FALSE,
  ...
)
```

Arguments

<code>data</code>	Optional participant data.frame. If NULL, fields are extracted from RAP.
<code>trait</code>	Trait names or abbreviations accepted by ukb_prs_catalog() .
<code>field_id</code>	Optional explicit published PRS field IDs.
<code>type</code>	"standard", "enhanced", or "all".
<code>id_col</code>	Participant ID column in data.
<code>dataset</code>	Optional RAP dataset passed to rap_extract_pheno() .
<code>include_testing</code>	Include field 26200 for Enhanced PRS requests.
<code>eligibility</code>	Enhanced PRS eligibility handling. "set_na" keeps all participants but clears Enhanced scores outside field 26200; "require" keeps testing-subgroup participants only; "annotate" only appends the indicator.
<code>standardize</code>	Add within-sample z-scores with suffix <code>_z</code> .
<code>dry_run</code>	Return a RAP extraction plan without running it.
<code>...</code>	Additional arguments passed to rap_extract_pheno() .

Value

A data.table of participant IDs and named PRS columns, or a `rap_extract_plan` in dry-run mode.

match_propensity	<i>Propensity Score Matching</i>
------------------	----------------------------------

Description

Perform propensity score matching using nearest neighbor or optimal matching.

Usage

```
match_propensity(
  data,
  ps_col = "ps",
  treatment,
  ratio = 1,
  caliper = 0.2,
  method = c("nearest", "optimal"),
  replace = FALSE,
  exact_match = NULL
)
```

Arguments

<code>data</code>	A <code>data.table</code> containing propensity scores.
<code>ps_col</code>	Character string specifying the propensity score column name. Default "ps".
<code>treatment</code>	Character string specifying the treatment variable name.
<code>ratio</code>	Numeric matching ratio (1:ratio). Default 1 for 1:1 matching.
<code>caliper</code>	Numeric caliper width in standard deviations of PS. Default 0.2.
<code>method</code>	Character string specifying matching method: "nearest" or "optimal".
<code>replace</code>	Logical; whether to match with replacement. Default FALSE.
<code>exact_match</code>	Character vector of variable names for exact matching. Default NULL.

Value

A `data.table` with matched data, including:

match_id Matched pair identifier

match_distance Distance between matched pairs

metabolite_to_metaboanalyst_name

Map metabolite names to MetaboAnalyst-compatible names

Description

Convert common UK Biobank Nightingale metabolite labels to names that are more likely to be recognized by MetaboAnalystR name cross-referencing. Users can provide a custom mapping table to override or extend the built-in map.

Usage

```
metabolite_to_metaboanalyst_name(
  metabolites,
  mapping_table = NULL,
  mapping_metabolite_col = "metabolite",
  mapping_name_col = "metaboanalyst_name",
  drop_unmapped = FALSE
)
```

Arguments

metabolites Character vector of metabolite names.

mapping_table Optional data.frame with metabolite-to-name mappings.

mapping_metabolite_col
 Column in mapping_table containing input metabolite labels. Default "metabolite".

mapping_name_col
 Column in mapping_table containing mapped names. Default "metaboanalyst_name".

drop_unmapped Logical. If TRUE, keep only mapped rows. Default FALSE.

Value

A data.frame with metabolite, metaboanalyst_name, and mapping_source.

Examples

```
metabolite_to_metaboanalyst_name(c("Acetate", "Alanine", "LDL Cholesterol"))
```

parse_cancer_registry *Parse Cancer Registry Records*

Description

Extracts cancer registry records from UK Biobank fields 40006, 40005, 40011, and 40012 into a standardized long-format table. Field 40006 stores cancer ICD-10 type, 40005 stores diagnosis date, 40011 stores tumour histology, and 40012 stores tumour behaviour.

Usage

```
parse_cancer_registry(dt)
```

Arguments

dt A data.table or data.frame containing UKB cancer registry columns.

Value

A data.table with columns: eid, cancer_icd10_code, diag_date, cancer_histology, cancer_behaviour, and source.

parse_death_records	<i>Parse Death Registry Records</i>
---------------------	-------------------------------------

Description

Extracts death registry data from UK Biobank linked mortality records. Parses both primary (p40001) and contributing (p40002) causes of death along with death dates (p40000). Caution: Death records only contain ICD-10 codes.

Usage

parse_death_records(dt)

Arguments

dt A data.table or data.frame containing UKB data with columns: eid, p40001_i*, p40002_i*_a*, and p40000_i*.

Details

Death causes serve as definitive diagnosis confirmation. If a participant died from a specific disease, the death date becomes the diagnosis date for that condition (if not previously diagnosed).

Value

A data.table with columns:

- eid** Participant identifier
- death_code** ICD-10 cause of death code
- death_date** Date of death
- source** Data source identifier ("Death")
- cause_type** "primary" or "secondary"

parse_gp_clinical	<i>Parse Filtered UKB GP Clinical Records</i>
-------------------	---

Description

Standardizes filtered gp_clinical records into one row per Read v2 / CTV3 code. This function does not read the full RAP table and does not call Spark. If a GP query is supplied, exact code matching adds the disease label.

Usage

```
parse_gp_clinical(
  data,
  query = NULL,
  special_date_action = c("flag", "exclude"),
  keep_empty_code = FALSE
)
```

Arguments

data	Filtered gp_clinical data.frame/data.table.
query	Optional ukb_gp_query. If omitted, the query attached by rap_run_gp_query() is used when available.
special_date_action	Keep and flag UKB placeholder dates ("flag") or remove those records ("exclude"). Missing dates remain because they can still represent coded events, but are never usable for timing.
keep_empty_code	Retain records where Read v2 / CTV3 code is empty.

Value

A standardized long-format data.table.

parse_gp_query	<i>Parse a GP Disease Query</i>
----------------	---------------------------------

Description

Validates a user-supplied Read v2 / CTV3 disease code table before a gp_clinical query is sent to UK Biobank RAP Spark SQL. Matching is exact: Read code punctuation and leading zeroes are preserved.

Usage

```
parse_gp_query(
  codes,
  date_from = NULL,
  date_to = NULL,
  providers = NULL,
  eids = NULL
)
```

Arguments

codes	A data.frame with columns disease, code_system, and code. code_system accepts common forms of Read v2 ("Read2", "read_2") and CTV3 ("CTV3", "read_3").
date_from	Optional earliest GP event date.
date_to	Optional latest GP event date.
providers	Optional integer provider codes (1 to 4).
eids	Optional participant identifiers. This is intended for small development cohorts; large cohorts should be represented by a Spark-side cohort join in a future extension.

Value

An object of class `ukb_gp_query`.

Examples

```
query <- parse_gp_query(data.frame(
  disease = c("Hypertension", "Type 2 diabetes"),
  code_system = c("Read2", "CTV3"),
  code = c("G20..", "C10E.")
))
```

parse_gp_registrations

Parse UKB GP Registration Records

Description

Standardizes the four official columns in the UK Biobank `gp_registrations` table and flags the pseudo dates defined by UKB Data-Coding 819. Registration records remain separate from clinical disease records.

Usage

```
parse_gp_registrations(data, special_date_action = c("flag", "exclude"))
```

Arguments

data A data.frame/data.table with eid, data_provider, reg_date, and deduct_date.
special_date_action Keep and flag UKB pseudo dates ("flag") or remove registration rows containing a pseudo date ("exclude").

Value

A long-format data.table with one row per GP registration record.

parse_icd10_diagnoses *Parse ICD-10 Hospital Diagnosis Records*

Description

Extracts ICD-10 diagnosis codes from UK Biobank hospital inpatient data. Converts the mixed-format storage (Python list string in p41270 + date array in p41280_a*) into a standardized long-format data.table.

Usage

```
parse_icd10_diagnoses(dt)
```

Arguments

dt A data.table or data.frame containing UKB data with columns: eid, p41270, and p41280_a* date columns.

Details

The function implements the Index-Match logic specified in the UKB data dictionary: the k-th element in p41270 corresponds to date column p41280_a(k-1) (0-indexed).

Processing pipeline:

1. Parse Python list string format in p41270
2. Melt p41280_a* date columns to long format
3. Join codes and dates by eid and positional index

Value

A data.table with columns:

eid Participant identifier
icd10_code ICD-10 diagnosis code
diag_date Date of diagnosis
source Data source identifier ("ICD10")

parse_icd9_diagnoses *Parse ICD-9 Hospital Diagnosis Records*

Description

Extracts ICD-9 diagnosis codes from UK Biobank hospital inpatient data. Converts the mixed-format storage (Python list string in p41271 + date array in p41281_a*) into a standardized long-format data.table.

Usage

```
parse_icd9_diagnoses(dt)
```

Arguments

dt A data.table or data.frame containing UKB data with columns: eid, p41271, and p41281_a* date columns.

Details

ICD-9 codes in UKB follow the format: 3-5 digits, optionally prefixed with V or E. The function handles logical NA columns that may occur when all values are missing.

Value

A data.table with columns:

eid Participant identifier

icd9_code ICD-9 diagnosis code

diag_date Date of diagnosis

source Data source identifier ("ICD9")

parse_opcs4_procedures

Parse OPCS4 Hospital Procedure Records

Description

Extracts OPCS4 operative procedure codes from UK Biobank hospital inpatient summary operations data. Supports the common export shape where p41272 stores a list-string of codes and p41282_a* stores the corresponding dates, while also tolerating expanded p41272_a* columns.

Usage

```
parse_opcs4_procedures(dt)
```

Arguments

dt A data.table or data.frame containing UKB data with columns: eid, and either p41272 or p41272_a*, plus p41282_a* date columns.

Details

The function implements the same index-matching logic used for UKB summary diagnosis fields: the k-th procedure code in p41272 corresponds to the date stored in p41282_a(k-1) (0-indexed).

Value

A data.table with columns:

eid Participant identifier

opcs4_code OPCS4 procedure code

diag_date Date of first recorded procedure for that code/index

source Data source identifier ("OPCS4")

parse_self_reported_illnesses

Parse Self-Reported Illness Records

Description

Extracts self-reported illness data from UK Biobank touchscreen questionnaire. Converts coded illness data (p20002_i*_a*) and interpolated year of diagnosis (p20008_i*_a*) into a standardized long-format data.table.

Usage

```
parse_self_reported_illnesses(dt, baseline_col = "p53_i0")
```

Arguments

dt A data.table or data.frame containing UKB data with columns: eid, p20002_i*_a*, and p20008_i*_a* columns.

baseline_col Column name for baseline date (default: "p53_i0").

Details

Year-to-date conversion logic:

- p20008 stores fractional years (e.g., 1983.5 = mid-1983)
- Fractional part * 12 = approximate month
- Special values (-1, -3) indicate "don't know" or "prefer not to answer"

Value

A data.table with columns:

eid Participant identifier

sr_code Self-report illness code

diag_date Approximate date of diagnosis

source Data source identifier ("Self-report")

instance Assessment instance (0, 1, 2, 3)

array_idx Array index within instance

baseline_report Whether the illness was reported at baseline

diagnosis_date_quality How the diagnosis date was obtained

plot.ukb_ml_flow	<i>Plot a UKB ML Flow Object</i>
------------------	----------------------------------

Description

Plot a UKB ML Flow Object

Usage

```
## S3 method for class 'ukb_ml_flow'
plot(x, type = c("roc", "shap_beeswarm"), ...)
```

Arguments

x	A ukb_ml_flow object.
type	Plot type: "roc" or "shap_beeswarm".
...	Additional arguments passed to the underlying plot function.

Value

A ggplot2 object.

plot.ukb_ml_flow_compare

Plot a UKB ML Flow Comparison Object

Description

Plot a UKB ML Flow Comparison Object

Usage

```
## S3 method for class 'ukb_ml_flow_compare'
plot(x, type = c("roc"), ...)
```

Arguments

x	A ukb_ml_flow_compare object.
type	Plot type. Currently "roc".
...	Additional arguments passed to plot_ml_roc_compare .

Value

A ggplot2 object.

plot_balance

Plot Covariate Balance (Love Plot)

Description

Create a Love plot comparing standardized mean differences before and after matching/weighting.

Usage

```
plot_balance(
  balance_before,
  balance_after,
  threshold = 0.1,
  title = "Covariate Balance",
  xlab = "Standardized Mean Difference"
)
```

Arguments

balance_before	A data.frame from <code>assess_balance()</code> for unmatched data.
balance_after	A data.frame from <code>assess_balance()</code> for matched/weighted data.
threshold	Numeric threshold for balance (vertical lines). Default 0.1.
title	Character string for plot title. Default "Covariate Balance".
xlab	Character string for x-axis label. Default "Standardized Mean Difference".

Value

A ggplot2 object.

plot_calibration	<i>Plot Calibration Curve</i>
------------------	-------------------------------

Description

Create a calibration plot comparing predicted probabilities to observed outcomes.

Usage

```
plot_calibration(  
  data,  
  predicted,  
  observed,  
  n_bins = 10,  
  smooth = TRUE,  
  conf_int = TRUE  
)
```

Arguments

data	A data.frame or data.table.
predicted	Character string specifying the column with predicted probabilities.
observed	Character string specifying the column with observed binary outcomes.
n_bins	Integer number of bins for calibration. Default 10.
smooth	Logical; whether to add a smooth calibration line. Default TRUE.
conf_int	Logical; whether to show confidence intervals. Default TRUE.

Value

A ggplot2 object.

plot_correlation	<i>Visualize correlation matrix as a heatmap</i>
------------------	--

Description

Create an annotated heatmap of a correlation matrix with customizable appearance. This helps identify patterns, multicollinearity, and variable relationships visually.

Usage

```
plot_correlation(  
  corr_matrix,  
  title = "Correlation Matrix",  
  show_values = TRUE,  
  digits = 2,  
  text_size = 3,  
  color_low = "#3B4CC0",  
  color_mid = "white",  
  color_high = "#B40426",  
  upper_triangle = FALSE  
)
```

Arguments

corr_matrix	A numeric correlation matrix (from <code>run_correlation()</code> or <code>cor()</code>).
title	Character string. Plot title. Default: "Correlation Matrix".
show_values	Logical. If TRUE, display correlation values on tiles. Default: TRUE.
digits	Integer. Number of decimal places for correlation values. Default: 2.
text_size	Numeric. Size of text labels on tiles. Default: 3.
color_low	Character. Color for negative correlations. Default: "#3B4CC0" (blue).
color_mid	Character. Color for zero correlation. Default: "white".
color_high	Character. Color for positive correlations. Default: "#B40426" (red).
upper_triangle	Logical. If TRUE, show only upper triangle. Default: FALSE.

Value

A ggplot2 object. Can be further customized with ggplot2 functions.

plot_cox_loghr_correlation
<i>Plot training-validation Cox log(HR) concordance</i>

Description

Plot training-validation Cox log(HR) concordance

Usage

```
plot_cox_loghr_correlation(  
  comparison,  
  train_loghr_col = "train_logHR",  
  validation_loghr_col = "validation_logHR",  
  highlight_col = "train_significant_bonferroni",  
  highlight_label = "Train Bonferroni significant"  
)
```

Arguments

- comparison Comparison table from [ukb_compare_cox_results\(\)](#).
- train_loghr_col Training log(HR) column.
- validation_loghr_col Validation log(HR) column.
- highlight_col Optional logical column used to highlight proteins.
- highlight_label Highlight legend label.

Value

A ggplot object.

plot_cox_sensitivity_correlation
<i>Plot sensitivity-analysis Cox log(HR) concordance</i>

Description

Plot sensitivity-analysis Cox log(HR) concordance

Usage

```
plot_cox_sensitivity_correlation(
  comparison,
  sensitivity_col = "sensitivity",
  main_loghr_col = "main_logHR",
  sensitivity_loghr_col = "sensitivity_logHR",
  highlight_col = "main_significant_bonferroni",
  highlight_label = "Main Bonferroni significant",
  nrow = NULL,
  ncol = NULL
)
```

Arguments

comparison	Comparison table from <code>ukb_compare_sensitivity_cox()</code> .
sensitivity_col	Sensitivity-analysis label column.
main_loghr_col	Main-analysis log(HR) column.
sensitivity_loghr_col	Sensitivity-analysis log(HR) column.
highlight_col	Optional logical column used to highlight variables.
highlight_label	Highlight legend label.
nrow, ncol	Facet layout.

Value

A ggplot object.

plot_enrichment_lollipop

Plot enrichment results as a lollipop chart via TCMDATA

Description

A thin wrapper around `TCMDATA::ggllollipop()` for enrichment results. This function accepts either a raw `enrichResult` object or a list returned by one of the proteomics ORA helpers in this package.

Usage

```
plot_enrichment_lollipop(x, ...)
```

Arguments

- x An enrichResult object, or a list containing ora_result.
- ... Additional arguments passed to TCMDATA::gglolipop().

Value

A ggplot2 object.

plot_forest	<i>Plot Forest Plot for Subgroup Analysis</i>
-------------	---

Description

Create a forest plot to visualize subgroup analysis results with effect estimates and confidence intervals.

Usage

```
plot_forest(  
  results,  
  estimate_col = "estimate",  
  lower_col = "lower95",  
  upper_col = "upper95",  
  label_col = "subgroup",  
  pvalue_col = "pvalue",  
  p_interaction_col = "p_interaction",  
  null_value = 1,  
  log_scale = TRUE,  
  colors = NULL,  
  title = "Subgroup Analysis",  
  xlab = "Hazard Ratio (95% CI)",  
  show_n = TRUE,  
  show_events = TRUE  
)
```

Arguments

- results A data.frame from run_subgroup_analysis() or run_multi_subgroup().
- estimate_col Character string specifying the column name for effect estimates. Default "estimate".
- lower_col Character string specifying the column for lower CI. Default "lower95".
- upper_col Character string specifying the column for upper CI. Default "upper95".
- label_col Character string specifying the column for subgroup labels. Default "subgroup".
- pvalue_col Character string specifying the column for p-values. Default "pvalue".

p_interaction_col	Character string for interaction p-value column. Default "p_interaction".
null_value	Numeric value for the null effect line. Default 1 (for HR/OR).
log_scale	Logical; whether to use log scale for x-axis. Default TRUE.
colors	Character vector of colors. Default NULL uses ggplot2 defaults.
title	Character string for plot title. Default "Subgroup Analysis".
xlab	Character string for x-axis label. Default "Hazard Ratio (95% CI)".
show_n	Logical; whether to show sample size. Default TRUE.
show_events	Logical; whether to show event count. Default TRUE.

Value

A ggplot2 object.

plot_go_ora_bar	<i>Plot GO ORA results as a bar chart via TCMDATA</i>
-----------------	---

Description

A thin wrapper around `TCMDATA::go_barplot()` for GO enrichment results. This function accepts either a raw `enrichResult` object or a list returned by `run_protein_ora()`.

Usage

```
plot_go_ora_bar(x, ...)
```

Arguments

x	An <code>enrichResult</code> object, or a list containing <code>ora_result</code> .
...	Additional arguments passed to <code>TCMDATA::go_barplot()</code> .

Value

A ggplot2 object.

plot_heatmap	<i>Plot a publication-style heatmap</i>
--------------	---

Description

Draw a compact heatmap from long-format data. The function uses string column names and `.data` pronouns internally, which makes it suitable for scripted package workflows and CRAN checks.

Usage

```
plot_heatmap(
  data,
  x,
  y,
  fill,
  label = NULL,
  show_values = FALSE,
  low = "#2F6FA3",
  mid = "#F7F7F7",
  high = "#C74732",
  midpoint = 0,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  fill_lab = NULL,
  base_size = 7
)
```

Arguments

<code>data</code>	A data.frame.
<code>x</code>	Character column name for the x axis.
<code>y</code>	Character column name for the y axis.
<code>fill</code>	Character column name for the heatmap value.
<code>label</code>	Optional character column name for tile labels.
<code>show_values</code>	Logical. If TRUE, show values or label on tiles.
<code>low</code>	Low-end color for the diverging scale.
<code>mid</code>	Midpoint color for the diverging scale.
<code>high</code>	High-end color for the diverging scale.
<code>midpoint</code>	Midpoint for the diverging scale.
<code>title</code>	Optional title. If NULL, no title is shown.
<code>xlab</code>	Optional x-axis label.
<code>ylab</code>	Optional y-axis label.
<code>fill_lab</code>	Optional fill legend label.
<code>base_size</code>	Base font size.

Value

A ggplot object.

plot_km_curve

Plot Kaplan-Meier Survival Curve

Description

Create a Kaplan-Meier survival curve with optional risk table and log-rank p-value.

Usage

```
plot_km_curve(
  data,
  time_col,
  status_col,
  group_col = NULL,
  conf_int = TRUE,
  risk_table = TRUE,
  censor_marks = TRUE,
  palette = "jco",
  title = NULL,
  xlab = "Time (years)",
  ylab = "Survival Probability",
  legend_title = "Group",
  median_line = TRUE,
  pvalue = TRUE,
  xlim = NULL,
  break_time = NULL
)
```

Arguments

data	A data.frame or data.table containing survival data.
time_col	Character string specifying the time column name.
status_col	Character string specifying the event status column name.
group_col	Character string specifying the grouping variable. Default NULL for overall curve.
conf_int	Logical; whether to show confidence intervals. Default TRUE.
risk_table	Logical; whether to show number at risk table. Default TRUE.
censor_marks	Logical; whether to show censoring marks. Default TRUE.
palette	Character string specifying color palette. Default "jco". Options: "jco", "nejm", "lancet", "npg", or custom color vector.
title	Character string for plot title. Default NULL.

xlab	Character string for x-axis label. Default "Time (years)".
ylab	Character string for y-axis label. Default "Survival Probability".
legend_title	Character string for legend title. Default "Group".
median_line	Logical; whether to show median survival line. Default TRUE.
pvalue	Logical; whether to show log-rank p-value. Default TRUE.
xlim	Numeric vector of length 2 for x-axis limits. Default NULL.
break_time	Numeric value for x-axis tick interval. Default NULL.

Value

A ggplot2 object (or a list with plot and risk table if risk_table = TRUE).

plot_mediation	<i>Plot Mediation Analysis Results</i>
----------------	--

Description

Create visualizations for mediation analysis results, including path diagrams, effect bar charts, and decomposition plots.

Usage

```
plot_mediation(
  mediation_result,
  type = c("effects", "path", "decomposition"),
  show_ci = TRUE,
  show_pvalue = TRUE,
  exponentiate = FALSE,
  title = NULL,
  colors = NULL
)
```

Arguments

mediation_result	An object of class "mediation_result" from run_mediation().
type	Character string specifying plot type: "path" Path diagram showing exposure -> mediator -> outcome "effects" Bar chart of effect estimates with confidence intervals "decomposition" Pie/bar chart showing effect decomposition (NDE vs NIE)
show_ci	Logical; whether to show confidence intervals. Default TRUE.
show_pvalue	Logical; whether to show p-values. Default TRUE.
exponentiate	Logical; whether to exponentiate estimates (for HR/OR). Default FALSE.
title	Character string for plot title. Default NULL (auto-generated).
colors	Character vector of colors. Default NULL uses package defaults.

Value

A ggplot2 object.

plot_mediation_forest *Plot Forest Plot for Multiple Mediator Analysis*

Description

Create a forest plot to visualize results from multiple mediator analysis.

Usage

```
plot_mediation_forest(  
  multi_mediation_result,  
  effect_type = c("tnie", "pnde", "te", "pm"),  
  exponentiate = FALSE,  
  null_value = 0,  
  title = "Mediation Analysis: Multiple Mediators"  
)
```

Arguments

multi_mediation_result	A data.frame from run_multi_mediator().
effect_type	Character string specifying which effect to display: "tnie" (indirect effect), "pnde" (direct effect), "te" (total effect), or "pm" (proportion mediated). Default "tnie".
exponentiate	Logical; whether to exponentiate estimates. Default FALSE.
null_value	Numeric; null effect value for reference line. Default 0.
title	Character string for plot title.

Value

A ggplot2 object.

`plot_metabolite_ora_barplot`*Plot metabolite ORA results as a bar plot*

Description

Plot metabolite ORA results as a bar plot

Usage

```
plot_metabolite_ora_barplot(  
  x,  
  top_n = 15,  
  p_col = "pvalue",  
  pathway_col = "pathway",  
  fill_color = "#2F6FA3"  
)
```

Arguments

<code>x</code>	A data.frame returned by <code>run_metabolite_ora()</code> \$ora_result or a <code>ukb_metabolite_ora</code> object.
<code>top_n</code>	Number of pathways to show. Default 15.
<code>p_col</code>	P-value column used for ordering and color. Default "pvalue".
<code>pathway_col</code>	Column containing pathway names. Default "pathway".
<code>fill_color</code>	Bar color.

Value

A ggplot object.

`plot_metabolite_ora_dotplot`*Plot metabolite ORA results as a dot plot*

Description

Plot metabolite ORA results as a dot plot

Usage

```
plot_metabolite_ora_dotplot(
  x,
  top_n = 15,
  p_col = "pvalue",
  size_col = "hits",
  pathway_col = "pathway",
  color_low = "#2F6FA3",
  color_high = "#C74732"
)
```

Arguments

x	A data.frame returned by run_metabolite_ora()\$ora_result or a ukb_metabolite_ora object.
top_n	Number of pathways to show. Default 15.
p_col	P-value column used for ordering and color. Default "pvalue".
size_col	Column used for point size. Default "hits".
pathway_col	Column containing pathway names. Default "pathway".
color_low, color_high	Colors for the sequential p-value gradient.

Value

A ggplot object.

plot_mi_diagnostics *Plot Multiple Imputation Diagnostics*

Description

Creates diagnostic plots for multiple imputation results, including fraction of missing information (FMI), variance ratios, and degrees of freedom.

Usage

```
plot_mi_diagnostics(
  mi_result,
  type = c("fmi", "variance_ratio", "df"),
  title = NULL
)
```

Arguments

mi_result	An object of class <code>mi_pooled_result</code> from <code>pool_mi_models()</code> .
type	Character string specifying the diagnostic plot type: "fmi" Bar plot of FMI for each coefficient "variance_ratio" Ratio of between- to within-imputation variance "df" Degrees of freedom for each coefficient
title	Character string for plot title. If NULL, auto-generated.

Value

A ggplot2 object.

plot_mi_pooled	<i>Plot Multiple Imputation Pooled Results</i>
----------------	--

Description

Creates a forest plot for pooled estimates from multiple imputation analysis.

Usage

```
plot_mi_pooled(
  mi_result,
  terms = NULL,
  exponentiate = NULL,
  null_value = NULL,
  title = "Pooled Estimates (Multiple Imputation)",
  colors = NULL,
  show_fmi = TRUE
)
```

Arguments

mi_result	An object of class <code>mi_pooled_result</code> from <code>pool_mi_models()</code> .
terms	Character vector of terms to include. If NULL, all terms except intercept are shown.
exponentiate	Logical; whether to exponentiate estimates. If NULL, uses the setting from the <code>mi_result</code> object.
null_value	Numeric; reference line value. If NULL, automatically set based on exponentiation (0 for linear scale, 1 for exp scale).
title	Character string for plot title.
colors	Named character vector for colors. Default uses package palette.
show_fmi	Logical; whether to display FMI (Fraction of Missing Information) as point size or annotation. Default TRUE.

Value

A ggplot2 object.

plot_ml_calibration	<i>Plot Calibration Curve</i>
---------------------	-------------------------------

Description

Create calibration curve plot showing predicted vs observed probabilities.

Usage

```
plot_ml_calibration(object, title = "Calibration Curve", ...)
```

Arguments

object	A ukb_ml_calibration object from ukb_ml_calibration()
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_compare	<i>Plot Model Comparison</i>
-----------------	------------------------------

Description

Create comparison plot for multiple ML models.

Usage

```
plot_ml_compare(  
  object,  
  metric = NULL,  
  type = c("bar", "dot"),  
  title = "Model Comparison",  
  ...  
)
```

Arguments

object	A ukb_ml_compare object from ukb_ml_compare()
metric	Metric to highlight (default first available)
type	Plot type: "bar", "dot", or "radar"
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_confusion	<i>Plot Confusion Matrix</i>
-------------------	------------------------------

Description

Create heatmap visualization of confusion matrix.

Usage

```
plot_ml_confusion(  
  object,  
  normalize = TRUE,  
  colors = c("white", "#E34A33"),  
  title = "Confusion Matrix",  
  ...  
)
```

Arguments

object	A ukb_ml_confusion object from ukb_ml_confusion()
normalize	Whether to show percentages (default TRUE)
colors	Color gradient (default c("white", "#E34A33"))
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_dca	<i>Plot Decision Curve Analysis</i>
-------------	-------------------------------------

Description

Create a Decision Curve Analysis plot showing net benefit of the model compared to treat-all and treat-none strategies.

Usage

```
plot_ml_dca(object, title = "Decision Curve Analysis", ...)
```

Arguments

object	A <code>ukb_ml_dca</code> object from <code>ukb_ml_dca()</code>
title	Plot title
...	Additional arguments

Value

A `ggplot2` object

plot_ml_gain	<i>Plot Gain Curve</i>
--------------	------------------------

Description

Create a Gain curve plot comparing model targeting against random selection.

Usage

```
plot_ml_gain(object, title = "Gain Curve", ...)
```

Arguments

object	A <code>ukb_ml_gain_lift</code> object from <code>ukb_ml_gain_lift()</code>
title	Plot title
...	Additional arguments

Value

A `ggplot2` object

plot_ml_importance	<i>Plot Variable Importance</i>
--------------------	---------------------------------

Description

Create a bar plot of variable importance from a trained ML model.

Usage

```
plot_ml_importance(
  object,
  n_features = 20,
  type = c("bar", "dot"),
  color = "#3182BD",
  title = "Variable Importance",
  ...
)
```

Arguments

object	A <code>ukb_ml</code> object from <code>ukb_ml_model()</code>
n_features	Number of top features to display (default 20)
type	Plot type: "bar" or "dot"
color	Bar color (default "#3182BD")
title	Plot title
...	Additional arguments

Value

A `ggplot2` object

plot_ml_ks	<i>Plot KS Curve</i>
------------	----------------------

Description

Create a KS (Kolmogorov-Smirnov) curve plot showing TPR, FPR, and their difference (KS statistic) across thresholds.

Usage

```
plot_ml_ks(object, title = "KS Curve", ...)
```

Arguments

object	A ukb_ml_ks object from ukb_ml_ks()
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_lift	<i>Plot Lift Curve</i>
--------------	------------------------

Description

Create a Lift curve plot showing the ratio of model vs random targeting.

Usage

```
plot_ml_lift(object, title = "Lift Curve", ...)
```

Arguments

object	A ukb_ml_gain_lift object from ukb_ml_gain_lift()
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_pr	<i>Plot PR Curve</i>
------------	----------------------

Description

Create a Precision-Recall curve plot with AUPRC annotation.

Usage

```
plot_ml_pr(object, title = "PR Curve", ...)
```

Arguments

object	A ukb_ml_pr object from ukb_ml_pr()
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_roc	<i>Plot ROC Curves</i>
-------------	------------------------

Description

Create ROC curve plot for one or more ML models.

Usage

```
plot_ml_roc(object, ci_alpha = 0.2, title = "ROC Curve", ...)
```

Arguments

object	A ukb_ml_roc object from ukb_ml_roc()
ci_alpha	Alpha for confidence interval ribbon (default 0.2)
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_roc_compare	<i>Plot One or More ROC Curves from Tidy ROC Data</i>
---------------------	---

Description

Creates a publication-ready ROC curve plot from one or more data frames returned by [ukb_ml_roc_data](#). AUC and 95% confidence interval values are included in the legend when available.

Usage

```
plot_ml_roc_compare(
  roc_data,
  colors = NULL,
  show_auc = TRUE,
  title = NULL,
  xlab = "1 - Specificity",
  ylab = "Sensitivity",
  legend_position = "bottom",
  base_size = 7,
  ...
)
```

Arguments

roc_data	A data.frame returned by ukb_ml_roc_data , a row-bound data.frame of multiple ROC tables, or a list of such data.frames.
colors	Optional named or unnamed vector of line colors.
show_auc	Logical. Include AUC and 95% CI in the legend labels.
title	Optional plot title. If NULL, no title is shown.
xlab	X-axis label.
ylab	Y-axis label.
legend_position	Legend position passed to <code>theme()</code> .
base_size	Base font size.
...	Additional arguments reserved for future use.

Value

A ggplot2 object.

plot_participant_flow *Plot a participant flow table*

Description

Plot a participant flow table

Usage

```
plot_participant_flow(
  flow,
  show_removed = TRUE,
  show_events = TRUE,
  fill = "#2C7FB8"
)
```

Arguments

flow	A <code>ukb_participant_flow</code> object.
show_removed	Logical. If TRUE, annotate removals at each step.
show_events	Logical. If TRUE, include event counts in labels when <code>outcome_col</code> was supplied to <code>ukb_participant_flow()</code> .
fill	Fill color for the retained-participant bars.

Value

A `ggplot` object.

Examples

```
dat <- data.frame(eid = 1:5, age = c(50, 60, NA, 55, 70), status = c(0, 1, 0, 1, 0))
flow <- ukb_participant_flow(dat, list("Complete age" = "age"), outcome_col = "status")
plot_participant_flow(flow)
```

`plot_ps_distribution` *Plot Propensity Score Distribution*

Description

Visualize the distribution of propensity scores by treatment group.

Usage

```
plot_ps_distribution(
  data,
  ps_col = "ps",
  treatment,
  type = c("histogram", "density", "mirror"),
  matched = FALSE,
  match_col = NULL
)
```

Arguments

data	A <code>data.frame</code> or <code>data.table</code> containing propensity scores.
ps_col	Character string specifying the PS column name. Default "ps".
treatment	Character string specifying the treatment variable name.
type	Character string specifying plot type: "histogram", "density", or "mirror".
matched	Logical; whether to show matched vs unmatched. Default FALSE.
match_col	Character string for the matching indicator column. Default NULL.

Value

A `ggplot2` object.

plot_rcs	<i>Plot a restricted cubic spline exposure-response curve</i>
----------	---

Description

Produces a publication-ready ggplot2 figure from a `ukb_rcs` object returned by `run_rcs`. The main panel shows the estimated effect curve with a 95% (histogram, density, or rug) is drawn behind the curve to show exposure density. P values and the knot count are annotated by default.

Usage

```
plot_rcs(x, ...)

## S3 method for class 'ukb_rcs'
plot(x, ...)

## S3 method for class 'ukb_rcs'
plot_rcs(
  x,
  show_distribution = TRUE,
  distribution = c("histogram", "density", "rug"),
  show_ref = TRUE,
  show_p = TRUE,
  show_knots = TRUE,
  curve_color = "#2166AC",
  dist_color = "#AECDE8",
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

<code>x</code>	A <code>ukb_rcs</code> object from <code>run_rcs</code> .
<code>...</code>	Additional arguments (currently unused).
<code>show_distribution</code>	Logical. Whether to overlay an exposure distribution layer. Default TRUE.
<code>distribution</code>	One of "histogram" (default), "density", or "rug".
<code>show_ref</code>	Logical. Whether to mark the reference value with a point. Default TRUE.
<code>show_p</code>	Logical. Whether to annotate P-overall and P-nonlinear. Default TRUE.
<code>show_knots</code>	Logical. Whether to annotate the knot count. Default TRUE.
<code>curve_color</code>	Character. Hex color for the main curve and ribbon. Default "#2166AC" (deep blue).
<code>dist_color</code>	Character. Fill color for the distribution layer. Default "#AECDE8".

title	Character. Plot title. Default NULL (no title).
xlab	Character. x-axis label. Default: the exposure variable name.
ylab	Character. y-axis label. Default is chosen from model type.

Value

A ggplot2 object.

plot_regression_volcano

Plot a volcano-style regression summary

Description

Create a volcano-style plot from regression summary results such as `runmulti_cox()` or `runmulti_logit()`. The x-axis shows the supplied effect estimate column (for example HR or OR), and the y-axis shows $-\log_{10}(P)$. Points can be highlighted by an adjusted p-value column, while labels are selected from the largest and smallest highlighted effects.

Usage

```
plot_regression_volcano(
  data,
  effect_col = NULL,
  p_col = "pvalue",
  adjusted_p_col = NULL,
  label_col = NULL,
  significance_cutoff = 0.05,
  top_n_label_each = 5,
  null_effect = 1,
  x_lab = NULL,
  y_lab = NULL,
  x_limits = NULL,
  y_limits = NULL,
  point_size = 1.05,
  label_size = 2,
  colors = c(neutral = "#D8D8D8", lower = "#2F6FA3", higher = "#C74732"),
  show_cutoff = TRUE
)
```

Arguments

data	A data.frame containing regression results.
effect_col	Character. Column containing the effect estimate to plot on the x-axis. If NULL, the function uses HR, then OR, then estimate when available.
p_col	Character. Column containing raw p-values. Default "pvalue".

adjusted_p_col	Optional character. Column used for highlighting significant points, such as "p_bonferroni" or "p_bh". If NULL, p_col is used.
label_col	Optional character. Column used for point labels. If NULL, the function uses gene_symbol, then protein_clean, then variable when available.
significance_cutoff	Numeric cutoff applied to adjusted_p_col. Default 0.05.
top_n_label_each	Integer. Number of highlighted proteins to label from each direction. Direction is defined relative to null_effect.
null_effect	Numeric null effect. Use 1 for ratio estimates such as HR/OR and 0 for beta estimates. Default 1.
x_lab, y_lab	Axis labels. If NULL, defaults are generated.
x_limits, y_limits	Optional numeric vectors of length 2 for axis limits.
point_size	Numeric point size. Default 1.05.
label_size	Numeric label size. Default 2.
colors	Named character vector for groups neutral, lower, and higher.
show_cutoff	Logical. Whether to draw a horizontal significance cutoff line. Default TRUE.

Value

A ggplot2 object with attributes plot_data and label_data.

plot_scatter	<i>Plot a publication-style scatter plot</i>
--------------	--

Description

Draw a scatter plot with optional color grouping, linear smooth, and reference line. This is intended for compact association or validation panels.

Usage

```
plot_scatter(
  data,
  x,
  y,
  color = NULL,
  palette = NULL,
  add_smooth = TRUE,
  add_identity = FALSE,
  alpha = 0.72,
  point_size = 1.2,
  title = NULL,
```

```

    xlab = NULL,
    ylab = NULL,
    base_size = 7
  )

```

Arguments

data	A data.frame.
x	Character numeric column name for the x axis.
y	Character numeric column name for the y axis.
color	Optional grouping column for point colors.
palette	Optional vector of colors.
add_smooth	Logical. Add a linear smooth line.
add_identity	Logical. Add a dashed $y = x$ reference line.
alpha	Point alpha.
point_size	Point size.
title	Optional title. If NULL, no title is shown.
xlab	Optional x-axis label.
ylab	Optional y-axis label.
base_size	Base font size.

Value

A ggplot object.

plot_shap_beeswarm	<i>Plot SHAP Beeswarm Summary</i>
--------------------	-----------------------------------

Description

Creates a SHAP beeswarm plot directly from a `ukb_shap` object. The plot displays the top features ranked by mean absolute SHAP value, with point color representing the normalized feature value.

Usage

```

plot_shap_beeswarm(
  object,
  max_features = 20,
  label_map = NULL,
  feature_col = "feature",
  label_col = "label",
  colors = c("#1E88E5", "#7B3294", "#FF0051"),
  point_size = 0.58,
  alpha = 0.62,

```

```

    jitter_height = 0.18,
    seed = 20260509,
    title = NULL,
    xlab = "SHAP value",
    legend_title = "Feature value",
    base_size = 7,
    return_data = FALSE,
    ...
  )

```

Arguments

object	A <code>ukb_shap</code> object from ukb_shap .
max_features	Maximum number of features to display.
label_map	Optional named vector or <code>data.frame</code> mapping feature names to display labels. For a <code>data.frame</code> , columns are controlled by <code>feature_col</code> and <code>label_col</code> .
feature_col	Feature column in <code>label_map</code> when it is a <code>data.frame</code> .
label_col	Label column in <code>label_map</code> when it is a <code>data.frame</code> .
colors	Three or more colors for the low-to-high feature value scale.
point_size	Point size.
alpha	Point transparency.
jitter_height	Vertical jitter height.
seed	Optional seed for reproducible jitter.
title	Optional plot title. If <code>NULL</code> , no title is shown.
xlab	X-axis label.
legend_title	Legend title.
base_size	Base font size.
return_data	Logical. If <code>TRUE</code> , returns a list with plot and data; otherwise returns only the <code>ggplot</code> object.
...	Additional arguments reserved for future use.

Value

A `ggplot2` object, or a list with plot data when `return_data = TRUE`.

plot_shap_dependence	<i>Plot SHAP Dependence</i>
----------------------	-----------------------------

Description

Create SHAP dependence plot showing the relationship between a feature’s value and its SHAP value.

Usage

```
plot_shap_dependence(  
  object,  
  feature,  
  color_feature = NULL,  
  alpha = 0.5,  
  smooth = TRUE,  
  title = NULL,  
  ...  
)
```

Arguments

object	A ukb_shap object
feature	Feature name to analyze
color_feature	Optional feature for coloring points (interaction)
alpha	Point transparency (default 0.5)
smooth	Add smooth line (default TRUE)
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_shap_force	<i>Plot SHAP Force (Waterfall)</i>
-----------------	------------------------------------

Description

Create a waterfall plot showing feature contributions for a single prediction.

Usage

```
plot_shap_force(object, row_id = 1, max_features = 10, title = NULL, ...)
```

Arguments

object	A ukb_shap object
row_id	Row index to explain (default 1)
max_features	Maximum features to show (default 10)
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_shap_summary	<i>Plot SHAP Summary</i>
-------------------	--------------------------

Description

Create SHAP summary plot (beeswarm or bar) for feature importance.

Usage

```
plot_shap_summary(  
  object,  
  max_features = 20,  
  type = c("beeswarm", "bar"),  
  color_palette = "viridis",  
  title = "SHAP Summary",  
  ...  
)
```

Arguments

object	A ukb_shap object from ukb_shap()
max_features	Maximum features to display (default 20)
type	Plot type: "beeswarm" or "bar"
color_palette	Color palette for beeswarm (default "viridis")
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_stacked_bar	<i>Plot a publication-style stacked bar chart</i>
------------------	---

Description

Summarize observations by `x` and `fill`, then draw either proportional or count-based stacked bars.

Usage

```
plot_stacked_bar(  
  data,  
  x,  
  fill,  
  weight = NULL,  
  position = c("fill", "stack"),  
  palette = NULL,  
  title = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  legend_title = NULL,  
  base_size = 7  
)
```

Arguments

<code>data</code>	A data.frame.
<code>x</code>	Character column name for bar groups.
<code>fill</code>	Character column name for stack groups.
<code>weight</code>	Optional numeric column name for weighted summaries.
<code>position</code>	Either "fill" for proportions or "stack" for counts.
<code>palette</code>	Optional vector of fill colors.
<code>title</code>	Optional title. If NULL, no title is shown.
<code>xlab</code>	Optional x-axis label.
<code>ylab</code>	Optional y-axis label.
<code>legend_title</code>	Optional legend title.
<code>base_size</code>	Base font size.

Value

A ggplot object.

plot_top_hrBars	<i>Plot top positive and inverse Cox associations</i>
-----------------	---

Description

Plot top positive and inverse Cox associations

Usage

```
plot_top_hrBars(  
  top_results,  
  facet_col = "dataset",  
  hr_col = "HR",  
  lower_col = "lower95",  
  upper_col = "upper95",  
  label_col = "label"  
)
```

Arguments

top_results	A data.frame from ukb_top_hr_results() or equivalent.
facet_col	Optional column used for faceting, commonly "dataset".
hr_col	HR column.
lower_col	Lower confidence-limit column.
upper_col	Upper confidence-limit column.
label_col	Label column.

Value

A ggplot object.

plot_violin	<i>Plot a publication-style violin plot</i>
-------------	---

Description

Draw grouped distributions using violin layers with optional boxplot overlay.

Usage

```
plot_violin(
  data,
  x,
  y,
  fill = NULL,
  palette = NULL,
  add_boxplot = TRUE,
  add_points = FALSE,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  base_size = 7
)
```

Arguments

<code>data</code>	A data.frame.
<code>x</code>	Character column name for groups.
<code>y</code>	Character numeric column name.
<code>fill</code>	Optional fill grouping column. Defaults to <code>x</code> .
<code>palette</code>	Optional vector of fill colors.
<code>add_boxplot</code>	Logical. Overlay a narrow boxplot.
<code>add_points</code>	Logical. Overlay jittered observations.
<code>title</code>	Optional title. If <code>NULL</code> , no title is shown.
<code>xlab</code>	Optional x-axis label.
<code>ylab</code>	Optional y-axis label.
<code>base_size</code>	Base font size.

Value

A ggplot object.

`pool_custom_estimates` *Pool Custom Estimates from Multiple Imputations*

Description

Combines custom parameter estimates (not limited to regression coefficients) from multiply imputed datasets using Rubin's Rules.

Usage

```
pool_custom_estimates(
  estimates,
  variances,
  df.complete = Inf,
  conf.level = 0.95,
  labels = NULL
)
```

Arguments

<code>estimates</code>	A list of numeric vectors containing point estimates from each imputed dataset. All vectors must have the same length.
<code>variances</code>	A list of variance-covariance matrices (or single variances as 1x1 matrices) corresponding to the estimates.
<code>df.complete</code>	Complete-data degrees of freedom. Default Inf.
<code>conf.level</code>	Confidence level for intervals. Default 0.95.
<code>labels</code>	Character vector of labels for the estimates. If NULL, names are taken from the first estimate vector or generated as "est1", "est2", etc.

Value

An object of class `mi_pooled_result`.

<code>pool_mi_models</code>	<i>Pool Results from Multiple Imputation Models</i>
-----------------------------	---

Description

Combines results of regression analyses performed on multiply imputed datasets using Rubin's Rules via the `mitools` package.

Usage

```
pool_mi_models(
  models = NULL,
  datasets = NULL,
  formula = NULL,
  model_type = c("lm", "logistic", "poisson", "cox", "negbin"),
  family = NULL,
  df.complete = Inf,
  conf.level = 0.95,
  exponentiate = NULL
)
```

Arguments

<code>models</code>	A list of fitted model objects (one per imputed dataset). If NULL, models will be fitted using <code>datasets</code> and <code>formula</code> .
<code>datasets</code>	A list of data.frames or an <code>imputationList</code> object. Required if <code>models</code> is NULL.
<code>formula</code>	A formula specifying the model. Required if <code>datasets</code> is provided.
<code>model_type</code>	Character string specifying the model type: "lm" Linear regression "logistic" Logistic regression (GLM with binomial family) "poisson" Poisson regression "cox" Cox proportional hazards model "negbin" Negative binomial regression
<code>family</code>	A family object for GLM. If NULL, inferred from <code>model_type</code> .
<code>df.complete</code>	Complete-data degrees of freedom for small-sample correction. Default is Inf (large sample approximation).
<code>conf.level</code>	Confidence level for intervals. Default 0.95.
<code>exponentiate</code>	Logical; whether to exponentiate coefficients (for OR/HR/RR). If NULL, automatically determined based on model type.

Value

An object of class `mi_pooled_result` containing:

pooled Data frame with pooled estimates, standard errors, CIs, p-values, and FMI

mi_result The raw `MIresult` object from `mitools`

n_imputations Number of imputed datasets

model_type The model type used

formula The model formula

exponentiated Whether estimates are exponentiated

call The function call

<code>prepare_prs_weights</code>	<i>Prepare a PLINK2 PRS Weight File</i>
----------------------------------	---

Description

Prepare a PLINK2 PRS Weight File

Usage

```
prepare_prs_weights(
  weights,
  variant_col = "ID",
  effect_allele_col = "A1",
  weight_col = "WEIGHT",
  other_allele_col = NULL,
  chr_col = NULL,
  pos_col = NULL,
  effect_allele_freq_col = NULL,
  weight_type = c("beta", "log_or", "or", "direct"),
  genome_build = NULL,
  output = NULL,
  overwrite = FALSE
)
```

Arguments

<code>weights</code>	A data.frame or a delimited weight-file path.
<code>variant_col</code> , <code>effect_allele_col</code> , <code>weight_col</code>	Column names or positions.
<code>other_allele_col</code> , <code>chr_col</code> , <code>pos_col</code> , <code>effect_allele_freq_col</code>	Optional columns used by harmonize_prs_weights() .
<code>weight_type</code>	Input scale: "beta", "log_or", and "direct" are used unchanged; "or" is transformed with <code>log()</code> .
<code>genome_build</code>	Optional "GRCh37" or "GRCh38" metadata.
<code>output</code>	Optional normalized tab-delimited output path. An explicit output is required before the object can be passed to ukb_plan_prs() .
<code>overwrite</code>	Overwrite an existing output.

Value

A normalized data.frame of class `ukb_prs_weights`, with columns ID, A1, and WEIGHT.

preprocess_baseline	<i>Preprocess UKB baseline variables</i>
---------------------	--

Description

A unified function to preprocess UKB baseline characteristics with automatic field mapping and standardized transformations.

Usage

```
preprocess_baseline(
  df,
  variables,
  custom_mapping = NULL,
  missing_action = c("keep", "drop"),
  invalid_codes = c(-1, -3)
)
```

Arguments

<code>df</code>	A data.table or data.frame containing UKB data from rap platform export.
<code>variables</code>	Character vector of variable names to process. Use <code>get_variable_info()</code> to see available variables.
<code>custom_mapping</code>	Optional named list for user-defined variable mappings. Each element should have: <code>ukb_col</code> (required), <code>description</code> (optional). Example: <code>list(my_var = list(ukb_col = "p12345_i0", description = "My custom var"))</code>
<code>missing_action</code>	Character. How to handle missing values: • "keep": Keep as NA (default) • "drop": Remove rows with any missing values in processed variables
<code>invalid_codes</code>	Numeric vector of UKB codes to treat as missing. Default: <code>c(-1, -3)</code> which are "Prefer not to answer" and "Do not know"

Value

A data.table with original data plus processed variable columns

```
print.mediation_result
```

Print Method for Mediation Results

Description

Print mediation analysis results.

Usage

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

Arguments

<code>x</code>	An object of class "mediation_result".
<code>...</code>	Additional arguments passed to summary.

Value

Invisibly returns `x`, the original mediation result object.

protein_to_gene_symbol

Convert protein identifiers to gene symbols

Description

Convert a vector of protein identifiers into HGNC gene symbols for downstream enrichment analysis. When a custom mapping table is supplied, it is used first. Remaining unmatched identifiers can then be mapped with `clusterProfiler::bitr()`. Inputs in UK Biobank Olink coding 143 format, such as "IL6;Interleukin-6", and RAP-exported Olink column names such as "olink_instance_0.eno2" are parsed automatically. Multi-target Olink symbols such as "IL12A_IL12B" are expanded into one row per gene symbol.

Usage

```
protein_to_gene_symbol(
  proteins,
  protein_col = NULL,
  from_type = "SYMBOL",
  mapping_table = NULL,
  mapping_protein_col = "protein",
  mapping_symbol_col = "gene_symbol",
  organism_db = "org.Hs.eg.db",
  drop_unmapped = TRUE,
  input_format = c("auto", "rap_olink", "symbol", "raw")
)
```

Arguments

<code>proteins</code>	A character vector of protein identifiers, or a data.frame containing a protein identifier column.
<code>protein_col</code>	Optional column name when <code>proteins</code> is a data.frame.
<code>from_type</code>	Character string. Identifier type used by Bioconductor when <code>mapping_table</code> does not fully resolve the input. Default is "SYMBOL".
<code>mapping_table</code>	Optional data.frame containing custom protein-to-symbol mappings.
<code>mapping_protein_col</code>	Column name in <code>mapping_table</code> containing protein identifiers. Default is "protein".
<code>mapping_symbol_col</code>	Column name in <code>mapping_table</code> containing gene symbols. Default is "gene_symbol".
<code>organism_db</code>	Character string naming the OrgDb package. Default is "org.Hs.eg.db".
<code>drop_unmapped</code>	Logical. If TRUE, drop rows without a mapped gene symbol. Default is TRUE.

input_format Input naming convention. "auto" parses only recognized RAP Olink column names and Olink semicolon coding; "rap_olink" explicitly extracts the suffix after the final dot; "symbol" validates gene-symbol style identifiers; and "raw" preserves identifiers apart from trimming whitespace and converting to upper case.

Value

A data.frame with columns protein, gene_symbol, and mapping_source. Its mapping_summary attribute reports input, mapped, and unmapped protein counts.

rank_protein_ppi_nodes

Rank nodes in a PPI network by integrated centrality

Description

A thin wrapper around TCMDATA::rank_ppi_nodes().

Usage

```
rank_protein_ppi_nodes(
  ppi,
  metrics = c("degree", "betweenness", "closeness", "eccentricity", "radiality",
    "Stress", "MCC", "MNC", "DMNC", "BN", "EPC"),
  weights = NULL,
  use_weight = TRUE,
  na_rm = TRUE
)
```

Arguments

ppi	An igraph object, a list returned by get_protein_ppi(), or a list containing a graph element.
metrics	Character vector of node metrics used for integrated ranking.
weights	Optional numeric weights for metrics.
use_weight	Logical. Whether to prefer weighted betweenness and closeness metrics. Default is TRUE.
na_rm	Logical. Whether to ignore missing values during normalization. Default is TRUE.

Value

A list with components graph and table.

rap_extract_pheno	<i>Extract RAP Phenotype Data Synchronously</i>
-------------------	---

Description

Uses `dx extract_dataset --fields-file` and reads the RAP-generated result back into R within the active RAP session. This is intended for small to medium extractions. For large phenotype pulls, use `rap_submit_extract()`.

Usage

```
rap_extract_pheno(
  field_id = NULL,
  field_names = NULL,
  variables = NULL,
  dataset = NULL,
  output = NULL,
  read = TRUE,
  strip_entity_prefix = FALSE,
  dry_run = FALSE,
  timeout = 300,
  ...
)
```

Arguments

<code>field_id</code>	UKB numeric field IDs to extract.
<code>field_names</code>	Exact RAP dataset column names to extract.
<code>variables</code>	Optional predefined variable names from <code>get_variable_info()</code> .
<code>dataset</code>	Dataset file name. If <code>NULL</code> , <code>rap_find_dataset()</code> is used.
<code>output</code>	Optional CSV output path in the current RAP session. If <code>NULL</code> , a temporary file is used.
<code>read</code>	Logical. If <code>TRUE</code> , read the CSV into R and return a <code>data.table</code> . If <code>FALSE</code> , return the output path.
<code>strip_entity_prefix</code>	Logical. If <code>TRUE</code> , remove "participant." from returned column names.
<code>dry_run</code>	Logical. If <code>TRUE</code> , return the extraction plan without running <code>dx extract_dataset</code> .
<code>timeout</code>	Timeout in seconds for the extraction.
<code>...</code>	Additional arguments passed to <code>rap_plan_extract()</code> .

Value

A `data.table` when `read = TRUE`; otherwise the output CSV path. In dry-run mode, returns a `rap_extract_plan`.

rap_find_dataset	<i>Find the RAP Dataset File in the Current Project</i>
------------------	---

Description

Find the RAP Dataset File in the Current Project

Usage

```
rap_find_dataset(refresh = FALSE, timeout = 30)
```

Arguments

refresh	Logical. If TRUE, ignore the cached dataset name and call dx ls again.
timeout	Timeout in seconds for the dx ls call.

Value

A character scalar naming the detected .dataset file.

rap_list_fields	<i>List Approved RAP Dataset Fields</i>
-----------------	---

Description

List Approved RAP Dataset Fields

Usage

```
rap_list_fields(
  dataset = NULL,
  pattern = NULL,
  entity = "participant",
  refresh = FALSE,
  timeout = 120
)
```

Arguments

dataset	Dataset file name. If NULL, rap_find_dataset() is used.
pattern	Optional regular expression applied to field names and titles.
entity	Dataset entity. Defaults to "participant".
refresh	Logical. If TRUE, bypass the session cache.
timeout	Timeout in seconds for dx extract_dataset --list-fields.

Value

A data.frame with columns field_name and title.

rap_plan_extract	<i>Plan a RAP Phenotype Extraction</i>
------------------	--

Description

Plan a RAP Phenotype Extraction

Usage

```
rap_plan_extract(  
  field_id = NULL,  
  field_names = NULL,  
  variables = NULL,  
  dataset = NULL,  
  fields_df = NULL,  
  entity = "participant",  
  include_eid = TRUE,  
  table_exporter = FALSE,  
  manifest = NULL  
)
```

Arguments

- field_id UKB numeric field IDs to extract. All instances and arrays are included.
- field_names Exact RAP dataset column names, such as "participant.p31" or "p31".
- variables Optional predefined variable names from get_variable_info().
- dataset Dataset file name. If NULL, rap_find_dataset() is used.
- fields_df Optional cached field listing from rap_list_fields().
- entity Dataset entity. Defaults to "participant".
- include_eid Logical. Include participant ID automatically.
- table_exporter Logical. If TRUE, return field names in the format expected by the RAP table-exporter app.
- manifest Optional manifest CSV path in the current RAP session.

Value

A list containing extraction field names, matched requests, unmatched requests, dataset, entity, and column counts.

rap_plan_gp_query	<i>Plan a RAP Spark SQL GP Query</i>
-------------------	--------------------------------------

Description

Creates auditable Spark SQL that selects only requested Read v2 / CTV3 records from the stable RAP gp_clinical view. It does not execute the query or load GP records into R.

Usage

```
rap_plan_gp_query(
  query,
  database = NULL,
  table = "gp_clinical",
  cohort_table = NULL,
  cohort_database = database,
  collect = c("records", "summary")
)
```

Arguments

query	A ukb_gp_query from parse_gp_query() .
database	Optional RAP Spark database name. If NULL, the active database of the supplied Spark connection is used.
table	GP clinical table name. Defaults to the stable "gp_clinical" view.
cohort_table	Optional Spark table/view containing an eid column. When supplied, a left-semi join restricts GP records before collection.
cohort_database	Optional database containing cohort_table. Defaults to database.
collect	Return record-level SQL ("records") or push participant by disease aggregation into Spark ("summary").

Value

A ukb_gp_query_plan containing the SQL and normalized query.

rap_run_gp_query	<i>Run a RAP Spark SQL GP Query</i>
------------------	-------------------------------------

Description

Executes a planned GP query through an existing Spark DBI connection. The function adds a row guard before collecting results into the R worker. It is intended for a Spark-enabled UK Biobank RAP R notebook.

Usage

```
rap_run_gp_query(
  plan,
  connection = NULL,
  output = NULL,
  max_rows = 1000000L,
  dry_run = FALSE
)
```

Arguments

plan	A plan from <code>rap_plan_gp_query()</code> .
connection	An active Spark DBI connection, commonly created with <code>sparklyr::spark_connect()</code> .
output	Optional .csv or .tsv path for the filtered records.
max_rows	Maximum rows allowed to be collected into R. The executed SQL requests at most <code>max_rows + 1</code> rows and stops if the limit is exceeded.
dry_run	If TRUE, return the SQL without executing it.

Value

A data.table containing filtered GP records or a participant-disease summary, according to `plan$collect`. If output is supplied, the same result is also written to disk.

rap_submit_extract	<i>Submit a RAP Table-Exporter Phenotype Extraction Job</i>
--------------------	---

Description

Submits an asynchronous RAP table-exporter job. This is the preferred interface for large extraction jobs because the work runs on RAP rather than inside the current R session.

Usage

```
rap_submit_extract(
  field_id = NULL,
  field_names = NULL,
  variables = NULL,
  dataset = NULL,
  file = NULL,
  instance_type = NULL,
  priority = c("low", "high"),
  dry_run = FALSE,
  manifest = NULL,
  ...
)
```

Arguments

<code>field_id</code>	UKB numeric field IDs to extract.
<code>field_names</code>	Exact RAP dataset column names to extract.
<code>variables</code>	Optional predefined variable names from <code>get_variable_info()</code> .
<code>dataset</code>	Dataset file name. If <code>NULL</code> , <code>rap_find_dataset()</code> is used.
<code>file</code>	Output file stem on RAP. Defaults to "ukba_pheno_YYYYMMDD_HHMMSS".
<code>instance_type</code>	DNAnexus instance type. If <code>NULL</code> , selected from the number of columns.
<code>priority</code>	Job priority: "low" or "high".
<code>dry_run</code>	Logical. If <code>TRUE</code> , return the planned fields and command metadata without uploading or submitting.
<code>manifest</code>	Optional manifest CSV path in the current RAP session.
<code>...</code>	Additional arguments passed to <code>rap_plan_extract()</code> .

Value

A list with class `rap_extract_job` containing job metadata. In dry-run mode, returns a `rap_extract_plan`.

<code>read_prs_scores</code>	<i>Read and Combine PLINK2 PRS Scores</i>
------------------------------	---

Description

Read and Combine PLINK2 PRS Scores

Usage

```
read_prs_scores(  
  files,  
  score_col = NULL,  
  score_name = "PRS",  
  standardize = FALSE  
)
```

Arguments

<code>files</code>	One or more PLINK2 .sscore files.
<code>score_col</code>	Optional score-sum column name(s). By default, all columns ending in <code>_SUM</code> are used.
<code>score_name</code>	Output score name(s). When omitted for a multi-score file, names are derived from the <code>_SUM</code> column headers.
<code>standardize</code>	Add a z-score column.

Value

A `data.table` with FID, IID, score sums, allele denominator, and number of combined files. `PRS_AVG` is retained for a single score.

run_correlation	<i>Calculate correlation between variables</i>
-----------------	--

Description

Before the formal regression analysis, it can be useful to check the correlation between variables. This function calculates the correlation matrix for a set of specified variables, which can help identify potential multicollinearity issues or inform variable selection.

Usage

```
run_correlation(df, vars, method = "pearson", threshold = 0.7)
```

Arguments

df	A data.frame or data.table containing the variables of interest.
vars	A character vector of column names for which to calculate the correlation matrix.
method	The method to use for calculating correlation. Options are "pearson", "spearman", or "kendall". Default is "pearson".
threshold	Numeric value between 0 and 1. If specified, the variables with absolute correlation above this threshold will be highlighted in the output. Default is 0.7.

Value

A correlation matrix of the specified variables.

run_gp_workflow	<i>Run an End-to-End UKB GP Phenotype Workflow</i>
-----------------	--

Description

Runs the independent GP workflow from a parsed disease query through RAP Spark SQL extraction, clinical/registration parsing, coverage summary, and participant-level phenotype integration. The query is the first argument so the function can follow [parse_gp_query\(\)](#) with the base R pipe (`|>`).

Usage

```
run_gp_workflow(  
  query,  
  connection = NULL,  
  database = NULL,  
  participants = NULL,  
  observation_end = NULL,
```

```

clinical_data = NULL,
registration_data = NULL,
clinical_output = NULL,
registration_output = NULL,
max_clinical_rows = 1000000L,
max_registration_rows = 1000000L,
dry_run = FALSE,
cohort_table = NULL,
cohort_database = database,
collect = c("summary", "records"),
window_start = NULL,
window_end = NULL,
index_date = NULL,
min_lookback_days = 0L,
min_followup_days = 0L,
min_coverage_fraction = 1
)

```

Arguments

query	A ukb_gp_query from parse_gp_query() .
connection	An active RAP Spark DBI connection. Not required when both <code>clinical_data</code> and <code>registration_data</code> are supplied for local testing.
database	Optional RAP Spark database name.
participants	Optional participant vector or data.frame with eid.
observation_end	Optional study/data-release cut-off used to close open registration intervals.
clinical_data	Optional pre-extracted gp_clinical records.
registration_data	Optional pre-extracted gp_registrations records.
clinical_output	Optional local CSV/TSV path for filtered clinical records.
registration_output	Optional local CSV/TSV path for registration records.
max_clinical_rows	Maximum clinical records collected into R.
max_registration_rows	Maximum registration records collected into R.
dry_run	Return both SQL plans without executing them.
cohort_table	Optional Spark cohort table/view containing eid. Both clinical and registration queries use a left-semi join to this table.
cohort_database	Optional database containing cohort_table. Defaults to database.
collect	RAP clinical collection mode. "summary" performs disease aggregation in Spark; "records" collects matched record-level rows.

window_start	Optional study-window start passed to <code>assess_gp_observability()</code> . Supply together with window_end to enable strict control eligibility.
window_end	Optional study-window end.
index_date	Optional participant index/baseline date.
min_lookback_days	Minimum continuous coverage before index_date.
min_followup_days	Minimum continuous coverage after index_date.
min_coverage_fraction	Minimum covered fraction of the study window.

Value

A participant-by-disease GP phenotype data.table, or a dry-run plan.

run_imputation	<i>Multiple imputation and merge back to full data</i>
----------------	--

Description

Run multiple imputation with the CRAN package **mice** on a subset of variables, then merge the imputed columns back to the original dataset by an ID column.

Usage

```
run_imputation(
  data,
  id_col = "eid",
  vars,
  factor_vars = NULL,
  method = "pmm",
  m = 5,
  maxit = 10,
  seed = 1234,
  print = TRUE,
  additional_data = NULL,
  additional_join = c("inner", "left")
)
```

Arguments

data	A data.frame/data.table containing the cohort.
id_col	Name of the ID column. Default is "eid".
vars	Character vector of column names to impute.
factor_vars	Optional character vector of variables (subset of vars) to treat as categorical (factors).

<code>method</code>	Imputation method passed to <code>mice()</code> . Default is "pmm".
<code>m</code>	Number of multiple imputations. Default is 5.
<code>maxit</code>	Maximum number of iterations. Default is 10.
<code>seed</code>	Random seed for reproducibility.
<code>print</code>	Logical. If TRUE, show mice iteration logs.
<code>additional_data</code>	Optional named list of extra datasets to merge after imputation. Each element must contain <code>id_col</code> . Example: <code>list(protein = protein_df, metabolomics = meta_df)</code> .
<code>additional_join</code>	Join type for additional datasets. One of "inner" or "left". Default is "inner".

Details

This function is designed for workflows where you want to keep a set of "static" columns (exposures, outcomes, follow-up time, etc.) untouched while imputing a selected set of covariates.

The function:

1. Subsets the input data to the requested variables.
2. Runs `mice()`.
3. Creates `m` completed datasets and merges imputed columns back.
4. Optionally merges additional datasets (e.g., omics) by ID.

Factor handling: for variables listed in `factor_vars`, the function will coerce them to factors before imputation. All other variables in `vars` are coerced to numeric.

Value

A list with:

- `imp`: the mice `mids` object
- `data_list`: a list of length `m` containing completed and merged datasets

References

<https://github.com/amices/mice>

run_mediation

*Run Causal Mediation Analysis***Description**

Perform regression-based causal mediation analysis using the regmedint package. Supports linear, logistic, and Cox proportional hazards models for the outcome, and linear or logistic models for the mediator.

Usage

```
run_mediation(
  data,
  exposure,
  mediator,
  outcome,
  covariates = NULL,
  exposure_levels = c(0, 1),
  mediator_value = 0,
  covariate_values = NULL,
  mediator_type = c("continuous", "binary"),
  outcome_type = c("linear", "logistic", "cox"),
  endpoint = NULL,
  interaction = TRUE,
  boot = FALSE,
  boot_n = 1000,
  conf_level = 0.95
)
```

Arguments

data	A data.frame or data.table containing all variables.
exposure	Character string specifying the exposure (treatment) variable name.
mediator	Character string specifying the mediator variable name.
outcome	Character string specifying the outcome variable name. For Cox models, this should be the time variable.
covariates	Character vector of covariate names. Default NULL.
exposure_levels	Numeric vector of length 2: c(a0, a1) where a0 is the reference level and a1 is the comparison level. Default c(0, 1).
mediator_value	Numeric value at which to evaluate the controlled direct effect (CDE). Default 0.
covariate_values	Numeric vector of covariate values at which to evaluate conditional effects. If NULL, uses mean (continuous) or mode (categorical).

<code>mediator_type</code>	Character string: "continuous" or "binary". Default "continuous".
<code>outcome_type</code>	Character string: "linear", "logistic", or "cox". Default "linear".
<code>endpoint</code>	Character vector of length 2 for Cox models: <code>c("time_col", "status_col")</code> . Required when <code>outcome_type = "cox"</code> .
<code>interaction</code>	Logical; whether to include exposure-mediator interaction in the outcome model. Default TRUE.
<code>boot</code>	Logical; whether to use bootstrap for confidence intervals. Default FALSE.
<code>boot_n</code>	Integer; number of bootstrap replicates. Default 1000.
<code>conf_level</code>	Numeric; confidence level. Default 0.95.

Details

This function wraps the `regmedint` package to provide a user-friendly interface for causal mediation analysis. It implements the methods described in Valeri & VanderWeele (2013, 2015).

Effect definitions:

- `cde`: Controlled Direct Effect - effect of exposure with mediator fixed
- `pnde`: Pure Natural Direct Effect - direct effect (traditional NDE)
- `tnie`: Total Natural Indirect Effect - indirect effect (traditional NIE)
- `tnde`: Total Natural Direct Effect
- `pnie`: Pure Natural Indirect Effect
- `te`: Total Effect = NDE + NIE
- `pm`: Proportion Mediated = NIE / TE

Value

An object of class "mediation_result" containing:

effects data.frame with effect estimates, SE, CI, and p-values

mediator_model Fitted mediator model object

outcome_model Fitted outcome model object

regmedint_obj Original `regmedint` object (if available)

call The matched call

params List of analysis parameters

References

Valeri L, VanderWeele TJ. Mediation analysis allowing for exposure-mediator interactions and causal interpretation. *Psychological Methods*. 2013;18(2):137-150.

run_metabolite_or	<i>Run metabolite over-representation analysis</i>
-------------------	--

Description

Run ORA for a metabolite list. The recommended first backend is backend = "custom", where users provide a two-column metabolite pathway library. A backend = "metaboanalyst" interface is also provided for users who have installed MetaboAnalystR and want to use its metabolite-set libraries, such as "smpdb_pathway".

Usage

```
run_metabolite_or(
  metabolites,
  pathway_db = NULL,
  universe = NULL,
  backend = c("custom", "metaboanalyst"),
  id_type = "name",
  library = "smpdb_pathway",
  mapping_table = NULL,
  pathway_col = "pathway",
  metabolite_col = "metabolite",
  min_metabolites = 3,
  p_adjust_method = "BH",
  run_subprocess = TRUE
)
```

Arguments

metabolites	Character vector of metabolite names.
pathway_db	Optional data.frame for custom ORA. Must contain pathway and metabolite columns.
universe	Optional background metabolite vector. If NULL, the pathway library metabolites are used for custom ORA.
backend	One of "custom" or "metaboanalyst".
id_type	Metabolite identifier type for MetaboAnalystR cross-referencing. Default "name".
library	MetaboAnalystR metabolite-set library. Default "smpdb_pathway".
mapping_table	Optional custom mapping table passed to metabolite_to_metaboanalyst_name().
pathway_col	Column name in pathway_db containing pathway names. Default "pathway".
metabolite_col	Column name in pathway_db containing metabolite names. Default "metabolite".
min_metabolites	Minimum mapped metabolites required for ORA. Default 3.
p_adjust_method	Multiple-testing method used by <code>p.adjust()</code> . Default "BH".
run_subprocess	Logical. For backend = "metaboanalyst", run MetaboAnalystR in a clean subprocess to avoid global-state issues. Default TRUE.

Value

A list of class `ukb_metabolite_ora` with components `input`, `mapping`, `matched`, `unmatched`, `ora_result`, `backend`, `library`, and `mapping_summary`.

Examples

```
panel <- load_ukb_metabolite_panel()
hits <- c("Alanine", "Glutamine", "Glycine", "Lactate", "Pyruvate")
pathway_db <- data.frame(
  pathway = c(rep("Amino acid metabolism", 3), rep("Energy metabolism", 2)),
  metabolite = c("L-Alanine", "L-Glutamine", "Glycine", "Lactic acid", "Pyruvic acid")
)
run_metabolite_ora(hits, pathway_db = pathway_db, backend = "custom")
```

<code>run_multi_mediator</code>	<i>Run Multiple Mediator Analysis</i>
---------------------------------	---------------------------------------

Description

Perform mediation analysis for multiple potential mediators, testing each one separately.

Usage

```
run_multi_mediator(
  data,
  exposure,
  mediators,
  outcome,
  covariates = NULL,
  mediator_type = "continuous",
  outcome_type = "linear",
  endpoint = NULL,
  ...
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> containing all variables.
<code>exposure</code>	Character string specifying the exposure (treatment) variable name.
<code>mediators</code>	Character vector of mediator variable names.
<code>outcome</code>	Character string specifying the outcome variable name. For Cox models, this should be the time variable.
<code>covariates</code>	Character vector of covariate names. Default <code>NULL</code> .
<code>mediator_type</code>	Character string: "continuous" or "binary". Default "continuous".
<code>outcome_type</code>	Character string: "linear", "logistic", or "cox". Default "linear".

endpoint	Character vector of length 2 for Cox models: c("time_col", "status_col"). Required when outcome_type = "cox".
...	Additional arguments passed to run_mediation().

Value

A data.frame with mediation results for each mediator, including:

mediator Mediator variable name
tnie Total natural indirect effect estimate
tnie_se Standard error of TNIE
tnie_p P-value for TNIE
pnde Pure natural direct effect estimate
te Total effect estimate
pm Proportion mediated
pm_se Standard error of proportion mediated

run_multi_subgroup	<i>Run Multiple Subgroup Analyses</i>
--------------------	---------------------------------------

Description

Perform subgroup analyses across multiple subgroup variables.

Usage

```
run_multi_subgroup(
  data,
  exposure,
  outcome = NULL,
  subgroup_vars,
  covariates = NULL,
  model_type = c("cox", "logistic", "linear", "glm", "negbin"),
  family = "poisson",
  endpoint = NULL
)
```

Arguments

data	A data.frame or data.table containing all variables.
exposure	Character string specifying the exposure variable name.
outcome	Character string specifying the outcome variable name. For Cox models, this can be NULL if endpoint is specified.
subgroup_vars	Character vector of subgroup variable names.

covariates	Character vector of covariate names to adjust for. Default NULL.
model_type	Character string specifying model type: "cox", "logistic", "linear", "glm", or "negbin".
family	For model_type = "glm" only: the GLM family. Accepts a character string, function, or family object (see runmulti_glm). Default "poisson".
endpoint	Character vector of length 2 for Cox models: c("time", "status"). Required when model_type = "cox".

Value

A data.frame with results from all subgroup analyses combined.

run_protein_kegg_ora	<i>Run KEGG ORA enrichment for proteomics hits</i>
----------------------	--

Description

Convert protein identifiers to gene symbols, then to Entrez IDs, and run over-representation analysis (ORA) with `clusterProfiler::enrichKEGG()`.

Usage

```
run_protein_kegg_ora(
  proteins,
  protein_col = NULL,
  from_type = "SYMBOL",
  mapping_table = NULL,
  mapping_protein_col = "protein",
  mapping_symbol_col = "gene_symbol",
  universe = NULL,
  organism_db = "org.Hs.eg.db",
  organism = "hsa",
  pvalueCutoff = 0.05,
  qvalueCutoff = 0.2,
  pAdjustMethod = "BH",
  minGSSize = 10,
  maxGSSize = 500,
  readable = TRUE,
  use_internal_data = FALSE,
  input_format = c("auto", "rap_olink", "symbol", "raw")
)
```

Arguments

proteins	A character vector of protein identifiers, or a data.frame containing a protein identifier column.
protein_col	Optional column name when proteins is a data.frame.
from_type	Character string describing the input identifier type for Bioconductor-based mapping. Default is "SYMBOL".
mapping_table	Optional data.frame containing custom protein-to-symbol mappings.
mapping_protein_col	Column name in mapping_table containing protein identifiers. Default is "protein".
mapping_symbol_col	Column name in mapping_table containing gene symbols. Default is "gene_symbol".
universe	Optional character vector of background protein identifiers. These identifiers are converted with the same rules as proteins.
organism_db	Character string naming the OrgDb package. Default is "org.Hs.eg.db".
organism	Character string for KEGG organism code. Default is "hsa".
pvalueCutoff	Numeric p-value cutoff for ORA. Default is 0.05.
qvalueCutoff	Numeric q-value cutoff for ORA. Default is 0.2.
pAdjustMethod	Character string for multiple-testing correction. Default is "BH".
minGSSize	Minimum gene set size. Default is 10.
maxGSSize	Maximum gene set size. Default is 500.
readable	Logical. Passed to clusterProfiler::enrichGO(). Default is TRUE.
use_internal_data	Logical. Passed to clusterProfiler::enrichKEGG(). Default is FALSE.
input_format	Input naming convention passed to protein_to_gene_symbol() .

Value

A list with components gene_symbols, entrez_ids, mapping, mapping_summary, universe_symbols, universe_entrez_ids, and ora_result.

run_protein_ora	<i>Run GO ORA enrichment for proteomics hits</i>
-----------------	--

Description

Convert protein identifiers to gene symbols and run over-representation analysis (ORA) with clusterProfiler::enrichGO(). This is the GO-specific interface for proteomics hits extracted from UK Biobank RAP Olink data.

Usage

```
run_protein_ora(
  proteins,
  protein_col = NULL,
  from_type = "SYMBOL",
  mapping_table = NULL,
  mapping_protein_col = "protein",
  mapping_symbol_col = "gene_symbol",
  universe = NULL,
  organism_db = "org.Hs.eg.db",
  ont = "BP",
  pvalueCutoff = 0.05,
  qvalueCutoff = 0.2,
  pAdjustMethod = "BH",
  minGSSize = 10,
  maxGSSize = 500,
  readable = TRUE,
  input_format = c("auto", "rap_olink", "symbol", "raw")
)
```

Arguments

proteins	A character vector of protein identifiers, or a data.frame containing a protein identifier column.
protein_col	Optional column name when proteins is a data.frame.
from_type	Character string describing the input identifier type for Bioconductor-based mapping. Default is "SYMBOL".
mapping_table	Optional data.frame containing custom protein-to-symbol mappings.
mapping_protein_col	Column name in mapping_table containing protein identifiers. Default is "protein".
mapping_symbol_col	Column name in mapping_table containing gene symbols. Default is "gene_symbol".
universe	Optional character vector of background protein identifiers. These identifiers are converted with the same rules as proteins.
organism_db	Character string naming the OrgDb package. Default is "org.Hs.eg.db".
ont	One of "BP", "MF", "CC", or "ALL". Passed to clusterProfiler::enrichGO().
pvalueCutoff	Numeric p-value cutoff for ORA. Default is 0.05.
qvalueCutoff	Numeric q-value cutoff for ORA. Default is 0.2.
pAdjustMethod	Character string for multiple-testing correction. Default is "BH".
minGSSize	Minimum gene set size. Default is 10.
maxGSSize	Maximum gene set size. Default is 500.
readable	Logical. Passed to clusterProfiler::enrichGO(). Default is TRUE.
input_format	Input naming convention passed to protein_to_gene_symbol() .

Value

A list with components `gene_symbols`, `mapping`, `mapping_summary`, `universe_symbols`, and `ora_result`.

`run_protein_ppi_clustering`*Cluster a protein-protein interaction network*

Description

Unified interface for community detection in STRING-derived PPI networks. New analyses should call this function and choose the algorithm with `method`. Method-specific helper functions are retained internally.

Usage

```
run_protein_ppi_clustering(  
  ppi,  
  method = c("fastgreedy", "louvain", "mcode", "mcl"),  
  ...  
)
```

Arguments

<code>ppi</code>	An igraph object, a list returned by <code>get_protein_ppi()</code> , or a list containing a graph element.
<code>method</code>	Clustering algorithm. Options are "fastgreedy", "louvain", "mcode", and "mcl".
<code>...</code>	Method-specific arguments passed to the selected clustering helper, such as <code>n_clusters</code> for "fastgreedy", <code>resolution</code> for "louvain", <code>vwp</code> for "mcode", or <code>inflation</code> for "mcl".

Value

An igraph object with method-specific cluster attributes.

run_protein_ppi_robustness

Evaluate PPI network robustness for selected protein targets

Description

Convert target protein identifiers to gene symbols and run STRING-network robustness analysis via TCMDATA::ppi_knock().

Usage

```
run_protein_ppi_robustness(
  ppi,
  targets,
  target_col = NULL,
  from_type = "SYMBOL",
  mapping_table = NULL,
  mapping_protein_col = "protein",
  mapping_symbol_col = "gene_symbol",
  organism_db = "org.Hs.eg.db",
  n_perm = 100L,
  weight_attr = "score",
  rewired_niter = 10L,
  seed = 42L,
  input_format = c("auto", "rap_olink", "symbol", "raw")
)
```

Arguments

ppi	An igraph object, a list returned by get_protein_ppi(), or a list containing a graph element.
targets	A character vector of target protein identifiers, or a data.frame containing a target identifier column.
target_col	Optional column name when targets is a data.frame.
from_type	Character string describing the input identifier type for Bioconductor-based mapping. Default is "SYMBOL".
mapping_table	Optional data.frame containing custom protein-to-symbol mappings.
mapping_protein_col	Column name in mapping_table containing protein identifiers. Default is "protein".
mapping_symbol_col	Column name in mapping_table containing gene symbols. Default is "gene_symbol".
organism_db	Character string naming the OrgDb package. Default is "org.Hs.eg.db".
n_perm	Integer. Number of permutation iterations. Default is 100.
weight_attr	Character. Edge attribute containing the confidence score. Default is "score".

rewire_niter	Integer. Rewiring multiplier used in the null model. Default is 10.
seed	Integer random seed. Default is 42.
input_format	Input naming convention passed to <code>protein_to_gene_symbol()</code> .

Value

A list with components `targets`, `mapping`, and `robustness`.

run_rcs	<i>Fit a restricted cubic spline exposure-response model</i>
---------	--

Description

Fits a restricted cubic spline (RCS) model to characterise nonlinear exposure-response relationships. Supports Cox, logistic, and linear regression. Returns prediction curves, confidence intervals, overall and nonlinear P values, and the AIC-selected knot count. The returned object is passed directly to `plot_rcs()` for publication-ready figures.

Usage

```
run_rcs(
  data,
  exposure,
  covariates = NULL,
  model_type = c("cox", "logistic", "linear"),
  endpoint = NULL,
  outcome = NULL,
  knots = NULL,
  knot_range = 3:7,
  ref = NULL,
  ref_quantile = 0.5,
  conf_level = 0.95,
  trim_quantiles = c(0.01, 0.99),
  grid_size = 200L,
  backend = c("rms", "ns")
)
```

Arguments

<code>data</code>	A <code>data.frame</code> containing all required columns.
<code>exposure</code>	Character. Name of the continuous exposure variable.
<code>covariates</code>	Character vector of covariate names, or <code>NULL</code> .
<code>model_type</code>	One of <code>"cox"</code> , <code>"logistic"</code> , <code>"linear"</code> .
<code>endpoint</code>	Character vector of length 2 giving <code>c(time, status)</code> column names. Required when <code>model_type = "cox"</code> .

<code>outcome</code>	Character. Outcome column name. Required for "logistic" and "linear".
<code>knots</code>	Integer. Number of knots (3-7). If NULL, the knot count with the lowest AIC within <code>knot_range</code> is chosen automatically.
<code>knot_range</code>	Integer vector of candidate knot counts for AIC selection. Default 3:7.
<code>ref</code>	Numeric. Reference value for the exposure. If NULL, <code>ref_quantile</code> is used.
<code>ref_quantile</code>	Numeric (0-1). Quantile of the exposure used as the reference when <code>ref</code> is NULL. Default 0.5 (median).
<code>conf_level</code>	Numeric. Confidence level for intervals. Default 0.95.
<code>trim_quantiles</code>	Numeric vector of length 2. Exposure values outside these quantiles are excluded before fitting. Default <code>c(0.01, 0.99)</code> .
<code>grid_size</code>	Integer. Number of points in the prediction grid. Default 200.
<code>backend</code>	One of "rms" (default, requires the rms package) or "ns" (base-R natural cubic splines, no additional dependencies).

Value

An object of class `c("ukb_rcs", "list")` with elements:

model The fitted model object.

model_type Character. One of "cox", "logistic", "linear".

backend Character. "rms" or "ns".

exposure Character. Name of the exposure variable.

covariates Character vector of covariate names.

endpoint Character vector. Cox endpoint columns.

outcome Character. Outcome column name.

knots Integer. Number of knots used.

ref Numeric. Reference exposure value.

n Integer. Number of observations in the fitted model.

n_event Integer. Number of events (Cox only, else NA).

p_overall Numeric. Overall P value for the exposure term.

p_nonlinear Numeric. P value for the nonlinear component.

prediction data.frame with columns `x`, `estimate`, `lower95`, `upper95`.

distribution data.frame with column `x` (untrimmed exposure values).

aic_table data.frame with columns `knots` and `AIC`.

run_regression	<i>Run a regression model (unified interface)</i>
----------------	---

Description

A unified wrapper around `runmulti_cox`, `runmulti_lm`, `runmulti_logit`, `runmulti_glm`, `runmulti_negbin`, and `runmulti_gam`. Select the model family with `type`.

Usage

```
run_regression(
  data,
  main_var,
  type = c("cox", "lm", "logit", "glm", "negbin", "gam"),
  outcome = NULL,
  endpoint = c("time", "status"),
  covariates = NULL,
  covariate_sets = NULL,
  family = NULL,
  smooth = TRUE,
  ...
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> containing all variables.
<code>main_var</code>	A character vector of main variable names to test.
<code>type</code>	One of "cox", "lm", "logit", "glm", "negbin", or "gam".
<code>outcome</code>	For all types except "cox": a character string naming the outcome column.
<code>endpoint</code>	For "cox": a character vector of length 2 <code>c("time", "status")</code> . Ignored for other types.
<code>covariates</code>	A character vector of covariate names. Default <code>NULL</code> .
<code>covariate_sets</code>	Optional named list of covariate sets for nested epidemiological models. Each element must be <code>NULL</code> or a character vector of covariate names. When supplied, <code>run_regression()</code> runs the same exposure-outcome model once per covariate set and returns one stacked table with a <code>model</code> column.
<code>family</code>	For "glm" and "gam": the model family. Accepts a character string, function, or family object. Default "poisson" for "glm" and "gaussian" for "gam". See runmulti_glm for details.
<code>smooth</code>	For "gam" only: logical. Use a smooth spline term (<code>TRUE</code> , default) or a linear term (<code>FALSE</code>).
<code>...</code>	Additional arguments forwarded to the underlying fitting function.

Value

A data.frame whose columns depend on type:

cox variable, coef, se, z, HR, lower95, upper95, pvalue, n, n_event

lm variable, beta, lower95, upper95, pvalue

logit variable, OR, lower95, upper95, pvalue

glm variable, family, link, beta, lower95, upper95, pvalue, n

negbin variable, IRR, lower95, upper95, pvalue, theta, n

gam (smooth) variable, edf, ref_df, F, pvalue, family, link, n

gam (linear) variable, beta, lower95, upper95, pvalue, family, link, n

run_subgroup_analysis *Run Subgroup Analysis*

Description

Perform subgroup analysis by fitting regression models within each level of a subgroup variable and calculating interaction p-values.

Usage

```
run_subgroup_analysis(
  data,
  exposure,
  outcome = NULL,
  subgroup_var,
  covariates = NULL,
  model_type = c("cox", "logistic", "linear", "glm", "negbin"),
  family = "poisson",
  endpoint = NULL,
  ref_level = NULL
)
```

Arguments

data	A data.frame or data.table containing all variables.
exposure	Character string specifying the exposure variable name.
outcome	Character string specifying the outcome variable name. For Cox models, this can be NULL if endpoint is specified.
subgroup_var	Character string specifying the subgroup variable name.
covariates	Character vector of covariate names to adjust for. Default NULL.
model_type	Character string specifying model type: "cox", "logistic", "linear", "glm", or "negbin".

family	For <code>model_type = "glm"</code> only: the GLM family. Accepts a character string, function, or family object (see runmulti_glm). Default <code>"poisson"</code> .
endpoint	Character vector of length 2 for Cox models: <code>c("time", "status")</code> . Required when <code>model_type = "cox"</code> .
ref_level	Character string specifying the reference level for the subgroup variable. If <code>NULL</code> , the first level is used as reference.

Value

A `data.frame` with columns:

subgroup_var Name of the subgroup variable
subgroup Subgroup level
n Sample size in subgroup
n_event Number of events (for Cox/logistic models)
estimate Effect estimate (HR for Cox, OR for logistic, Beta for linear)
lower95 Lower 95% CI
upper95 Upper 95% CI
pvalue P-value for the exposure effect
p_interaction P-value for interaction between exposure and subgroup

run_weighted_analysis *Run Weighted Analysis*

Description

Fit regression models using IPTW weights with robust standard errors.

Usage

```
run_weighted_analysis(
  data,
  exposure,
  outcome = NULL,
  covariates = NULL,
  weight_col = "weight",
  model_type = c("cox", "logistic", "linear"),
  endpoint = NULL,
  robust_se = TRUE
)
```

Arguments

data	A data.frame or data.table containing all variables and weights.
exposure	Character string specifying the exposure variable name.
outcome	Character string specifying the outcome variable name (for logistic/linear).
covariates	Character vector of covariate names. Default NULL.
weight_col	Character string specifying the weight column name. Default "weight".
model_type	Character string specifying model type: "cox", "logistic", or "linear".
endpoint	Character vector of length 2 for Cox models: c("time", "status").
robust_se	Logical; whether to use robust standard errors. Default TRUE.

Value

A data.frame with effect estimates and confidence intervals.

runmulti_competing	<i>Run Multiple Fine-Gray Competing-Risk Models</i>
--------------------	---

Description

Run Multiple Fine-Gray Competing-Risk Models

Usage

```
runmulti_competing(
  data,
  main_var,
  covariates = NULL,
  time_col,
  event_col,
  compete_col = NULL,
  event_value = 1,
  compete_value = 2,
  conf_level = 0.95,
  ...
)
```

Arguments

data	A data.frame or data.table.
main_var	Character vector of exposure variable names.
covariates	Optional character vector of covariates.
time_col	Follow-up time column.
event_col	Event-status column, or the primary-event column in dual-column mode.

compete_col	Optional competing-event column in dual-column mode.
event_value	Event code used in single-column mode.
compete_value	Competing-event code used in single-column mode.
conf_level	Confidence level, reserved for future use.
...	Additional arguments passed to the weighted Cox fit.

Value

A data.frame with subdistribution hazard ratios.

runmulti_cox	<i>Run multiple Cox proportional hazards models</i>
--------------	---

Description

Fit Cox proportional hazards models for each main variable separately. When `covariates` is `NULL`, univariate models are fitted. Otherwise, multivariate models adjusting for the specified covariates are fitted.

Usage

```
runmulti_cox(
  data,
  main_var,
  covariates = NULL,
  endpoint = c("time", "status"),
  ...
)
```

Arguments

<code>data</code>	A data.frame or data.table containing all variables.
<code>main_var</code>	A character vector of main variable names to test.
<code>covariates</code>	A character vector of covariate names to adjust for. Default <code>NULL</code> (univariate).
<code>endpoint</code>	A character vector of length 2: <code>c("time", "status")</code> , indicating survival time and event columns.
<code>...</code>	Additional arguments passed to <code>coxph()</code> .

Value

A data.frame with columns: `variable`, `contrast`, `coef`, `se`, `z`, `HR`, `lower95`, `upper95`, `pvalue`, `n`, and `n_event`.

runmulti_cox_lag	<i>Run Lagged Cox Sensitivity Analyses</i>
------------------	--

Description

Run Lagged Cox Sensitivity Analyses

Usage

```
runmulti_cox_lag(  
  data,  
  main_var,  
  covariates = NULL,  
  endpoint = c("time", "status"),  
  lag_years = c(0, 1, 2, 5),  
  verbose = TRUE,  
  ...  
)
```

Arguments

- data A data.frame or data.table.
- main_var Character vector of exposure variable names.
- covariates Optional character vector of covariates.
- endpoint Character vector of length 2: c(time, status).
- lag_years Numeric vector of lag windows in years. 0 means no filtering.
- verbose Logical; print progress messages.
- ... Additional arguments passed to coxph().

Value

A data.frame containing lag-specific hazard-ratio estimates.

runmulti_cox_zph	<i>Run Multiple Cox Models with PH Diagnostics</i>
------------------	--

Description

Run Multiple Cox Models with PH Diagnostics

Usage

```
runmulti_cox_zph(
  data,
  main_var,
  covariates = NULL,
  endpoint = c("time", "status"),
  transform = c("km", "rank", "identity"),
  alpha = 0.05,
  keep_models = FALSE,
  ...
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> .
<code>main_var</code>	Character vector of exposure variable names.
<code>covariates</code>	Optional character vector of covariate names.
<code>endpoint</code>	Character vector of length 2: <code>c(time, status)</code> .
<code>transform</code>	Character scalar passed to <code>cox.zph()</code> .
<code>alpha</code>	Numeric threshold for flagging PH violations.
<code>keep_models</code>	Logical; if <code>TRUE</code> , attach fitted models as an attribute.
<code>...</code>	Additional arguments passed to <code>coxph()</code> .

Value

A `data.frame` with effect estimates and PH-diagnostic columns.

runmulti_gam

Run multiple generalised additive models

Description

Fit GAMs (`gam`) for each main variable separately. By default each main variable enters the model as a penalised thin-plate regression spline `s(var)`, allowing non-linear dose-response relationships to be detected.

When `smooth = TRUE` (default) the returned table reports the smooth term's estimated degrees of freedom (`edf`), `F`-statistic, and `p`-value - useful for screening whether an association exists and whether it is non-linear (`edf > 1`). When `smooth = FALSE` the main variable enters as a parametric linear term and the output mirrors `runmulti_glm(beta, Wald CI, p-value)`.

Usage

```
runmulti_gam(
  data,
  main_var,
  outcome,
  covariates = NULL,
  smooth = TRUE,
  family = "gaussian",
  k = -1,
  ...
)
```

Arguments

<code>data</code>	A data.frame or data.table containing all variables.
<code>main_var</code>	A character vector of main variable names to test.
<code>outcome</code>	A character string specifying the outcome column.
<code>covariates</code>	A character vector of covariate names added as parametric linear terms. Default NULL.
<code>smooth</code>	Logical. If TRUE (default) the main variable is modelled as <code>s(var)</code> . If FALSE it is treated as a linear term.
<code>family</code>	A GLM family controlling the response distribution. Accepts the same forms as runmulti_glm : character string, function, or family object. Default "gaussian".
<code>k</code>	Integer. Basis dimension for each smooth term. -1 (default) lets mgcv choose automatically.
<code>...</code>	Additional arguments passed to <code>gam()</code> .

Value

When `smooth = TRUE`: a data.frame with columns `variable`, `edf`, `ref_df`, `F`, `pvalue`, `family`, `link`, `n`. When `smooth = FALSE`: `variable`, `beta`, `lower95`, `upper95`, `pvalue`, `family`, `link`, `n`.

runmulti_glm

Run multiple generalised linear models

Description

Fit GLMs for each main variable separately using any stats family. When `covariates` is NULL, univariate models are fitted. Otherwise, multivariate models are fitted.

Quasi-families (quasipoisson, quasibinomial) use Wald confidence intervals because profile-likelihood CIs are not available for quasi-likelihood models. All other families use profile-likelihood CIs via `confint`.

Usage

```
runmulti_glm(
  data,
  main_var,
  family = "poisson",
  outcome,
  covariates = NULL,
  ...
)
```

Arguments

data	A data.frame or data.table containing all variables.
main_var	A character vector of main variable names to test.
family	A GLM family. Accepted forms: • A character string naming a stats family function, e.g. "poisson", "Gamma", "gaussian", "quasipoisson", "quasibinomial", "inverse.gaussian". • A family function, e.g. poisson. • A family object, e.g. poisson(link = "sqrt").
outcome	A character string specifying the outcome column.
covariates	A character vector of covariate names. Default NULL.
...	Additional arguments passed to glm().

Value

A data.frame with columns: variable, contrast, family, link, beta, lower95, upper95, pvalue, and n. Multi-level categorical exposures return one row per non-reference contrast. For log- or logit-link families $\exp(\text{beta})$ gives the ratio-scale effect (IRR, rate ratio, etc.).

runmulti_lm

Run multiple linear regression models

Description

Fit linear regression models (lm) for each main variable separately. When covariates is NULL, univariate models are fitted. Otherwise, multivariate models adjusting for the specified covariates are fitted.

Usage

```
runmulti_lm(data, main_var, covariates = NULL, outcome, ...)
```

Arguments

<code>data</code>	A data.frame or data.table containing all variables.
<code>main_var</code>	A character vector of main variable names to test.
<code>covariates</code>	A character vector of covariate names to adjust for. Default NULL (univariate).
<code>outcome</code>	A character string specifying the outcome (dependent) variable name.
<code>...</code>	Additional arguments passed to <code>lm()</code> .

Value

A data.frame with columns: `variable`, `contrast`, `beta`, `lower95`, `upper95`, and `pvalue`. Multi-level categorical exposures return one row per non-reference contrast.

<code>runmulti_logit</code>	<i>Run multiple logistic regression models</i>
-----------------------------	--

Description

Fit logistic regression models (`glm` with `family = binomial`) for each main variable separately. When `covariates` is NULL, univariate models are fitted. Otherwise, multivariate models adjusting for the specified covariates are fitted.

Usage

```
runmulti_logit(data, main_var, covariates = NULL, outcome, ...)
```

Arguments

<code>data</code>	A data.frame or data.table containing all variables.
<code>main_var</code>	A character vector of main variable names to test.
<code>covariates</code>	A character vector of covariate names to adjust for. Default NULL (univariate).
<code>outcome</code>	A character string specifying the binary outcome (dependent) variable name (0/1).
<code>...</code>	Additional arguments passed to <code>glm()</code> .

Value

A data.frame with columns: `variable`, `contrast`, `OR`, `lower95`, `upper95`, and `pvalue`. Multi-level categorical exposures return one row per non-reference contrast.

runmulti_negbin	<i>Run multiple negative-binomial regression models</i>
-----------------	---

Description

Fit negative-binomial GLMs (`glm.nb`) for each main variable separately. This is the standard approach for overdispersed count outcomes where the Poisson variance assumption is violated.

The overdispersion parameter `theta` is estimated per model and reported alongside the effect estimate.

Usage

```
runmulti_negbin(data, main_var, outcome, covariates = NULL, ...)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> containing all variables.
<code>main_var</code>	A character vector of main variable names to test.
<code>outcome</code>	A character string specifying the count outcome column.
<code>covariates</code>	A character vector of covariate names. Default <code>NULL</code> .
<code>...</code>	Additional arguments passed to <code>glm.nb()</code> .

Value

A `data.frame` with columns: `variable`, `contrast`, `IRR`, `lower95`, `upper95`, `pvalue`, `theta`, and `n`. Multi-level categorical exposures return one row per non-reference contrast. `IRR` is the incidence rate ratio ($\exp(\beta)$). `theta` is the estimated negative-binomial dispersion parameter (larger values indicate less overdispersion).

runmulti_trend	<i>Run Grouped-Exposure Trend Tests</i>
----------------	---

Description

Run Grouped-Exposure Trend Tests

Usage

```
runmulti_trend(
  data,
  main_var,
  outcome = NULL,
  covariates = NULL,
  model_type = c("cox", "logistic", "linear"),
  endpoint = NULL,
  ref_level = NULL,
  score_method = c("integer", "median", "custom"),
  custom_scores = NULL,
  include_level_estimates = TRUE,
  ...
)
```

Arguments

<code>data</code>	A data.frame or data.table.
<code>main_var</code>	Character vector of grouped exposure variable names.
<code>outcome</code>	Outcome column for logistic or linear models.
<code>covariates</code>	Optional character vector of covariates.
<code>model_type</code>	One of "cox", "logistic", or "linear".
<code>endpoint</code>	Character vector of length 2 for Cox models.
<code>ref_level</code>	Optional reference level applied to every grouped exposure.
<code>score_method</code>	One of "integer", "median", or "custom".
<code>custom_scores</code>	Optional named list of custom score mappings.
<code>include_level_estimates</code>	Logical; if TRUE, include category-specific estimates.
<code>...</code>	Additional arguments passed to the fitted model.

Value

A data.frame containing grouped-effect estimates and a repeated `p_trend` column for each exposure.

score_protein_ppi_clusters

Score network clusters in a PPI graph

Description

A thin wrapper around `TCMDATA::add_cluster_score()`.

Usage

```
score_protein_ppi_clusters(ppi, cluster_attr = "louvain_cluster", min_size = 3)
```

Arguments

ppi	An igraph object, a list returned by <code>get_protein_ppi()</code> , or a list containing a graph element.
cluster_attr	Character. Vertex attribute containing cluster labels. Default is "louvain_cluster".
min_size	Integer. Minimum cluster size to retain. Default is 3.

Value

A data.frame containing cluster scores.

```
select_incident_by_years
```

Select Incident Cases by Time Since Enrollment

Description

Keep only participants with incident events and classify them as occurring within `n_years` or after `n_years` since enrollment. By default, the function uses `outcome_surv_time` and `outcome_status` generated by `build_survival_dataset()`. If the follow-up time column is not available, the function can compute it from enrollment and event dates.

Usage

```
select_incident_by_years(
  df,
  n_years = 5,
  time_col = "outcome_surv_time",
  status_col = "outcome_status",
  baseline_col = "p53_i0",
  event_date_col = "earliest_date",
  group_col = "incident_timing",
  output = c("combined", "split"),
  copy = TRUE,
  verbose = TRUE
)
```

Arguments

df	A data.frame or data.table.
n_years	Numeric scalar. Cutoff in years for classifying incident events. Default is 5.
time_col	Column name for follow-up time in years. Default is "outcome_surv_time".

status_col	Column name for event status where 1 indicates an incident event. Default is "outcome_status".
baseline_col	Column name for enrollment date. Used only when time_col is not present. Default is "p53_i0".
event_date_col	Column name for event date. Used only when time_col is not present or when status_col is unavailable. Default is "earliest_date".
group_col	Name of the output grouping column. Default is "incident_timing".
output	Output format. "combined" returns one filtered data object; "split" returns a named list with two filtered data objects for events within and after n_years.
copy	Logical scalar. If TRUE and df is a data.table, work on a copied object before filtering.
verbose	Logical scalar. If TRUE, print a short selection summary.

Value

If output = "combined", a filtered object with the same class as df, containing only participants with incident events. The output adds group_col. If time_col is missing but can be derived from dates, the function also adds time_col. If output = "split", a named list with within_n_years and after_n_years, each preserving the same class as df.

Examples

```
df <- data.frame(
  id = 1:5,
  outcome_surv_time = c(1.2, 4.9, 5.0, 8.1, 3.0),
  outcome_status = c(1, 1, 1, 1, 0)
)

result <- select_incident_by_years(df, n_years = 5)
table(result$incident_timing)

split_result <- select_incident_by_years(df, n_years = 5, output = "split")
names(split_result)
```

sensitivity_exclude_early_events

Exclude Early Events for Sensitivity Analysis

Description

Remove participants who experienced the event within the first n_years of follow-up. The returned dataset keeps the same columns and class as the input so it can be passed directly to the standard regression functions.

Usage

```
sensitivity_exclude_early_events(
  data,
  endpoint = c("outcome_surv_time", "outcome_status"),
  n_years,
  copy = TRUE,
  verbose = TRUE
)
```

Arguments

data	A data.frame or data.table.
endpoint	Character vector of length 2 giving the time and status columns, e.g. c("outcome_surv_time", "outcome_status").
n_years	Numeric scalar. Events with follow-up time less than or equal to this value will be excluded.
copy	Logical scalar. If TRUE and data is a data.table, work on a copied object before filtering.
verbose	Logical scalar. If TRUE, print a short filtering summary.

Value

An object with the same class and columns as data, with filtered rows removed. A `sensitivity_info` attribute is added for auditability.

Examples

```
dt_sens <- sensitivity_exclude_early_events(
  data = mtcars,
  endpoint = c("wt", "vs"),
  n_years = 3
)
```

sensitivity_exclude_missing_covariates

Exclude Rows with Missing Covariates for Sensitivity Analysis

Description

Remove participants with missing values in any of the specified covariates. The returned dataset keeps the same columns and class as the input so it can be passed directly to the standard regression functions.

Usage

```
sensitivity_exclude_missing_covariates(
  data,
  covariates,
  copy = TRUE,
  stepwise = FALSE,
  verbose = TRUE
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> .
<code>covariates</code>	Character vector of covariate names to check.
<code>copy</code>	Logical scalar. If <code>TRUE</code> and <code>data</code> is a <code>data.table</code> , work on a copied object before filtering.
<code>stepwise</code>	Logical scalar. If <code>TRUE</code> , apply covariate missingness filters sequentially in the order provided and record a row-level flow table in <code>attr(result, "complete_case_flow")</code> .
<code>verbose</code>	Logical scalar. If <code>TRUE</code> , print a short filtering summary.

Value

An object with the same class and columns as `data`, with filtered rows removed. A `sensitivity_info` attribute is added for auditability.

Examples

```
dt_sens <- sensitivity_exclude_missing_covariates(
  data = mtcars,
  covariates = c("hp", "wt")
)
```

<code>split_prs_weights</code>	<i>Split PRS Weights by Chromosome</i>
--------------------------------	--

Description

Write one score file per chromosome. Empty chromosomes are skipped by default, so `ukb_plan_prs()` does not launch jobs which cannot contribute to the score.

Usage

```
split_prs_weights(
  weights,
  output_dir,
  prefix = "prs_weights",
  chromosomes = NULL,
  drop_empty = TRUE,
  overwrite = FALSE
)
```

Arguments

weights	A ukb_prs_harmonized or ukb_prs_weight_matrix object.
output_dir	Directory for chromosome-specific score files.
prefix	File-name prefix.
chromosomes	Optional chromosome subset. Defaults to chromosomes present in weights.
drop_empty	Skip requested chromosomes without weights.
overwrite	Overwrite existing files.

Value

A ukb_prs_weight_files object containing named file paths and row counts.

subset_protein_ppi	<i>Filter a STRING PPI network via TCMDATA</i>
--------------------	--

Description

A thin wrapper around `TCMDATA::ppi_subset()` for STRING-derived PPI networks.

Usage

```
subset_protein_ppi(
  ppi,
  n = NULL,
  score_cutoff = 0.7,
  edge_attr = "score",
  rm_isolates = TRUE
)
```

Arguments

ppi	An igraph object, a list returned by <code>get_protein_ppi()</code> , or a list containing a graph element.
n	Integer. Number of top-degree nodes to keep. If <code>NULL</code> , no degree filtering is applied.
score_cutoff	Numeric. Minimum STRING confidence score to retain. Default is 0.7.
edge_attr	Character. Edge attribute containing the confidence score. Default is "score".
rm_isolates	Logical. Remove isolated nodes after filtering? Default is <code>TRUE</code> .

Value

An igraph object.

summarise_gp_coverage *Summarise UKB GP Registration Coverage*

Description

Summarises parsed GP registration intervals to one row per participant. Open-ended intervals are bounded only when `observation_end` is supplied. Filtered clinical records may be supplied as secondary evidence, but they produce "clinical_only" rather than confirmed registration coverage.

Usage

```
summarise_gp_coverage(
  registrations,
  clinical_records = NULL,
  participants = NULL,
  observation_end = NULL
)
```

Arguments

registrations	Output from <code>parse_gp_registrations()</code> .
clinical_records	Optional filtered <code>gp_clinical</code> records (raw or parsed). Only the <code>eid</code> column is used.
participants	Optional participant vector or <code>data.frame</code> with <code>eid</code> . Requested participants without registration/clinical evidence are retained with <code>coverage_status = "unknown"</code> .
observation_end	Optional study/data-release cut-off used to close missing deduction dates and cap later deduction dates.

Value

A participant-level `data.table` containing coverage bounds, days, registration counts, providers, and coverage status.

summarise_gp_diagnoses

Summarise GP Diagnoses to Participant Level

Description

Collapses matched, standardized GP records to one row per participant and disease. Only participants with at least one matching GP code are returned; absence from this result must not be interpreted as a confirmed control without separately evaluating GP registration and coverage.

Usage

```
summarise_gp_diagnoses(gp_records)
```

Arguments

gp_records Output from `parse_gp_clinical()` with a disease column.

Value

A data.table with GP case indicator, earliest usable GP event date, record counts, and source.

summary.mediation_result

Summary Method for Mediation Results

Description

Print a summary of mediation analysis results.

Usage

```
## S3 method for class 'mediation_result'
summary(object, exponentiate = FALSE, ...)
```

Arguments

object An object of class "mediation_result".

exponentiate Logical; whether to exponentiate estimates (for HR/OR). Default FALSE.

... Additional arguments (unused).

Value

Invisibly returns the object.

tidy.mi_pooled_result *Tidy Method for mi_pooled_result*

Description

Returns a tidy data frame of pooled estimates, compatible with broom package style.

Usage

```
tidy.mi_pooled_result(
  x,
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  ...
)
```

Arguments

<code>x</code>	An <code>mi_pooled_result</code> object.
<code>conf.int</code>	Logical; include confidence intervals? Default TRUE.
<code>conf.level</code>	Confidence level. Default 0.95.
<code>exponentiate</code>	Logical; exponentiate estimates? Default FALSE.
<code>...</code>	Additional arguments (ignored).

Value

A data frame with columns: `term`, `estimate`, `std.error`, `statistic`, `p.value`, and optionally `conf.low`, `conf.high`, `fmi`.

ukb_check_rap_env *Check the UK Biobank RAP execution environment*

Description

Inspect whether the current R session is running inside a UK Biobank Research Analysis Platform (RAP)-like environment and return reproducible diagnostics for RAP-aware workflows. The function only checks environment variables, local paths, and the availability of the `dx` command-line tool unless `check_auth = TRUE`; it does not read or export participant-level data.

Usage

```
ukb_check_rap_env(
  output_dir = NULL,
  require_rap = FALSE,
  require_dx = FALSE,
  check_auth = FALSE,
  check_write = FALSE,
  verbose = TRUE
)
```

Arguments

output_dir	Optional output directory to assess.
require_rap	Logical. If TRUE, mark the check as failed when the session does not appear to be running on RAP.
require_dx	Logical. If TRUE, mark the check as failed when the dx command-line tool is unavailable.
check_auth	Logical. If TRUE, call <code>dx whoami</code> and <code>dx env --bash</code> to check DNAnexus authentication and the active project context. This check does not inspect participant-level data.
check_write	Logical. If TRUE and <code>output_dir</code> is provided, test whether a small temporary file can be written and removed.
verbose	Logical. If TRUE, print a compact summary.

Value

A list with class `ukb_rap_env` containing RAP environment metadata and a check table.

Examples

```
env <- ukb_check_rap_env(verbose = FALSE)
```

ukb_clean_missing	<i>Clean UK Biobank Missing and Non-response Values</i>
-------------------	---

Description

Converts common UK Biobank non-response labels and numeric missing codes into analysis-ready missing values. Empty strings are always converted to NA. Informative non-response labels can either be converted to NA or retained as "Unknown" for modelling.

Usage

```
ukb_clean_missing(  
  data,  
  cols = NULL,  
  action = c("na", "unknown"),  
  extra_labels = NULL,  
  numeric_codes = c(-1, -3),  
  trim = TRUE,  
  in_place = FALSE,  
  verbose = TRUE  
)
```

Arguments

data	A data.frame or data.table.
cols	Optional character vector of columns to clean. If NULL, all columns are considered.
action	How to handle informative character labels: "na" converts them to NA; "unknown" converts them to "Unknown". Numeric missing codes are always converted to NA.
extra_labels	Additional character labels to treat as informative missing.
numeric_codes	Numeric values to treat as missing. Defaults to common UKB values -1 and -3.
trim	Logical. Trim leading/trailing whitespace in character columns.
in_place	Logical. If TRUE and data is a data.table, modify by reference. Default FALSE returns a cleaned copy.
verbose	Logical. Print a concise cleaning summary.

Value

A data.table.

ukb_compare_cox_results
<i>Compare Cox results between training and validation sets</i>

Description

Merge two Cox result tables by variable, summarize replication of training-set significant variables in validation, and compute log(HR) correlations.

Usage

```
ukb_compare_cox_results(  
  train_results,  
  validation_results,  
  variable_col = "variable",  
  hr_col = "HR",  
  p_col = "pvalue",  
  train_prefix = "train",  
  validation_prefix = "validation",  
  p_adjust_methods = c("BH", "bonferroni"),  
  alpha = 0.05  
)
```

Arguments

- train_results Cox result table for the training set.
- validation_results Cox result table for the validation set.
- variable_col Variable column name.
- hr_col Hazard-ratio column name.
- p_col Raw p-value column name.
- train_prefix Prefix for training-set columns in the comparison table.
- validation_prefix Prefix for validation-set columns.
- p_adjust_methods Multiple-testing correction methods to add when adjusted p-value columns are absent. Defaults to BH and Bonferroni.
- alpha Significance threshold.

Value

A list with train_results, validation_results, comparison, replication_summary, and correlation_summary.

ukb_compare_sensitivity_cox
<i>Compare sensitivity Cox results against a main analysis</i>

Description

Merge one or more sensitivity-analysis Cox result tables with a main Cox result table, then summarize concordance by sensitivity analysis.

Usage

```
ukb_compare_sensitivity_cox(  
  main_results,  
  sensitivity_results,  
  sensitivity_col = "sensitivity",  
  variable_col = "variable",  
  hr_col = "HR",  
  p_col = "pvalue",  
  main_prefix = "main",  
  sensitivity_prefix = "sensitivity",  
  p_adjust_methods = c("BH", "bonferroni"),  
  alpha = 0.05  
)
```

Arguments

- main_results Main Cox result table.
- sensitivity_results
 Sensitivity Cox result table containing one row per variable and sensitivity analysis.
- sensitivity_col
 Column identifying the sensitivity analysis.
- variable_col Variable column.
- hr_col Hazard-ratio column.
- p_col Raw p-value column.
- main_prefix Prefix for main-analysis columns.
- sensitivity_prefix
 Prefix for sensitivity-analysis columns.
- p_adjust_methods
 Multiple-testing correction methods to add if absent.
- alpha Significance threshold.

Value

A list with standardized result tables, comparison table, and correlation summary.

ukb_cox_diagnostics	<i>Diagnose Proportional Hazards Assumptions for a Cox Model</i>
---------------------	--

Description

Diagnose Proportional Hazards Assumptions for a Cox Model

Usage

```
ukb_cox_diagnostics(
  model,
  transform = c("km", "rank", "identity"),
  terms = TRUE,
  global = TRUE,
  alpha = 0.05,
  return_object = TRUE
)
```

Arguments

model	A fitted coxph() model.
transform	Character scalar passed to cox.zph().
terms	Logical; keep term-level rows.
global	Logical; keep the GLOBAL row.
alpha	Numeric threshold for flagging PH violations.
return_object	Logical; if TRUE, include the raw cox.zph object.

Value

A list containing a tidy diagnostics table, the global p-value, and optionally the raw cox.zph object.

```
ukb_create_extraction_manifest
      Create a RAP extraction manifest
```

Description

Build a compact manifest describing the UKB fields intended for RAP extraction. This is designed as an auditable planning object that can be stored with analysis scripts before running rap_plan_extract() or rap_extract_pheno().

Usage

```
ukb_create_extraction_manifest(
  field_id = NULL,
  variable_set = NULL,
  variables = NULL,
  dataset = NULL,
  entity = "participant",
  output = NULL,
  include_eid = TRUE,
  purpose = NULL,
  notes = NULL
)
```

Arguments

<code>field_id</code>	Optional numeric or character vector of UKB field IDs.
<code>variable_set</code>	Optional curated variable-set name from <code>get_variable_sets()</code> .
<code>variables</code>	Optional predefined variable names from <code>get_variable_info()</code> .
<code>dataset</code>	Optional RAP dataset name.
<code>entity</code>	RAP entity name, usually "participant".
<code>output</code>	Optional planned extraction output path.
<code>include_eid</code>	Logical. Whether participant ID is expected in the extraction.
<code>purpose</code>	Optional short description of the analysis purpose.
<code>notes</code>	Optional free-text notes.

Value

A list with class `ukb_extraction_manifest`.

Examples

```
manifest <- ukb_create_extraction_manifest(
  field_id = c(31, 21022),
  variable_set = "clinical_core",
  purpose = "demo"
)
```

ukb_decode	<i>Decode UK Biobank RAP exports</i>
------------	--------------------------------------

Description

Decode UK Biobank RAP exports

Usage

```
ukb_decode(
  data,
  metadata = NULL,
  decode_names = TRUE,
  decode_values = TRUE,
  keep_raw = TRUE,
  suffix = "_label",
  ...
)
```

Arguments

<code>data</code>	A data.frame or data.table.
<code>metadata</code>	Optional object from <code>ukb_metadata_setup()</code> .
<code>decode_names</code>	Logical. If TRUE, rename UKB columns using field titles.
<code>decode_values</code>	Logical. If TRUE, decode coded values where coding metadata are available.
<code>keep_raw</code>	Logical. If TRUE, decoded labels are added as new columns.
<code>suffix</code>	Suffix for decoded label columns when <code>keep_raw = TRUE</code> .
<code>...</code>	Arguments passed to <code>ukb_metadata_setup()</code> when <code>metadata</code> is NULL.

Value

A data.frame or data.table matching the input class.

ukb_decode_column_names

Decode UK Biobank column names

Description

Decode UK Biobank column names

Usage

```
ukb_decode_column_names(
  data,
  metadata = NULL,
  style = c("snake", "title", "field_id_title"),
  keep_instance = TRUE,
  keep_array = TRUE,
  max_nchar = 80,
  ...
)
```

Arguments

<code>data</code>	A data.frame or data.table.
<code>metadata</code>	Optional object from <code>ukb_metadata_setup()</code> .
<code>style</code>	Name style. "snake" converts field titles to snake_case, "title" uses the official title, and "field_id_title" prefixes the field ID.
<code>keep_instance</code>	Logical. Keep UKB instance suffixes such as <code>_i0</code> .
<code>keep_array</code>	Logical. Keep UKB array suffixes such as <code>_a0</code> .
<code>max_nchar</code>	Optional maximum column-name length.
<code>...</code>	Arguments passed to <code>ukb_metadata_setup()</code> when <code>metadata</code> is NULL.

Value

A data.frame or data.table matching the input class.

ukb_decode_values	<i>Decode UK Biobank coded values</i>
-------------------	---------------------------------------

Description

Decode UK Biobank coded values

Usage

```
ukb_decode_values(  
  data,  
  metadata = NULL,  
  keep_raw = TRUE,  
  suffix = "_label",  
  missing_to_na = TRUE,  
  ...  
)
```

Arguments

data	A data.frame or data.table.
metadata	Optional object from ukb_metadata_setup().
keep_raw	Logical. If TRUE, add label columns and keep raw columns.
suffix	Suffix for label columns when keep_raw = TRUE.
missing_to_na	Logical. If TRUE, UKB-style negative codes are set to NA in decoded label columns when no explicit label is available.
...	Arguments passed to ukb_metadata_setup() when metadata is NULL.

Value

A data.frame or data.table matching the input class.

ukb_demo	<i>Generate a small synthetic UK Biobank-style demo dataset</i>
----------	---

Description

Generates a small fully synthetic toy dataset for documentation and smoke-test workflows. The data are created at runtime from parametric and categorical toy distributions and are not stored in the package as participant-level records.

Usage

```
ukb_demo(n = NULL, seed = 20260618L)
```

Arguments

- n Optional number of rows to return. If NULL, 500 rows are returned.
- seed Integer random seed used to generate the toy data. The default provides reproducible examples. Use NULL to avoid setting a seed.

Value

A data.frame of synthetic cohort variables with missing values retained.

Examples

```
demo <- ukb_demo(100)
demo2 <- ukb_demo(100, seed = 1)
dim(demo)
names(demo)
```

ukb_dictionary_zh	<i>Chinese UK Biobank field-path dictionary</i>
-------------------	---

Description

A field-path dictionary used by [ukb_query_dictionary](#) to support Chinese-language lookup of UK Biobank variables. The table stores a six-level translated category hierarchy and variable label. It does not contain participant-level records and does not include official RAP data values. Official UKB field IDs and RAP column names should still be resolved against a project-specific RAP data dictionary generated inside RAP.

Usage

```
ukb_dictionary_zh
```

Format

A data frame with 34,953 rows and 6 translated hierarchy columns. The original UTF-8 column names are preserved in the data object and can be inspected with `names(ukb_dictionary_zh)` after loading the dataset.

Source

Curated UKBAnalytica Chinese field-path dictionary for metadata lookup. This dataset contains metadata labels only.

```
ukb_download_rap_dictionary
```

Download the official RAP data dictionary

Description

Runs `dx extract_dataset -ddd` inside UK Biobank RAP and returns the generated official data dictionary CSV path. This function checks that it is being executed in a RAP-like environment before calling `dx`.

Usage

```
ukb_download_rap_dictionary(
  dataset = NULL,
  output_dir = ".",
  delimiter = ",",
  timeout = 600,
  require_rap = TRUE
)
```

Arguments

<code>dataset</code>	RAP .dataset file or record identifier. If NULL, rap_find_dataset is used.
<code>output_dir</code>	Directory where the dictionary files should be written.
<code>delimiter</code>	Output delimiter passed to <code>dx extract_dataset</code> .
<code>timeout</code>	Timeout in seconds for the <code>dx</code> command.
<code>require_rap</code>	Logical. If TRUE, require a RAP-like environment.

Value

Path to the generated `*.data_dictionary.csv` file.

ukb_extract_fields	<i>Extract UK Biobank fields from a search result or field list</i>
--------------------	---

Description

Extract UK Biobank fields from a search result or field list

Usage

```
ukb_extract_fields(
  x = NULL,
  field_id = NULL,
  metadata = NULL,
  mode = c("plan", "sync", "job"),
  top_n = NULL,
  dataset = NULL,
  entity = "participant",
  ...
)
```

Arguments

<code>x</code>	Optional object returned by <code>ukb_search_fields()</code> or <code>ukb_field_info()</code> .
<code>field_id</code>	Optional UKB field IDs. Ignored when <code>x</code> supplies IDs.
<code>metadata</code>	Optional object from <code>ukb_metadata_setup()</code> .
<code>mode</code>	"plan" returns a RAP extraction plan, "sync" calls <code>rap_extract_pheno()</code> , and "job" calls <code>rap_submit_extract()</code> .
<code>top_n</code>	Optional number of top search-result fields to extract.
<code>dataset</code>	Optional RAP .dataset file name.
<code>entity</code>	RAP entity. Defaults to "participant".
<code>...</code>	Additional arguments passed to the selected RAP extraction function.

Value

A RAP extraction plan, a data.table, an output path, or a RAP job submission result depending on mode.

ukb_field_info	<i>Inspect one UK Biobank field</i>
----------------	-------------------------------------

Description

Inspect one UK Biobank field

Usage

```
ukb_field_info(  
  x,  
  by = c("auto", "field_id", "title", "rap_column", "variable"),  
  metadata = NULL,  
  live = FALSE,  
  ...  
)
```

Arguments

x	A UKB field ID, RAP column name, predefined UKBAnalytica variable name, or field title keyword.
by	Lookup mode. "auto" detects field IDs, RAP columns, predefined variable names, and otherwise falls back to title search.
metadata	Optional object from ukb_metadata_setup().
live	Logical. If TRUE and a single field ID is available, missing official metadata can be filled from the public UKB Showcase field page.
...	Arguments passed to ukb_metadata_setup() when metadata is NULL.

Value

An object of class ukb_field_info.

ukb_gwas_phewas_tools	<i>Curated GWAS and PheWAS Tool Set for UKB RAP</i>
-----------------------	---

Description

Return the tool set considered by UKBAnalytica for UK Biobank GWAS/PheWAS workflows. The table distinguishes the first supported path from optional alternatives. It is metadata only and does not install or execute software.

Usage

```
ukb_gwas_phewas_tools()
```

Value

A data.frame describing tool roles and support status.

ukb_gwas_qc_profile	<i>UKB Genetic Sample QC Profiles</i>
---------------------	---------------------------------------

Description

Create a declarative sample-QC profile for GWAS phenotype preparation. The default profile does not remove participants. The "rap_example" profile reproduces the sample filters used in the DNAnexus UKB RAP end-to-end GWAS/PheWAS tutorial: concordant reported/genetic sex, genetic ethnic grouping value 1, no reported sex-chromosome aneuploidy, and inclusion in the genetic principal-component calculation set.

Usage

```
ukb_gwas_qc_profile(  
  profile = c("none", "rap_example"),  
  ancestry_values = 1,  
  require_pca_sample = TRUE  
)
```

Arguments

- profile QC profile. "none" performs no built-in filtering; "rap_example" reproduces the official tutorial profile.
- ancestry_values Values accepted in UKB field 22006 when profile = "rap_example". Defaults to 1.
- require_pca_sample Logical. Require field 22020 to equal 1 in the tutorial profile.

Value

A list of class ukb_gwas_qc_profile.

ukb_metadata_setup	<i>Set up UK Biobank metadata for search, extraction, and decoding</i>
--------------------	--

Description

Builds a lightweight metadata object from any combination of RAP-approved fields, a UK Biobank data dictionary, and coding/encoding tables. This is the recommended first step before searching fields, inspecting field definitions, extracting phenotype columns, or decoding RAP exports.

Usage

```
ukb_metadata_setup(
  source = c("auto", "files", "rap"),
  data_dict = NULL,
  codings = NULL,
  fields_df = NULL,
  dataset = NULL,
  entity = "participant",
  cache = FALSE,
  cache_dir = NULL,
  refresh = FALSE,
  quiet = FALSE
)
```

Arguments

source	Metadata source strategy. "auto" uses any supplied files and tries RAP field discovery when available. "files" uses only supplied files and cached field listings. "rap" requires RAP field discovery.
data_dict	Optional UKB data dictionary file, such as a RAP data_dictionary.csv generated by <code>dx extract_dataset -ddd</code> , an older Data_Dictionary_Showcase.tsv, or an equivalent tabular export.
codings	Optional UKB coding/encoding table, such as an older Codings.tsv or an equivalent table with coding ID, value, and meaning columns.
fields_df	Optional cached output from <code>rap_list_fields()</code> .
dataset	Optional RAP .dataset file name.
entity	RAP dataset entity. Defaults to "participant".
cache	Logical. If TRUE, save the metadata object as an RDS file.
cache_dir	Optional cache directory. Defaults to <code>R_user_dir("UKBAnalytica", "cache")</code> .
refresh	Logical. Passed to <code>rap_list_fields()</code> when RAP discovery is used.
quiet	Logical. If FALSE, print short messages about unavailable optional metadata sources.

Value

An object of class `ukb_metadata`.

ukb_ml_as_split*Standardize Manual ML Train/Test Splits*

Description

Converts user-provided train/test (and optional validation) data frames into a standardized `ukb_ml_split` object. This object is consumed by the high level ML workflow and keeps the test set frozen until final evaluation.

Usage

```
ukb_ml_as_split(  
  train_data,  
  test_data,  
  validation_data = NULL,  
  id_col = NULL,  
  check_overlap = TRUE,  
  outcome = NULL,  
  outcome_type = c("auto", "binary", "multiclass", "continuous")  
)
```

Arguments

<code>train_data</code>	Training/development data.
<code>test_data</code>	Frozen test data.
<code>validation_data</code>	Optional validation data.
<code>id_col</code>	Optional participant ID column used to check overlap.
<code>check_overlap</code>	Logical. Check duplicated and overlapping IDs.
<code>outcome</code>	Outcome column name.
<code>outcome_type</code>	One of "auto", "binary", "multiclass", or "continuous".

Value

A `ukb_ml_split` object.

ukb_ml_calibration	<i>Calibration Curve Analysis</i>
--------------------	-----------------------------------

Description

Generate calibration curve to assess prediction reliability.

Usage

```
ukb_ml_calibration(  
  object,  
  newdata = NULL,  
  n_bins = 10,  
  method = c("none", "loess", "isotonic"),  
  plot = TRUE,  
  ...  
)
```

Arguments

object	A ukb_ml_workflow, ukb_ml_flow, ukb_ml_final, ukb_ml_test_eval, or legacy ukb_ml binary-classification object.
newdata	Optional evaluation data. Stored frozen-test predictions are used for workflow, flow, and test-evaluation objects when this is NULL. This argument is required for a ukb_ml_final object.
n_bins	Number of bins for calibration (default 10)
method	Smoothing method: "loess", "isotonic", or "none"
plot	Whether to create calibration plot
...	Additional arguments

Value

A ukb_ml_calibration object containing the grouped calibration table, Brier score, expected calibration error, prevalence, positive class, evaluation source, and optionally a ggplot object.

ukb_ml_compare	<i>Compare Multiple ML Models</i>
----------------	-----------------------------------

Description

Compare performance of multiple trained ML models.

Usage

```
ukb_ml_compare(
  ...,
  models = list(),
  metrics = NULL,
  test_data = NULL,
  plot = TRUE
)
```

Arguments

<code>...</code>	ukb_ml objects to compare
<code>models</code>	Alternative: list of ukb_ml objects
<code>metrics</code>	Metrics to compare
<code>test_data</code>	Optional common test data
<code>plot</code>	Whether to create comparison plot

Value

ukb_ml_compare object with comparison results

ukb_ml_compare_feature_sets

Compare Multiple Feature Sets with a Frozen-Test ML Workflow

Description

Runs the same machine-learning workflow across multiple feature sets using a shared `ukb_ml_split`. For binary outcomes, the function can tune models by cross-validation, learn a threshold on training-development predictions, refit the final model, evaluate the frozen test set, and return unified metrics, prediction, threshold, and ROC tables.

Usage

```
ukb_ml_compare_feature_sets(
  split,
  feature_sets,
  outcome = NULL,
  model = "xgboost",
  outcome_type = c("auto", "binary"),
  model_labels = NULL,
  param_grid = NULL,
  tune_params = list(),
  threshold_method = c("none", "fixed", "youden"),
  threshold_params = list(),
```

```

metrics = c("auc", "accuracy", "sensitivity", "specificity", "ppv", "npv", "f1",
  "brier"),
positive_class = NULL,
use_validation_in_refit = FALSE,
seed = NULL,
verbose = TRUE
)

```

Arguments

<code>split</code>	A <code>ukb_ml_split</code> object.
<code>feature_sets</code>	Named list of character vectors. Each vector contains the feature names used by one model.
<code>outcome</code>	Optional outcome column. Defaults to <code>split\$outcome</code> .
<code>model</code>	Model type passed to <code>ukb_ml_tune</code> and <code>ukb_ml_fit_final</code> .
<code>outcome_type</code>	Outcome type. Currently this helper is intended for binary classification.
<code>model_labels</code>	Optional labels for feature sets. Can be a named vector or a vector in the same order as <code>feature_sets</code> .
<code>param_grid</code>	Optional parameter grid. Can be a single grid shared by all models or a named list keyed by feature-set name.
<code>tune_params</code>	Additional arguments passed to <code>ukb_ml_tune</code> .
<code>threshold_method</code>	"none", "fixed", or "youden".
<code>threshold_params</code>	Additional arguments passed to <code>ukb_ml_threshold</code> .
<code>metrics</code>	Optional metric names passed to <code>ukb_ml_evaluate_test</code> .
<code>positive_class</code>	Optional positive class label for binary outcomes.
<code>use_validation_in_refit</code>	Logical passed to <code>ukb_ml_fit_final</code> .
<code>seed</code>	Optional random seed.
<code>verbose</code>	Logical.

Value

A `ukb_ml_feature_set_compare` object containing per-feature-set models and unified result tables.

ukb_ml_compare_flows *Compare Multiple Feature Sets and/or Models*

Description

Batch-runs [ukb_ml_flow](#) across feature-set and model combinations. The same frozen train/test split is reused for every combination, making the output suitable for comparing different feature groups, different machine-learning algorithms, or the full feature-set-by-model grid.

Usage

```
ukb_ml_compare_flows(
  formula = NULL,
  data = NULL,
  split = NULL,
  train_data = NULL,
  test_data = NULL,
  validation_data = NULL,
  id_col = NULL,
  outcome = NULL,
  feature_sets = NULL,
  features = NULL,
  models = "xgboost",
  compare = c("auto", "feature_sets", "models", "both"),
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
  feature_set_labels = NULL,
  model_labels = NULL,
  param_grid = NULL,
  tune = TRUE,
  tune_params = list(),
  threshold_params = list(),
  ...
)
```

Arguments

formula	Optional base formula. The response is used as the outcome. Predictors are used as the default feature set when <code>feature_sets</code> is <code>NULL</code> .
data, split, train_data, test_data, validation_data, id_col	Passed to ukb_ml_flow .
outcome	Optional outcome column. Required when <code>formula</code> is <code>NULL</code> .
feature_sets	Optional named list of feature vectors. If <code>NULL</code> , one feature set is derived from <code>formula</code> or <code>features</code> .
features	Optional feature names used when <code>formula</code> and <code>feature_sets</code> are <code>NULL</code> .
models	Character vector of models supported by ukb_ml_supported_models .

compare	Comparison mode: "auto", "feature_sets", "models", or "both". In "auto" mode, all supplied feature-set and model combinations are evaluated.
outcome_type	Outcome type passed to ukb_ml_flow .
feature_set_labels	Optional labels for feature sets.
model_labels	Optional labels for models.
param_grid	Optional hyperparameter grid. Can be a single grid shared by all combinations, a named list keyed by model, feature set, or "feature_set__model".
tune	Logical. Run hyperparameter tuning for each comparison flow.
tune_params	Optional list passed to ukb_ml_tune . Can also be keyed by model, feature set, or combination.
threshold_params	Optional list passed to ukb_ml_threshold . Can also be keyed by model, feature set, or combination.
...	Additional arguments passed to ukb_ml_flow , including outcome_type, split_params, threshold_method, metrics, positive_class, use_validation_in_refit, compute_shap, shap_params, seed, and verbose.

Value

A `ukb_ml_flow_compare` object containing flows, metrics, comparison, predictions, roc, and thresholds.

ukb_ml_confusion	<i>Confusion Matrix</i>
------------------	-------------------------

Description

Generate confusion matrix for classification model.

Usage

```
ukb_ml_confusion(object, newdata = NULL, threshold = 0.5, plot = TRUE, ...)
```

Arguments

object	A <code>ukb_ml</code> object
newdata	Optional new data
threshold	Classification threshold (default 0.5)
plot	Whether to create confusion matrix plot
...	Additional arguments

Value

`ukb_ml_confusion` object

Description

Perform k-fold cross-validation for ML models.

Usage

```
ukb_ml_cv(  
  formula,  
  data,  
  model = "rf",  
  task = "classification",  
  folds = 5,  
  repeats = 1,  
  stratify = TRUE,  
  metrics = NULL,  
  params = list(),  
  seed = NULL,  
  verbose = TRUE,  
  ...  
)
```

Arguments

formula	Model formula
data	Data frame
model	Model type
task	Task type
folds	Number of folds (default 5)
repeats	Number of repeats (default 1)
stratify	Use stratified folds for classification
metrics	Metrics to compute
params	Model parameters
seed	Random seed
verbose	Print progress
...	Additional arguments

Value

ukb_ml_cv object with cross-validation results

ukb_ml_dca	<i>Decision Curve Analysis</i>
------------	--------------------------------

Description

Compute Decision Curve Analysis (DCA) net benefit across a range of threshold probabilities for a binary classification model.

Usage

```
ukb_ml_dca(  
  object,  
  newdata = NULL,  
  plot = TRUE,  
  thresholds = seq(0.01, 0.99, by = 0.01),  
  harm = 0,  
  ...  
)
```

Arguments

object	A ukb_ml_workflow, ukb_ml_flow, ukb_ml_final, ukb_ml_test_eval, or legacy ukb_ml binary-classification object.
newdata	Optional evaluation data. Stored frozen-test predictions are used when available; this is required for a ukb_ml_final object.
plot	Whether to create the DCA plot (default TRUE)
thresholds	Numeric vector of threshold probabilities (default seq(0.01, 0.99, by = 0.01))
harm	Additional harm parameter subtracted from net benefit (default 0)
...	Additional arguments

Value

A ukb_ml_dca object containing threshold-specific net benefit for the model, treat-all and treat-none strategies, prevalence, positive class, evaluation source, harm, and optionally a ggplot object.

ukb_ml_evaluate_test	<i>Evaluate the Final Model Once on the Frozen Test Set</i>
----------------------	---

Description

Applies the final model, selected features, tuned hyperparameters, and fixed threshold to the frozen test set exactly once.

Usage

```
ukb_ml_evaluate_test(
  object,
  split,
  metrics = NULL,
  threshold = NULL,
  positive_class = NULL,
  verbose = TRUE
)
```

Arguments

object	A ukb_ml_final object.
split	A ukb_ml_split object.
metrics	Optional metric names to return.
threshold	Optional threshold override for binary classification.
positive_class	Optional positive class label.
verbose	Logical.

Value

A ukb_ml_test_eval object.

ukb_ml_feature_select *Select Features for UKB ML Workflows*

Description

Performs optional feature selection using only the training/development data in a ukb_ml_split. The test set is never used for feature selection.

Usage

```
ukb_ml_feature_select(
  split,
  formula,
  method = c("none", "boruta", "filter", "glmnet"),
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
  max_features = NULL,
  boruta_params = list(),
  keep_tentative = TRUE,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

split	A ukb_ml_split object.
formula	Model formula.
method	"none", "boruta", "filter", or "glmnet".
outcome_type	Outcome type. Defaults to the split outcome type.
max_features	Optional maximum number of selected features.
boruta_params	Parameters passed to Boruta: :Boruta().
keep_tentative	Logical. Keep Boruta tentative features.
seed	Optional random seed.
verbose	Logical.

Value

A ukb_ml_feature object.

ukb_ml_fit_final	<i>Refit the Final ML Model on Training Development Data</i>
------------------	--

Description

Fits the final model with selected features and tuned parameters using train or train plus validation data. The frozen test set is not used.

Usage

```
ukb_ml_fit_final(
  split,
  formula,
  model,
  best_params = list(),
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
  feature_spec = NULL,
  threshold = NULL,
  use_validation_in_refit = TRUE,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

split	A ukb_ml_split object.
formula	Model formula.
model	Model type.
best_params	Best hyperparameters.
outcome_type	Outcome type.
feature_spec	Optional ukb_ml_feature object.
threshold	Optional ukb_ml_threshold object.
use_validation_in_refit	Logical. If TRUE, refit on train + validation.
seed	Optional random seed.
verbose	Logical.
...	Additional arguments.

Value

A ukb_ml_final object.

ukb_ml_flow	<i>Run a Complete Single-Model UKB ML Flow</i>
-------------	--

Description

High-level single-model interface for common UK Biobank machine-learning analyses. The function can create or consume a frozen train/test split, tune model hyperparameters, learn a binary threshold, fit the final model, evaluate the frozen test set, prepare ROC data, and optionally compute SHAP values.

Usage

```
ukb_ml_flow(  
  formula = NULL,  
  data = NULL,  
  split = NULL,  
  train_data = NULL,  
  test_data = NULL,  
  validation_data = NULL,  
  id_col = NULL,  
  outcome = NULL,  
  features = NULL,  
  model = "xgboost",  
  model_id = "model",  
  model_label = NULL,  
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
```

```

split_params = list(),
param_grid = NULL,
tune = TRUE,
tune_params = list(),
best_params = NULL,
threshold_method = c("none", "fixed", "youden"),
threshold_params = list(),
metrics = NULL,
positive_class = NULL,
use_validation_in_refit = FALSE,
compute_shap = FALSE,
shap_data = NULL,
shap_params = list(),
seed = NULL,
verbose = TRUE
)

```

Arguments

formula	Model formula. Required unless both outcome and features are supplied.
data	Optional full dataset. Used to create a split when split is NULL and train_data/test_data are not supplied.
split	Optional <code>ukb_ml_split</code> object.
train_data, test_data, validation_data	Optional pre-split datasets used when split is NULL.
id_col	Optional participant ID column for overlap checks and output predictions.
outcome	Optional outcome column. Defaults to the response in formula or split\$outcome.
features	Optional feature names. Used when formula is NULL.
model	Model type passed to <code>ukb_ml_tune</code> and <code>ukb_ml_fit_final</code> .
model_id, model_label	Optional model identifier and display label.
outcome_type	Outcome type.
split_params	List passed to <code>ukb_ml_split_data</code> when splitting a full data object.
param_grid	Optional hyperparameter grid.
tune	Logical. Run hyperparameter tuning.
tune_params	Additional arguments passed to <code>ukb_ml_tune</code> .
best_params	Optional final model parameters when tune = FALSE.
threshold_method	"none", "fixed", or "youden".
threshold_params	Additional arguments passed to <code>ukb_ml_threshold</code> .
metrics	Optional metric names passed to <code>ukb_ml_evaluate_test</code> .
positive_class	Optional positive class label for binary outcomes.

use_validation_in_refit	Logical passed to ukb_ml_fit_final .
compute_shap	Logical. Compute SHAP values for the final model.
shap_data	Optional data used for SHAP. Defaults to the frozen test set.
shap_params	Additional arguments passed to ukb_shap .
seed	Optional random seed.
verbose	Logical.

Value

A `ukb_ml_flow` object with standardized components: `split`, `formula`, `features`, `tune`, `threshold`, `final_model`, `test_eval`, `metrics`, `predictions`, `roc`, and optional `shap`.

ukb_ml_gain_lift	<i>Gain and Lift Curve Analysis</i>
------------------	-------------------------------------

Description

Compute Gain and Lift curves for a binary classification model by ranking predictions into decile bins.

Usage

```
ukb_ml_gain_lift(object, newdata = NULL, plot = TRUE, n_bins = 10, ...)
```

Arguments

object	A <code>ukb_ml</code> object
newdata	Optional new data
plot	Whether to create gain and lift plots (default TRUE)
n_bins	Number of bins / deciles (default 10)
...	Additional arguments

Value

A `ukb_ml_gain_lift` object with field data containing: `decile`, `population_pct`, `positive_capture_pct`, `gain`, `lift`

ukb_ml_importance	<i>Get Variable Importance</i>
-------------------	--------------------------------

Description

Extract variable importance from a trained ML model.

Usage

```
ukb_ml_importance(object, type = NULL, ...)
```

Arguments

object	A ukb_ml object
type	Importance type (model-specific)
...	Additional arguments

Value

Data frame with variable importance scores

ukb_ml_ks	<i>KS Curve Analysis</i>
-----------	--------------------------

Description

Compute Kolmogorov-Smirnov curve (TPR - FPR vs threshold) for a binary classification model.

Usage

```
ukb_ml_ks(object, newdata = NULL, plot = TRUE, n_thresholds = 200, ...)
```

Arguments

object	A ukb_ml object
newdata	Optional new data for evaluation
plot	Whether to create the KS plot (default TRUE)
n_thresholds	Number of threshold points (default 200)
...	Additional arguments

Value

A ukb_ml_ks object with fields: data (threshold/tpr/fpr/ks), ks_stat (max KS), ks_threshold (threshold at max KS)

ukb_ml_metrics	<i>Calculate Model Performance Metrics</i>
----------------	--

Description

Compute performance metrics for a trained ML model.

Usage

```
ukb_ml_metrics(  
  object,  
  newdata = NULL,  
  metrics = NULL,  
  ci = FALSE,  
  ci_method = c("bootstrap", "delong"),  
  n_boot = 1000,  
  verbose = TRUE,  
  ...  
)
```

Arguments

object	A ukb_ml object
newdata	Optional new data for evaluation
metrics	Specific metrics to compute (NULL for defaults)
ci	Logical; compute confidence intervals (default FALSE)
ci_method	Method for CI: "bootstrap" or "delong" (for AUC)
n_boot	Number of bootstrap samples
verbose	Print results
...	Additional arguments

Value

Named vector or list with metrics and optional CIs

ukb_ml_model

*Train a Machine Learning Model***Description**

Unified interface for training machine learning models on UK Biobank data. Supports random forest, XGBoost, elastic net, SVM, and neural networks.

Usage

```
ukb_ml_model(
  formula,
  data,
  model = c("rf", "xgboost", "glmnet", "svm", "nnet", "logistic"),
  task = c("classification", "regression"),
  split_ratio = 0.8,
  stratify = TRUE,
  seed = NULL,
  sample_n = NULL,
  params = list(),
  cv = FALSE,
  cv_folds = 5,
  verbose = TRUE,
  ...
)
```

Arguments

formula	Model formula (e.g., outcome ~ var1 + var2)
data	Data frame containing variables
model	Model type: "rf", "xgboost", "glmnet", "svm", "nnet", "logistic"
task	Task type: "classification" or "regression"
split_ratio	Train/test split ratio (default 0.8)
stratify	Logical; use stratified sampling for classification (default TRUE)
seed	Random seed for reproducibility
sample_n	Optional; subsample data for large datasets
params	List of model-specific parameters
cv	Logical; perform cross-validation (default FALSE)
cv_folds	Number of CV folds (default 5)
verbose	Logical; print progress messages
...	Additional arguments passed to model function

Value

An object of class "ukb_ml" containing:

- model: The fitted model object
- model_type: Type of model used
- task: Task type (classification/regression)
- predictors: Names of predictor variables
- outcome: Name of outcome variable
- train_data: Training data
- test_data: Test data
- metrics: Model performance metrics

ukb_ml_nested_cv

Run Supplementary Nested Cross-Validation

Description

Estimates internal model performance with nested cross-validation using only the training portion of a `ukb_ml_split`. Each outer analysis fold runs feature selection and hyperparameter tuning without accessing its outer assessment fold. Validation and frozen test sets are never used. This function is intended as a supplementary robustness analysis; it does not replace final evaluation on an independent or frozen test set.

Usage

```
ukb_ml_nested_cv(
  split,
  formula,
  model,
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
  feature_select = c("none", "boruta", "filter", "glmnet"),
  feature_params = list(),
  tune = TRUE,
  tune_params = list(),
  outer_folds = 5,
  inner_folds = 5,
  repeats = 1,
  metric = NULL,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

<code>split</code>	A <code>ukb_ml_split</code> object. Only <code>split\$train</code> is used.
<code>formula</code>	Model formula.
<code>model</code>	Model type supported by ukb_ml_workflow .
<code>outcome_type</code>	Outcome type. Defaults to the split outcome type.
<code>feature_select</code>	Feature-selection method used within every inner and outer analysis fold.
<code>feature_params</code>	List passed to ukb_ml_feature_select .
<code>tune</code>	Logical. Tune hyperparameters within each outer analysis fold.
<code>tune_params</code>	Tuning specification matching ukb_ml_tune , including <code>search</code> , <code>param_grid</code> , <code>param_space</code> , <code>n_iter</code> , and <code>maximize</code> . Resampling is always inner-fold CV.
<code>outer_folds</code>	Number of outer assessment folds.
<code>inner_folds</code>	Number of inner tuning folds.
<code>repeats</code>	Number of repeated outer CV runs.
<code>metric</code>	Primary tuning metric. Defaults to AUC for binary outcomes, macro F1 for multiclass outcomes, and RMSE for continuous outcomes.
<code>seed</code>	Optional random seed.
<code>verbose</code>	Logical. Print outer-fold progress.

Value

A `ukb_ml_nested_cv` object containing outer-fold metrics, metric summaries, fold-specific selected features and tuning results, feature-selection frequencies, and outer-fold predictions. These estimates describe internal development-set performance and feature stability; they are not frozen-test estimates.

Examples

```
set.seed(42)
demo <- data.frame(
  outcome = factor(rep(c("control", "case"), each = 30)),
  age = rnorm(60),
  bmi = rnorm(60),
  marker = c(rnorm(30), rnorm(30, 0.8))
)
split <- ukb_ml_split_data(demo, outcome = "outcome", seed = 42,
                           verbose = FALSE)
nested <- ukb_ml_nested_cv(
  split,
  outcome ~ age + bmi + marker,
  model = "logistic",
  outer_folds = 3,
  inner_folds = 2,
  verbose = FALSE
)
nested$metric_summary
```

ukb_ml_pr	<i>Precision-Recall Curve Analysis</i>
-----------	--

Description

Compute Precision-Recall curve and area under PR curve (AUPRC) for a binary classification model.

Usage

```
ukb_ml_pr(object, newdata = NULL, plot = TRUE, n_thresholds = 200, ...)
```

Arguments

object	A ukb_ml_workflow, ukb_ml_flow, ukb_ml_final, ukb_ml_test_eval, or legacy ukb_ml binary-classification object.
newdata	Optional evaluation data. Stored frozen-test predictions are used when available; this is required for a ukb_ml_final object.
plot	Whether to create the PR plot (default TRUE)
n_thresholds	Maximum number of points retained for plotting. AUPRC is always calculated from all unique prediction thresholds.
...	Additional arguments

Value

A ukb_ml_pr object containing the precision-recall curve, average precision, prevalence, positive class, evaluation source, and optionally a ggplot object.

ukb_ml_predict	<i>Predict from ML Model</i>
----------------	------------------------------

Description

Generate predictions from a trained ukb_ml model.

Usage

```
ukb_ml_predict(  
  object,  
  newdata = NULL,  
  type = c("response", "prob", "class", "link"),  
  ...  
)
```

Arguments

object	A ukb_ml object from ukb_ml_model()
newdata	Optional new data for prediction. If NULL, uses test data.
type	Prediction type: "response", "prob", "class", "link"
...	Additional arguments

Value

Predictions as vector or matrix

ukb_ml_roc	<i>ROC Curve Analysis</i>
------------	---------------------------

Description

Generate ROC curve and calculate AUC with optional confidence intervals.

Usage

```
ukb_ml_roc(  
  object,  
  newdata = NULL,  
  plot = TRUE,  
  ci = TRUE,  
  ci_method = c("delong", "bootstrap"),  
  ...  
)
```

Arguments

object	A ukb_ml object or list of objects
newdata	Optional new data
plot	Whether to create ROC plot (default TRUE)
ci	Compute confidence interval for AUC
ci_method	Method: "delong" (default) or "bootstrap"
...	Additional arguments

Value

ukb_ml_roc object with ROC curve data

ukb_ml_roc_data*Create ROC Curve Data for Binary ML Predictions*

Description

Converts binary outcome predictions into a tidy ROC curve table with AUC and optional 95% confidence interval. This helper is useful for plotting one or more model ROC curves without re-running model evaluation.

Usage

```
ukb_ml_roc_data(  
  truth,  
  prob,  
  model_id = NULL,  
  model_label = NULL,  
  positive_class = NULL,  
  ci = TRUE,  
  ci_method = c("delong", "bootstrap"),  
  quiet = TRUE  
)
```

Arguments

truth	True binary outcome values.
prob	Predicted probability for the positive class.
model_id	Optional model identifier.
model_label	Optional model label used in plots.
positive_class	Optional positive class label. Defaults to the second level after converting truth to a factor.
ci	Logical. Calculate AUC 95% confidence interval.
ci_method	Method passed to <code>ci.auc()</code> ; usually "delong" or "bootstrap".
quiet	Logical passed to <code>roc()</code> .

Value

A data.frame with specificity, sensitivity, false-positive rate, threshold, AUC, and optional confidence interval columns.

ukb_ml_split_data *Split Data into Frozen ML Train/Test Sets*

Description

Creates a standardized `ukb_ml_split` object for the high-level ML workflow. Supports train/test and train/validation/test splits. The older `split_ratio/stratify_by = <column>` calling style is still accepted for compatibility.

Usage

```
ukb_ml_split_data(
  df,
  outcome = NULL,
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
  split = c("train_test", "train_valid_test"),
  train_ratio = 0.7,
  validation_ratio = 0.1,
  test_ratio = 0.2,
  split_ratio = NULL,
  stratify_by = c("auto", "outcome", "custom", "none"),
  stratify_col = NULL,
  regression_bins = 5,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

<code>df</code>	A data.frame or data.table.
<code>outcome</code>	Outcome column name. If NULL, a legacy random split is returned with <code>internal_validation</code> populated.
<code>outcome_type</code>	One of "auto", "binary", "multiclass", or "continuous".
<code>split</code>	Either "train_test" or "train_valid_test".
<code>train_ratio</code>	Training proportion.
<code>validation_ratio</code>	Validation proportion for train/validation/test.
<code>test_ratio</code>	Test proportion.
<code>split_ratio</code>	Deprecated compatibility alias for <code>train_ratio</code> .
<code>stratify_by</code>	"auto", "outcome", "custom", "none", or an older-style column name.
<code>stratify_col</code>	Column used when <code>stratify_by = "custom"</code> .
<code>regression_bins</code>	Number of quantile bins for continuous outcome stratification.
<code>seed</code>	Optional random seed.
<code>verbose</code>	Logical. Print split summary.

Value

A ukb_ml_split object.

ukb_ml_supported_models
<i>List Supported Machine Learning Models</i>

Description

Returns the machine-learning algorithms supported by the UKBAlytica ML workflow, including eligible outcome types, required R package, and default tuning parameters.

Usage

```
ukb_ml_supported_models(  
  outcome_type = c("all", "binary", "multiclass", "continuous")  
)
```

Arguments

outcome_type Optional outcome type filter: "all", "binary", "multiclass", or "continuous".

Value

A data.frame describing supported models.

Examples

```
ukb_ml_supported_models("binary")
```

ukb_ml_survival	<i>Train Survival Machine Learning Model</i>
-----------------	--

Description

Deprecated legacy interface for training machine learning models for survival analysis. New analyses should use [ukb_ml_survival_workflow](#), which freezes the test set before feature selection, tuning, final refit, and evaluation.

Usage

```
ukb_ml_survival(
  formula,
  data,
  model = c("rsf", "gbm_surv", "coxnet"),
  split_ratio = 0.8,
  seed = NULL,
  params = list(),
  verbose = TRUE,
  ...
)
```

Arguments

formula	Survival formula (e.g., <code>Surv(time, event) ~ x1 + x2</code>)
data	Data frame
model	Model type: "rsf" (random survival forest), "gbm_surv" (gradient boosting), "coxnet" (regularized Cox)
split_ratio	Train/test split ratio (default 0.8)
seed	Random seed
params	List of model-specific parameters
verbose	Print progress
...	Additional arguments

Value

A `ukb_ml_surv` object containing:

- `model`: Fitted survival model
- `c_index`: Harrell's C-index on test data
- `train_data`, `test_data`: Split datasets

`ukb_ml_survival_as_split`

Standardize Manual Survival ML Train/Test Splits

Description

Register user-provided survival train/test datasets as a frozen split object. This is the survival analogue of `ukb_ml_as_split`.

Usage

```
ukb_ml_survival_as_split(
  train_data,
  test_data,
  validation_data = NULL,
  time,
  event,
  id_col = NULL,
  check_overlap = TRUE
)
```

Arguments

<code>train_data</code>	Training/development data.
<code>test_data</code>	Frozen test data.
<code>validation_data</code>	Optional validation data.
<code>time</code>	Survival time column.
<code>event</code>	Event indicator column coded 0/1.
<code>id_col</code>	Optional participant ID column used to check overlap.
<code>check_overlap</code>	Logical. Check duplicated and overlapping IDs.

Value

A `ukb_ml_survival_split` object.

`ukb_ml_survival_evaluate_test`

Evaluate Survival ML Once on the Frozen Test Set

Description

Computes final survival ML metrics on the frozen test set. The primary metric is Harrell's C-index. Time-specific Brier scores use inverse probability of censoring weighting, with the censoring distribution estimated by Kaplan-Meier in the evaluation set.

Usage

```
ukb_ml_survival_evaluate_test(
  object,
  split,
  times = c(1, 3, 5, 10),
  verbose = TRUE,
  ...
)
```

Arguments

object	A ukb_ml_survival_final object.
split	A ukb_ml_survival_split object.
times	Time points for survival probability prediction.
verbose	Logical.
...	Additional arguments.

Value

A ukb_ml_survival_test_eval object.

ukb_ml_survival_feature_select

Select Features for Survival ML Workflows

Description

Performs optional feature selection using only training data. The test set is never used.

Usage

```
ukb_ml_survival_feature_select(  
  split,  
  formula,  
  method = c("none", "filter", "glmnet"),  
  max_features = NULL,  
  seed = NULL,  
  verbose = TRUE  
)
```

Arguments

split	A ukb_ml_survival_split object.
formula	Survival formula.
method	"none", "filter", or "glmnet".
max_features	Optional maximum number of selected features.
seed	Optional random seed.
verbose	Logical.

Value

A ukb_ml_survival_feature object.

`ukb_ml_survival_fit_final`*Refit Final Survival ML Model*

Description

Refits a survival ML model on training plus validation data when available, leaving the frozen test set untouched.

Usage

```
ukb_ml_survival_fit_final(  
  split,  
  formula,  
  model,  
  best_params = list(),  
  feature_spec = NULL,  
  seed = NULL,  
  verbose = TRUE,  
  ...  
)
```

Arguments

<code>split</code>	A <code>ukb_ml_survival_split</code> object.
<code>formula</code>	Survival formula.
<code>model</code>	Survival model type.
<code>best_params</code>	Model parameters.
<code>feature_spec</code>	Optional feature-selection result.
<code>seed</code>	Optional random seed.
<code>verbose</code>	Logical.
<code>...</code>	Additional arguments passed to the fitter.

Value

A `ukb_ml_survival_final` object.

ukb_ml_survival_importance
<i>Get Variable Importance for Survival Model</i>

Description

Get Variable Importance for Survival Model

Usage

```
ukb_ml_survival_importance(object, ...)
```

Arguments

object	A ukb_ml_surv object
...	Additional arguments

Value

Data frame with variable importance

ukb_ml_survival_predict
<i>Predict from Survival ML Model</i>

Description

Generate predictions from a survival ML model or survival ML workflow.

Usage

```
ukb_ml_survival_predict(  
  object,  
  newdata = NULL,  
  times = c(1, 3, 5, 10),  
  type = c("survival", "risk", "chf"),  
  ...  
)
```

Arguments

object	A ukb_ml_survival_workflow, ukb_ml_survival_final, or legacy ukb_ml_surv object.
newdata	Optional new data
times	Time points for survival prediction
type	Prediction type: "risk", "survival", "chf" (cumulative hazard)
...	Additional arguments

Value

Matrix of predictions (observations x time points)

ukb_ml_survival_shap	<i>SHAP Values for Survival Models</i>
----------------------	--

Description

Compute SHAP values for survival ML models at a specific time point.

Usage

```
ukb_ml_survival_shap(  
  object,  
  data = NULL,  
  time_point = 5,  
  nsim = 50,  
  sample_n = NULL,  
  seed = NULL,  
  verbose = TRUE,  
  method = c("auto", "kernelshap", "fastshap", "permutation", "kernalshap"),  
  background_data = NULL,  
  background_n = 200,  
  ...  
)
```

Arguments

object	A ukb_ml_surv object
data	Data for SHAP computation
time_point	Time point for SHAP calculation
nsim	Number of Monte Carlo samples
sample_n	Subsample size
seed	Random seed
verbose	Print progress
method	Model-agnostic SHAP backend. "permutation" is retained as an alias for "fastshap".
background_data	Optional reference data. Training data are used by default.
background_n	Maximum number of background observations.
...	Additional arguments

Value

A ukb_shap object

`ukb_ml_survival_split_data`*Split Data into Frozen Survival ML Train/Test Sets*

Description

Creates a frozen train/test or train/validation/test split for time-to-event machine learning. Event status is used for stratification by default.

Usage

```
ukb_ml_survival_split_data(  
  df,  
  time,  
  event,  
  split = c("train_test", "train_valid_test"),  
  train_ratio = 0.7,  
  validation_ratio = 0.1,  
  test_ratio = 0.2,  
  stratify_by = c("event", "custom", "none"),  
  stratify_col = NULL,  
  seed = NULL,  
  verbose = TRUE  
)
```

Arguments

<code>df</code>	A data.frame or data.table.
<code>time</code>	Survival time column.
<code>event</code>	Event indicator column coded 0/1.
<code>split</code>	Either "train_test" or "train_valid_test".
<code>train_ratio</code>	Training proportion.
<code>validation_ratio</code>	Validation proportion for train/validation/test.
<code>test_ratio</code>	Test proportion.
<code>stratify_by</code>	"event", "custom", "none", or a column name.
<code>stratify_col</code>	Column used when stratify_by = "custom".
<code>seed</code>	Optional random seed.
<code>verbose</code>	Logical. Print split summary.

Value

A `ukb_ml_survival_split` object.

ukb_ml_survival_tune *Tune Survival ML Hyperparameters Without Touching the Test Set*

Description

Tunes survival ML models using validation data or cross-validation inside the training set. The frozen test set is never used.

Usage

```
ukb_ml_survival_tune(
  split,
  formula,
  model,
  search = c("grid", "random"),
  param_grid = NULL,
  param_space = NULL,
  n_iter = NULL,
  resampling = c("cv", "validation"),
  folds = 5,
  metric = "c_index",
  maximize = TRUE,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

split	A ukb_ml_survival_split object.
formula	Survival formula.
model	"cox", "rsf", "gbm_surv", or "coxnet".
search	"grid" or "random".
param_grid	List or data.frame of candidate parameters.
param_space	Parameter space for random search.
n_iter	Number of random-search iterations.
resampling	"cv" or "validation".
folds	Number of CV folds.
metric	Currently "c_index".
maximize	Logical. Whether higher metric values are better.
seed	Optional random seed.
verbose	Logical.
...	Reserved for future extensions.

Value

A `ukb_ml_survival_tune` object.

`ukb_ml_survival_workflow`

Run a Frozen-Test Survival ML Workflow

Description

High-level survival ML workflow for time-to-event prediction. The test set is frozen before feature selection, hyperparameter tuning, final refit, and final evaluation.

Usage

```
ukb_ml_survival_workflow(
  formula,
  data = NULL,
  split = NULL,
  model = c("cox", "rsf", "gbm_surv", "coxnet"),
  split_params = list(),
  feature_select = c("none", "filter", "glmnet"),
  feature_params = list(),
  tune = TRUE,
  tune_params = list(),
  evaluation_times = c(1, 3, 5, 10),
  fit_final = TRUE,
  evaluate_test = TRUE,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

<code>formula</code>	Survival formula, for example <code>Surv(time, event) ~ x1 + x2</code> .
<code>data</code>	Optional full dataset. Required when <code>split</code> is <code>NULL</code> .
<code>split</code>	Optional <code>ukb_ml_survival_split</code> object.
<code>model</code>	"cox", "rsf", "gbm_surv", or "coxnet".
<code>split_params</code>	List passed to ukb_ml_survival_split_data .
<code>feature_select</code>	"none", "filter", or "glmnet".
<code>feature_params</code>	List passed to ukb_ml_survival_feature_select .
<code>tune</code>	Logical. Run hyperparameter tuning.
<code>tune_params</code>	List passed to ukb_ml_survival_tune .

evaluation_times	Time points for survival probability prediction.
fit_final	Logical. Refit final model.
evaluate_test	Logical. Evaluate once on frozen test set.
seed	Optional random seed.
verbose	Logical.
...	Additional arguments.

Value

A ukb_ml_survival_workflow object.

ukb_ml_threshold	<i>Learn a Binary Classification Threshold</i>
------------------	--

Description

Selects a binary classification threshold using a fixed value or Youden index on training-development predictions. The test set should never be supplied to this function.

Usage

```
ukb_ml_threshold(
  truth,
  prob,
  method = c("fixed", "youden"),
  fixed_threshold = 0.5,
  positive_class = NULL
)
```

Arguments

truth	True binary outcome values.
prob	Predicted probability for the positive class.
method	"fixed" or "youden".
fixed_threshold	Threshold used when method = "fixed".
positive_class	Optional positive class label.

Value

A ukb_ml_threshold object.

ukb_ml_tune

*Tune ML Hyperparameters Without Touching the Test Set***Description**

Searches model hyperparameters using only the training/development portion of a `ukb_ml_split`. The frozen test set is never used.

Usage

```
ukb_ml_tune(
  split,
  formula,
  model,
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
  search = c("grid", "random", "bayes"),
  param_grid = NULL,
  param_space = NULL,
  n_iter = NULL,
  resampling = c("cv", "validation"),
  folds = 5,
  metric = NULL,
  maximize = NULL,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

<code>split</code>	A <code>ukb_ml_split</code> object.
<code>formula</code>	Model formula.
<code>model</code>	Model type.
<code>outcome_type</code>	Outcome type.
<code>search</code>	"grid", "random", or "bayes". Bayesian search currently requires <code>rBayesianOptimization</code> ; if unavailable it falls back to random search with the same parameter space.
<code>param_grid</code>	List or <code>data.frame</code> of candidate parameters.
<code>param_space</code>	Parameter space for random or Bayesian search.
<code>n_iter</code>	Number of random/Bayesian iterations.
<code>resampling</code>	"cv" or "validation".
<code>folds</code>	Number of CV folds.
<code>metric</code>	Metric to optimize.
<code>maximize</code>	Logical. Whether higher metric values are better.

seed	Optional random seed.
verbose	Logical.
...	Reserved for future extensions.

Value

A ukb_ml_tune object.

ukb_ml_workflow	<i>Run a Frozen-Test UKB ML Workflow</i>
-----------------	--

Description

High-level, publication-oriented ML workflow for binary, multiclass, and continuous outcomes. The test set is frozen before feature selection, hyperparameter tuning, threshold learning, and final refit.

Usage

```
ukb_ml_workflow(
  formula,
  data = NULL,
  split = NULL,
  model,
  outcome_type = c("auto", "binary", "multiclass", "continuous"),
  split_params = list(),
  feature_select = c("none", "boruta", "filter", "glmnet"),
  feature_params = list(),
  tune = TRUE,
  tune_params = list(),
  threshold_method = c("none", "fixed", "youden"),
  threshold_params = list(),
  fit_final = TRUE,
  evaluate_test = TRUE,
  seed = NULL,
  verbose = TRUE,
  ...,
  nested_cv = FALSE,
  nested_cv_params = list()
)
```

Arguments

formula	Model formula.
data	Optional full dataset. Required when split is NULL.
split	Optional ukb_ml_split object.

model	Model type: "logistic", "linear", "rf", "xgboost", "glmnet", "svm", "nnet", "rpart", or "naive_bayes".
outcome_type	"auto", "binary", "multiclass", or "continuous".
split_params	List passed to ukb_ml_split_data when split is NULL.
feature_select	"none", "boruta", "filter", or "glmnet".
feature_params	List passed to ukb_ml_feature_select .
tune	Logical. Run hyperparameter tuning.
tune_params	List passed to ukb_ml_tune .
threshold_method	"none", "fixed", or "youden".
threshold_params	List passed to ukb_ml_threshold .
fit_final	Logical. Refit final model.
evaluate_test	Logical. Evaluate once on frozen test set.
seed	Optional random seed.
verbose	Logical.
...	Additional arguments.
nested_cv	Logical. Run supplementary nested cross-validation on the training split. Defaults to FALSE and does not alter the standard workflow when disabled.
nested_cv_params	Optional list containing outer_folds, inner_folds, repeats, metric, seed, or verbose for ukb_ml_nested_cv .

Value

A `ukb_ml_workflow` object containing the frozen data split, optional `nested_cv_result`, feature-selection and tuning results, threshold information, the refitted final model, and final test metrics and predictions.

ukb_participant_flow	<i>Build a participant flow table</i>
----------------------	---------------------------------------

Description

Apply sequential inclusion or exclusion rules and record the number of participants retained and removed at each step. Rules can be supplied as one-sided formulas, functions, logical vectors, or character vectors of variables requiring complete-case data.

Usage

```
ukb_participant_flow(  
  data,  
  steps,  
  id_col = NULL,  
  outcome_col = NULL,  
  event_value = 1,  
  start_label = "Initial population"  
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> .
<code>steps</code>	A named list of rules. Each rule can be: a one-sided formula such as <code>~ !is.na(age)</code> , a function returning a logical vector, a logical vector, or a character vector of variable names to retain complete cases.
<code>id_col</code>	Optional participant identifier column. If supplied, duplicate non-missing IDs are reported as an error.
<code>outcome_col</code>	Optional 0/1 outcome column used to count events after each step.
<code>event_value</code>	Value in <code>outcome_col</code> indicating an event. Defaults to 1.
<code>start_label</code>	Label for the first row.

Value

A `data.frame` with class `ukb_participant_flow`. The kept row index is stored in `attr(result, "kept_index")`.

Examples

```
dat <- data.frame(  
  eid = 1:5,  
  age = c(50, 60, NA, 55, 70),  
  status = c(0, 1, 0, 1, 0)  
)  
flow <- ukb_participant_flow(  
  dat,  
  steps = list("Complete age" = "age"),  
  id_col = "eid",  
  outcome_col = "status"  
)
```

ukb_plan_plink2	<i>Plan an Arbitrary PLINK2 Command</i>
-----------------	---

Description

Create an auditable, non-executing PLINK2 command plan from raw command-line arguments. This is the escape hatch for PLINK2 operations that do not have a dedicated UKBAnalytica wrapper. Arguments are passed as individual tokens and quoted before execution.

Dedicated wrappers such as [convert_gwas_datatype](#) and [ukb_plan_plink2_dosage](#) remain preferable for their validated inputs and output metadata.

Usage

```
ukb_plan_plink2(args, expected_outputs = character(), label = "plink2")
```

Arguments

args	Character vector containing PLINK2 command-line argument tokens in execution order.
expected_outputs	Optional character vector of expected output paths. This is metadata only; PLINK2 remains responsible for creating them.
label	Command label used in execution results.

Value

A list of class `ukb_plink2_plan`.

ukb_plan_plink2_dosage	<i>Plan PLINK2 Dosage Extraction for PheWAS</i>
------------------------	---

Description

Build auditable PLINK2 commands that extract selected variants from one or more UKB BGEN files and export additive dosages. The function only creates a plan; it does not execute PLINK2 or submit a RAP job.

Usage

```
ukb_plan_plink2_dosage(  
  bgen,  
  sample,  
  variant_file,  
  output_prefix,  
  keep_file = NULL,  
  ref_first = TRUE,  
  plink2_args = character()  
)
```

Arguments

bgen	One or more BGEN file paths.
sample	One sample file path or one path per BGEN.
variant_file	One variant inclusion file or one path per BGEN.
output_prefix	Output prefix in RAP-controlled storage.
keep_file	Optional two-column FID/IID sample inclusion file.
ref_first	Pass the PLINK2 BGEN ref-first modifier.
plink2_args	Additional PLINK2 argument tokens. Input, output, extraction, and sample-selection flags managed by this wrapper are rejected; use ukb_plan_plink2 for fully custom commands.

Value

A list of class `ukb_plink2_dosage_plan`.

ukb_plan_prs	<i>Plan PRS Calculation with PLINK2</i>
--------------	---

Description

Create one auditable PLINK2 `--score` command per genotype input. The plan requests score sums so chromosome-level results can be added without averaging artefacts. It does not execute PLINK2 or submit a DNAnexus job.

Usage

```
ukb_plan_prs(  
  weights,  
  bgen = NULL,  
  sample = NULL,  
  output_prefix,  
  keep_file = NULL,  
  variant_col = 1L,
```

```

effect_allele_col = 2L,
weight_col = 3L,
header = TRUE,
ref_first = TRUE,
no_mean_imputation = FALSE,
frequency_file = NULL,
error_on_freq_calc = !is.null(frequency_file),
list_variants = TRUE,
score_name = "PRS",
plink2_args = character(),
genotype = NULL,
genotype_format = c("auto", "bgen", "pgen", "bed")
)

```

Arguments

weights	Normalized output from <code>prepare_prs_weights()</code> , a combined matrix from <code>combine_prs_weights()</code> , chromosome files from <code>split_prs_weights()</code> , or a PLINK2 score-file path.
bgen	One or more BGEN paths. Retained for backward compatibility; new code can use genotype with <code>genotype_format</code> .
sample	One sample path or one path per BGEN. Must be NULL for PGEN and BED inputs, whose sample information is part of the file set.
output_prefix	Output prefix.
keep_file	Optional PLINK two-column sample inclusion file.
variant_col, effect_allele_col, weight_col	One-based columns in a raw score file. Normalized weights always use columns 1, 2, and 3.
header	Whether the score file has a header.
ref_first	Add the PLINK2 BGEN ref-first modifier.
no_mean_imputation	Add the PLINK2 modifier of the same name.
frequency_file	Optional PLINK2 allele-frequency file passed through <code>--read-freq</code> to make mean imputation independent of the scoring cohort.
error_on_freq_calc	Add <code>--error-on-freq-calc</code> . By default this is enabled when <code>frequency_file</code> is supplied.
list_variants	Write the variants actually used by PLINK2.
score_name	Name used for a single-score R output. Combined weight matrices retain their own score names.
plink2_args	Additional non-conflicting PLINK2 argument tokens.
genotype	One or more BGEN paths, PGEN prefixes/paths, or BED prefixes/paths.
genotype_format	Input format: "auto", "bgen", "pgen", or "bed".

Value

A `ukb_prs_plan` inheriting from `ukb_plink2_plan`.

<code>ukb_plan_regenie</code>	<i>Plan a REGENIE GWAS</i>
-------------------------------	----------------------------

Description

Build auditable REGENIE Step 1 and Step 2 command arguments from files produced by [ukb_write_gwas_phewas_phenotype](#). This function does not submit a RAP job and does not execute REGENIE.

The generated commands are intended for the REGENIE binary available in the RAP Swiss Army Knife app or another controlled RAP execution environment. For the managed RAP REGENIE orchestration app, use the same phenotype, covariate, sample, and variant-list files through that app's versioned input interface.

Usage

```
ukb_plan_regenie(
  files,
  bed_prefix,
  step1_extract,
  step2_bgen,
  step2_sample,
  output_prefix,
  phenotypes = NULL,
  covariate_file = NULL,
  covariates = NULL,
  categorical_covariates = NULL,
  trait_type = NULL,
  keep_file = NULL,
  step2_extract = NULL,
  prediction_file = NULL,
  ref_first = TRUE,
  firth = TRUE,
  spa = FALSE,
  p_threshold = 0.01,
  bsize_step1 = 1000L,
  bsize_step2 = 400L,
  threads = NULL,
  lowmem = TRUE,
  step1_args = character(),
  step2_args = character()
)
```

Arguments

files	A <code>ukb_gwas_phewas_files</code> object or path to a REGENIE-compatible phenotype file.
bed_prefix	Step 1 PLINK BED/BIM/FAM prefix, without extension.
step1_extract	Step 1 variant inclusion file.
step2_bgen	One or more Step 2 BGEN paths.
step2_sample	One sample path or one path per BGEN.
output_prefix	Output prefix in RAP-controlled storage.
phenotypes	Phenotype columns. Automatically obtained when <code>files</code> is a <code>UKBAnalyticaFiles</code> object.
covariate_file	Optional covariate file. Defaults to the generated REGENIE-compatible covariate file.
covariates	Covariate columns.
categorical_covariates	Subset of covariates treated as categorical.
trait_type	"binary" or "quantitative". Automatically obtained when all selected traits have the same type.
keep_file	Optional two-column FID/IID keep file. Defaults to the generated sample file.
step2_extract	Optional Step 2 variant inclusion file. Supply one path or one path per BGEN.
prediction_file	Step 1 prediction-list path expected by Step 2. If NULL, uses REGENIE's conventional <code><output_prefix>_step1_pred.list</code> path.
ref_first	Add <code>--ref-first</code> to Step 2.
firth	Use Firth fallback for binary traits.
spa	Use SPA instead of Firth fallback for binary traits. Only one of <code>firth</code> and <code>spa</code> may be true.
p_threshold	P-value threshold for approximate Firth/SPA fallback.
bsize_step1, bsize_step2	REGENIE block sizes.
threads	Optional positive integer thread count.
lowmem	Use REGENIE Step 1 low-memory mode.
step1_args, step2_args	Additional REGENIE argument tokens appended to the corresponding step. Flags managed by this wrapper are rejected; use ukb_plan_regenie_command for a fully custom command.

Value

A list of class `ukb_regenie_plan`.

ukb_plan_regenie_command
<i>Plan an Arbitrary REGENIE Command</i>

Description

Create an auditable, non-executing REGENIE command plan from raw command-line arguments. Arguments are preserved as individual tokens, so this escape hatch can represent official REGENIE options that are not exposed by the structured `ukb_plan_regenie` wrapper.

UKBAlytica validates token boundaries, but it does not attempt to validate every option or option combination. The installed REGENIE version remains authoritative.

Usage

```
ukb_plan_regenie_command(  
  args,  
  expected_outputs = character(),  
  label = "regenie"  
)
```

Arguments

args	Character vector containing REGENIE command-line argument tokens in execution order.
expected_outputs	Optional character vector of expected output paths. This is metadata only; REGENIE remains responsible for creating them.
label	Command label used in execution results.

Value

A list of class `ukb_regenie_command_plan` that also inherits from `ukb_regenie_plan`.

ukb_protein_annotation
<i>Annotate Olink-style protein variables</i>

Description

Annotate Olink-style protein variables

Usage

```
ukb_protein_annotation(  
  variables,  
  protein_prefix = "^olink_instance_0[.]",  
  drop_unmapped = FALSE  
)
```

Arguments

- variables Protein variable names.
- protein_prefix Regular expression prefix removed from variables.
- drop_unmapped Passed to `protein_to_gene_symbol()`.

Value

A data.frame with variable, protein_clean, gene_symbol, and mapping_source.

ukb_prs_catalog	<i>UK Biobank PRS Release V2 Catalogue</i>
-----------------	--

Description

Return the participant-level polygenic risk score fields in UK Biobank Showcase categories 301 (Standard) and 302 (Enhanced). Standard scores use external GWAS training data. Enhanced scores also use within-UKB training data and are available only in the PRS testing subgroup (field 26200).

Usage

```
ukb_prs_catalog(type = c("all", "standard", "enhanced"), trait = NULL)
```

Arguments

- type Score set to return: "standard", "enhanced", or "all".
- trait Optional trait name or abbreviation, for example "hypertension", "HT", or "T2D".

Value

A data.frame containing score type, trait, abbreviation, field ID, and testing-subgroup requirement.

ukb_query_dictionary *Query UK Biobank dictionary metadata*

Description

Searches UK Biobank variable metadata using a RAP-generated official data dictionary and the UKBAnalytica Chinese dictionary. Chinese queries are first matched against the built-in Chinese dictionary and translated into English candidate terms before matching the official dictionary. English queries, field IDs, and RAP/UKB column names are searched directly in the official dictionary.

This function is intended for RAP use. By default it requires a RAP-like environment; set `require_rap = FALSE` only for package development or tests using simulated dictionaries.

Usage

```
ukb_query_dictionary(
  query,
  official_dict = NULL,
  zh_dict = NULL,
  dataset = NULL,
  output_dir = tempdir(),
  language = c("auto", "zh", "en", "field_id", "column"),
  translation_map = NULL,
  max_results = 20,
  min_score = 0.35,
  require_rap = TRUE,
  timeout = 600
)
```

Arguments

<code>query</code>	Character vector of query terms, Chinese variable names, English names, UKB field IDs, or UKB/RAP column names.
<code>official_dict</code>	Optional official RAP data dictionary CSV. If NULL, ukb_download_rap_dictionary is called.
<code>zh_dict</code>	Optional Chinese dictionary CSV. Defaults to the UKBAnalytica built-in Chinese dictionary.
<code>dataset</code>	RAP .dataset file used when <code>official_dict</code> is NULL.
<code>output_dir</code>	Directory used when downloading the official dictionary.
<code>language</code>	Query language. "auto" detects Chinese, field IDs, and column names.
<code>translation_map</code>	Optional data.frame with columns <code>zh</code> and <code>en</code> , or a named character vector mapping Chinese terms to English query terms.
<code>max_results</code>	Maximum official dictionary matches returned per query.
<code>min_score</code>	Minimum internal matching score for official dictionary matches.
<code>require_rap</code>	Logical. Require a RAP-like environment before querying.
<code>timeout</code>	Timeout in seconds when downloading the official dictionary.

Value

A list of class `ukb_dictionary_query` with official matches, Chinese matches, query metadata, and source paths.

ukb_run_phewas	<i>Run PheWAS Using the PheWAS R Package</i>
----------------	--

Description

Convert the long ICD table in a `ukb_gwas_phewas_phenotype` object to phecodes and run the external PheWAS R package. The dependency is optional and is never installed automatically.

Usage

```
ukb_run_phewas(  
  x,  
  genotypes,  
  genotype_id_col = "IID",  
  genotype_cols = NULL,  
  covariate_cols = x$settings$covariates,  
  min_code_count = 1L,  
  min_records = 20L,  
  cores = 1L,  
  additive_genotypes = TRUE,  
  add_phecode_exclusions = TRUE,  
  sex_restrictions = TRUE,  
  vocabulary_map = c("icd10", "default"),  
  significance_threshold = c("p-value", "bonferroni", "fdr"),  
  alpha = 0.05  
)
```

Arguments

x	A <code>ukb_gwas_phewas_phenotype</code> .
genotypes	Data frame containing participant IDs and one or more additive genotype/dosage columns.
genotype_id_col	Participant ID column in genotypes.
genotype_cols	Genotype columns. Defaults to every non-ID column.
covariate_cols	Covariates from x used in every model.
min_code_count	Minimum code count used by PheWAS: <code>::createPhenotypes</code> .
min_records	Minimum case/control records used by PheWAS: <code>::phewas</code> .
cores	Number of R worker processes.
additive_genotypes	Treat genotype inputs as additive dosages.

add_phecode_exclusions	Apply phecode control exclusions.
sex_restrictions	Apply sex-specific phenotype restrictions when a sex table was prepared in x.
vocabulary_map	PheWAS vocabulary map: "icd10" matches the official RAP tutorial, while "default" supports the package's combined map.
significance_threshold	Threshold columns requested from PheWAS.
alpha	Family-wise/FDR alpha.

Value

A list of class `ukb_phewas_result`.

<code>ukb_run_plink2</code>	<i>Run a Planned PLINK2 Dosage Extraction</i>
-----------------------------	---

Description

Execute commands from [ukb_plan_plink2_dosage](#). Execution is opt-in, requires a PLINK2 binary, and does not itself submit a DNAnexus job.

Usage

```
ukb_run_plink2(
  plan,
  execute = FALSE,
  executable = "plink2",
  require_rap = TRUE,
  timeout = 86400
)
```

Arguments

plan	A plan inheriting from <code>ukb_plink2_plan</code> , including objects returned by ukb_plan_plink2 , ukb_plan_plink2_dosage , and convert_gwas_datatype .
execute	Logical. Defaults to FALSE.
executable	PLINK2 executable path or command name.
require_rap	Require a RAP-like environment before execution.
timeout	Per-command timeout in seconds.

Value

The unchanged plan when `execute = FALSE`; otherwise a list of command statuses and captured logs.

ukb_run_prs	<i>Run a Planned PRS Calculation</i>
-------------	--------------------------------------

Description

Run a Planned PRS Calculation

Usage

```
ukb_run_prs(  
  plan,  
  execute = FALSE,  
  executable = "plink2",  
  require_rap = TRUE,  
  timeout = 86400,  
  read_scores = TRUE,  
  standardize = FALSE,  
  overwrite = FALSE,  
  workers = 1L,  
  resume = FALSE  
)
```

Arguments

plan	A ukb_prs_plan.
execute	Execute PLINK2. Defaults to FALSE.
executable	PLINK2 command or path.
require_rap	Require a RAP-like environment before execution.
timeout	Per-command timeout in seconds.
read_scores	Read and combine successful .sscore outputs.
standardize	Add a combined PRS z-score.
overwrite	Permit PLINK2 to replace existing expected outputs.
workers	Number of chromosome commands to execute concurrently.
resume	Reuse complete existing chromosome outputs and execute only unfinished commands.

Value

The unchanged plan in dry-run mode; otherwise a ukb_prs_run containing PLINK2 logs and combined scores.

ukb_run_regenie	<i>Run a Planned REGENIE Analysis</i>
-----------------	---------------------------------------

Description

Execute direct REGENIE commands from [ukb_plan_regenie](#) or [ukb_plan_regenie_command](#). Execution is opt-in and requires a REGENIE binary. On RAP this is intended for an existing Swiss Army Knife or controlled custom execution environment; it does not itself submit a DNAnexus job.

Usage

```
ukb_run_regenie(  
  plan,  
  execute = FALSE,  
  steps = c("step1", "step2"),  
  executable = "regenie",  
  require_rap = TRUE,  
  timeout = 86400  
)
```

Arguments

plan	A plan inheriting from <code>ukb_regenie_plan</code> .
execute	Logical. Defaults to FALSE; when false, the plan is returned without running anything.
steps	Commands to run. For a structured plan, use "step1", "step2", or both. For a raw command plan, use its command label. NULL runs every command in the plan.
executable	REGENIE executable path or command name.
require_rap	Require a RAP-like environment before execution.
timeout	Per-command timeout in seconds.

Value

The unchanged plan when `execute = FALSE`; otherwise a list of command statuses and captured logs.

ukb_scale_with_parameters	<i>Standardize variables using existing scaling parameters</i>
---------------------------	--

Description

Apply previously estimated centering and scaling parameters to a data set. The parameter table can use either the native output from `ukb_standardize_by_train()` (variable, center, scale) or the legacy long format used by early case-study scripts (protein, statistic, value).

Usage

```
ukb_scale_with_parameters(data, parameters, variables = NULL)
```

Arguments

data	Data frame to transform.
parameters	Scaling parameter table.
variables	Optional variables to transform. Defaults to all variables available in the parameter table.

Value

A data.table with standardized variables.

ukb_search_fields	<i>Search UK Biobank fields</i>
-------------------	---------------------------------

Description

Search UK Biobank fields

Usage

```
ukb_search_fields(  
  query = NULL,  
  field_id = NULL,  
  metadata = NULL,  
  max_results = 50,  
  search_in = c("title", "description", "category", "field_name", "rap_field_names",  
    "coding_id"),  
  ...  
)
```

Arguments

query	Optional keyword matched against field title, description, category, coding ID, and RAP column names.
field_id	Optional UKB field IDs for exact lookup.
metadata	Optional object from <code>ukb_metadata_setup()</code> .
max_results	Maximum number of rows to return.
search_in	Columns to search.
...	Arguments passed to <code>ukb_metadata_setup()</code> when metadata is NULL.

Value

A data.frame of class `ukb_search_result`.

`ukb_sensitivity_suite` *Run a Cox sensitivity-analysis suite*

Description

Fit a primary Cox model and common sensitivity models using the same endpoint, exposure, and covariate structure. The suite currently supports complete-case filtering, exclusion of early events, and additional covariate adjustment sets.

Usage

```
ukb_sensitivity_suite(
  data,
  exposure,
  covariates = NULL,
  endpoint = c("outcome_surv_time", "outcome_status"),
  early_event_years = c(2, 4, 6),
  complete_case_covariates = NULL,
  additional_covariate_sets = NULL,
  conf_level = 0.95,
  verbose = TRUE
)
```

Arguments

data	A data.frame or data.table.
exposure	Character vector of exposure variables.
covariates	Optional character vector of primary adjustment covariates.
endpoint	Character vector of length 2 giving survival time and status.
early_event_years	Optional numeric vector of lag periods used to exclude events occurring at or before each cut point.

complete_case_covariates	Optional covariates for a complete-case sensitivity dataset.
additional_covariate_sets	Optional named list of extra covariate vectors. Each set is added to the primary covariates and refitted.
conf_level	Confidence level for hazard-ratio intervals.
verbose	Logical. If TRUE, print a compact summary.

Value

A list with class `ukb_sensitivity_suite` containing model objects, flow metadata, and a tidy summary table.

Examples

```
set.seed(1)
dat <- data.frame(
  time = rexp(100, 0.1),
  status = rbinom(100, 1, 0.3),
  exposure = rnorm(100),
  age = rnorm(100, 60, 5),
  sex = rbinom(100, 1, 0.5)
)
res <- ukb_sensitivity_suite(
  dat,
  exposure = "exposure",
  covariates = c("age", "sex"),
  endpoint = c("time", "status"),
  early_event_years = 1,
  verbose = FALSE
)
```

ukb_shap	<i>Compute SHAP Values</i>
----------	----------------------------

Description

Calculate SHAP values through native TreeSHAP, KernelSHAP, or FastSHAP while preserving one output structure across supported machine-learning models.

Usage

```
ukb_shap(
  object,
  data = NULL,
  nsim = 100,
  sample_n = NULL,
  seed = NULL,
```

```
    verbose = TRUE,
    class_level = NULL,
    method = c("auto", "treeshap", "kernelshap", "fastshap", "xgboost", "permutation",
               "kernalshap"),
    background_data = NULL,
    background_n = 200,
    ...
)
```

Arguments

object	A ukb_ml_workflow, ukb_ml_final, or legacy ukb_ml object.
data	Data to explain. A workflow uses its frozen test set when this is NULL. A final-model object requires explicit data.
nsim	Number of Monte Carlo samples for FastSHAP. Ignored by TreeSHAP and KernelSHAP.
sample_n	Optional number of observations to explain.
seed	Optional random seed.
verbose	Print progress messages.
class_level	Class probability to explain for multiclass models.
method	SHAP backend. "auto" uses TreeSHAP for XGBoost and an installed model-agnostic backend otherwise. Legacy values "xgboost" and "permutation" are aliases for "treeshap" and "fastshap". "kernalshap" is accepted as an alias for "kernelshap".
background_data	Optional reference data for KernelSHAP or FastSHAP. A workflow uses its training split by default. A final model uses data when no background is supplied.
background_n	Maximum number of background observations. Use NULL to retain all supplied background observations.
...	Additional arguments passed to the selected backend.

Value

A ukb_shap object containing the SHAP matrix, baseline, feature names and values, backend name, output scale, background sample size, and a local-accuracy diagnostic where available.

ukb_shap_dependence	<i>SHAP Dependence Values</i>
---------------------	-------------------------------

Description

Get SHAP dependence data for a specific feature.

Usage

```
ukb_shap_dependence(object, feature, color_feature = NULL, ...)
```

Arguments

- object A ukb_shap object
- feature Feature name to analyze
- color_feature Optional feature for coloring (interaction analysis)
- ... Additional arguments

Value

Data frame with feature values and SHAP values

ukb_shap_force	<i>SHAP Force Plot Data</i>
----------------	-----------------------------

Description

Get SHAP contribution data for a single observation (force plot).

Usage

```
ukb_shap_force(object, row_id = 1, max_features = 10, ...)
```

Arguments

- object A ukb_shap object
- row_id Row index to explain
- max_features Maximum features to show
- ... Additional arguments

Value

Data frame with feature contributions for the observation

ukb_shap_summary	<i>SHAP Summary Statistics</i>
------------------	--------------------------------

Description

Calculate summary statistics from SHAP values.

Usage

```
ukb_shap_summary(object, n = 20, ...)
```

Arguments

object	A ukb_shap object
n	Number of top features to show (default 20)
...	Additional arguments

Value

Data frame with feature importance based on SHAP

ukb_snapshot	<i>Record or Retrieve UKB Cohort Snapshots</i>
--------------	--

Description

Records lightweight cohort checkpoints during an analysis pipeline. Each snapshot stores row count, column count, number of columns containing missing values, complete row count, object size, and deltas from the previous snapshot. Calling `ukb_snapshot()` without data returns the current snapshot history.

Usage

```
ukb_snapshot(  
  data = NULL,  
  label = NULL,  
  id = "default",  
  reset = FALSE,  
  verbose = TRUE  
)
```

Arguments

<code>data</code>	Optional data.frame or data.table. If supplied, records a new snapshot.
<code>label</code>	Snapshot label. Required when recording a new snapshot.
<code>id</code>	Snapshot stream identifier. Use separate IDs for independent pipelines in the same R session.
<code>reset</code>	Logical. If TRUE, clears the snapshot history for id.
<code>verbose</code>	Logical. Print a concise snapshot summary.

Value

A data.table snapshot history.

ukb_standardize_by_train

Standardize variables using training-set parameters

Description

Standardize a set of variables in the training data and optionally apply the same centering and scaling parameters to a validation data set. This is useful for omics analyses where all downstream association estimates should be expressed per one training-set standard deviation.

Usage

```
ukb_standardize_by_train(
  train_data,
  validation_data = NULL,
  variables,
  center = TRUE,
  scale = TRUE
)
```

Arguments

<code>train_data</code>	Training data.
<code>validation_data</code>	Optional validation data.
<code>variables</code>	Character vector of variables to standardize.
<code>center</code>	Logical. If TRUE, subtract the training-set mean.
<code>scale</code>	Logical. If TRUE, divide by the training-set standard deviation.

Value

A list with train, validation, and parameters.

Examples

```
dat <- data.frame(x = 1:5, y = c(2, 3, 5, 7, 11))
ukb_standardize_by_train(dat, variables = c("x", "y"))$parameters
```

ukb_time_skeleton	<i>Build a UK Biobank follow-up time skeleton</i>
-------------------	---

Description

Creates a reusable participant-level time skeleton for prospective UK Biobank analyses. The function standardizes baseline date, approximate birth date, age at baseline, death date, loss-to-follow-up date, administrative censoring date, follow-up end date, and follow-up time. It does not define disease outcomes; instead, it provides a common time basis that can be reused by endpoint-specific functions such as [build_survival_dataset](#).

Usage

```
ukb_time_skeleton(
  data,
  id_col = "eid",
  baseline_col = "p53_i0",
  birth_year_col = "p34",
  birth_month_col = "p52",
  age_col = "p21022",
  death_date_cols = "^((participant\\.\\.?)p40000_i[0-9])+$",
  lost_to_followup_col = "p191",
  admin_censor_date = as.Date("2023-10-31"),
  keep_source_dates = TRUE
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> containing UK Biobank columns.
<code>id_col</code>	Participant identifier column. Default "eid".
<code>baseline_col</code>	Baseline assessment date column. Default "p53_i0".
<code>birth_year_col</code>	Year-of-birth column. Default "p34".
<code>birth_month_col</code>	Month-of-birth column. Default "p52".
<code>age_col</code>	Age-at-baseline column. Default "p21022". If missing, age is approximated from baseline date and birth year/month when available.
<code>death_date_cols</code>	Death date columns or a regular expression used to identify them. Default " <code>^((participant\\.\\.?)p40000_i[0-9])+\$</code> ".
<code>lost_to_followup_col</code>	Optional date lost to follow-up column. Default "p191".

admin_censor_date
Administrative censoring date.

keep_source_dates
Logical. If FALSE, source dates used to define censoring are removed from the output.

Value

A data.table with one row per participant and standardized follow-up time fields.

Examples

```
demo <- data.frame(
  eid = 1:3,
  p53_i0 = as.Date(c("2010-01-01", "2011-01-01", "2012-01-01")),
  p21022 = c(50, 60, 70),
  p40000_i0 = as.Date(c(NA, "2015-01-01", NA))
)

ukb_time_skeleton(demo, admin_censor_date = as.Date("2020-12-31"))
```

ukb_top_hr_results	Select top Cox associations by hazard ratio
--------------------	---

Description

Select top Cox associations by hazard ratio

Usage

```
ukb_top_hr_results(
  results,
  n_each_direction = 10,
  p_col = "p_bonferroni",
  alpha = 0.05,
  hr_col = "HR",
  label_cols = c("gene_symbol", "protein_clean", "variable"),
  dataset = NULL
)
```

Arguments

results Cox result table.

n_each_direction Number of HR > 1 and HR < 1 rows to keep.

p_col Adjusted p-value column used for filtering.

alpha Significance threshold.

hr_col	Hazard-ratio column.
label_cols	Candidate label columns.
dataset	Optional dataset label added to output.

Value

A data.frame.

ukb_train_validation_cox

Run Cox models in training and validation sets

Description

Fit the same multivariable Cox model series in a training set and validation set, optionally standardizing the main variables using training-set parameters, then summarize replication and log(HR) concordance.

Usage

```
ukb_train_validation_cox(
  train_data,
  validation_data,
  main_vars,
  covariates,
  endpoint,
  standardize_main_vars = TRUE,
  add_protein_annotation = FALSE,
  protein_prefix = "^olink_instance_0[.]",
  train_label = "train",
  validation_label = "validation",
  comparison_train_prefix = "train",
  comparison_validation_prefix = "valid",
  p_adjust_methods = c("BH", "bonferroni"),
  alpha = 0.05,
  ...
)
```

Arguments

train_data	Training data.
validation_data	Validation data.
main_vars	Main variables to evaluate.
covariates	Adjustment covariates.
endpoint	Two-column endpoint passed to <code>runmulti_cox()</code> .

<code>standardize_main_vars</code>	Logical. If TRUE, standardize <code>main_vars</code> using training-set means and SDs.
<code>add_protein_annotation</code>	Logical. If TRUE, add parsed protein names and gene symbols for Olink-style protein columns.
<code>protein_prefix</code>	Regular expression prefix removed from protein columns.
<code>train_label</code>	Training-set label.
<code>validation_label</code>	Validation-set label.
<code>comparison_train_prefix</code>	Prefix for training columns in the comparison table.
<code>comparison_validation_prefix</code>	Prefix for validation columns in the comparison table.
<code>p_adjust_methods</code>	P-value adjustment methods.
<code>alpha</code>	Significance threshold.
<code>...</code>	Additional arguments passed to <code>runmulti_cox()</code> .

Value

A list containing scaled data, scaling parameters, Cox results, and comparison summaries.

<code>ukb_validate_columns</code>	<i>Validate requested columns against a data object</i>
-----------------------------------	---

Description

Checks whether requested UKB/RAP columns are present in a `data.frame` or a character vector of available column names. The function can optionally treat `participant.p31` and `p31` as equivalent.

Usage

```
ukb_validate_columns(data, columns, ignore_entity_prefix = TRUE, error = FALSE)
```

Arguments

<code>data</code>	A <code>data.frame</code> / <code>data.table</code> or a character vector of available column names.
<code>columns</code>	Character vector of requested column names.
<code>ignore_entity_prefix</code>	Logical. If TRUE, compare both original names and names with a leading "participant." prefix removed.
<code>error</code>	Logical. If TRUE, stop when any requested column is missing.

Value

A `data.frame` of class `ukb_column_validation`.

Examples

```
dat <- data.frame(eid = 1:3, p31 = c(0, 1, 0))
ukb_validate_columns(dat, c("eid", "p31", "p21022"))
```

ukb_write_extraction_manifest

Write a RAP extraction manifest

Description

Write a RAP extraction manifest

Usage

```
ukb_write_extraction_manifest(manifest, path, format = c("csv", "rds"))
```

Arguments

manifest	A ukb_extraction_manifest object.
path	Output path.
format	Output format: "csv" writes the field table and a sidecar summary CSV, while "rds" writes the full manifest object.

Value

The output path, invisibly.

Examples

```
manifest <- ukb_create_extraction_manifest(field_id = c(31, 21022))
tmp <- tempfile(fileext = ".csv")
ukb_write_extraction_manifest(manifest, tmp)
```

ukb_write_gwas_phewas_phenotype

Write a GWAS/PheWAS Phenotype Object

Description

Materialize phenotype inputs in an explicit output directory. By default the function can only write from a RAP-like session to a path under /mnt/project. This keeps participant-level analysis files inside RAP-controlled storage.

Usage

```
ukb_write_gwas_phewas_phenotype(  
  x,  
  output_dir,  
  prefix = "ukba_gwas_phewas",  
  require_rap = TRUE,  
  overwrite = FALSE  
)
```

Arguments

x	Object returned by build_gwas_phewas_phenotype .
output_dir	Existing or new local directory. With require_rap = TRUE, it must be under /mnt/project.
prefix	File prefix.
require_rap	Require a RAP-like environment and RAP-scoped path.
overwrite	Overwrite existing files.

Value

A list of class `ukb_gwas_phewas_files` containing generated file paths and a non-participant-level manifest.

UKBAnalytica	<i>UKBAnalytica: UK Biobank Data Processing and Survival Analysis Toolkit</i>
--------------	---

Description

A high-performance R package for processing UK Biobank (UKB) Research Analysis Platform (RAP) data exports. Designed for epidemiological studies requiring efficient extraction of diagnosis records and generation of survival analysis datasets.

Details

Core Capabilities:

- Parse ICD-10/ICD-9 diagnosis codes from mixed-format data
- Parse OPCS4 operative procedure codes from hospital summary operations
- Process self-reported illness data with fractional year conversion
- Integrate death registry data as diagnosis sources
- Generate Cox regression-ready survival datasets
- Support flexible data source selection for sensitivity analyses

Key Functions:

- [parse_icd10_diagnoses](#): Extract ICD-10 hospital diagnoses
- [parse_icd9_diagnoses](#): Extract ICD-9 hospital diagnoses
- [parse_opcs4_procedures](#): Extract OPCS4 hospital procedures
- [parse_self_reported_illnesses](#): Extract self-reported conditions
- [parse_death_records](#): Extract death registry data
- [build_survival_dataset](#): Generate survival analysis data
- [extract_cases_by_source](#): Flexible source-specific extraction

UKB Data Fields:

- ICD-10: p41270 (codes) + p41280_a* (dates)
- ICD-9: p41271 (codes) + p41281_a* (dates)
- OPCS4: p41272 (codes) + p41282_a* (dates)
- Self-report: p20002_i*_a* (codes) + p20008_i*_a* (years)
- Death: p40001/p40002 (causes) + p40000 (dates)

Author(s)

Maintainer: Nan He <hinna01@163.com> ([ORCID](#))

References

UK Biobank Data Showcase: <https://biobank.ndph.ox.ac.uk/showcase/>

See Also

Useful links:

- <https://github.com/Hinna0818/UKBAlytica>
- <https://hinna0818.github.io/UKBAlytica/>
- Report bugs at <https://github.com/Hinna0818/UKBAlytica/issues>

Index

* datasets

- ukb_dictionary_zh, [147](#)
- assess_balance, [7](#)
- assess_gp_observability, [8](#)
- assess_gp_observability(), [46](#), [105](#)
- build_gwas_phewas_phenotype, [9](#), [214](#)
- build_survival_dataset, [11](#), [29](#), [209](#), [215](#)
- build_survival_dataset(), [131](#)
- calculate_air_pollution, [13](#)
- calculate_blood_pressure, [14](#)
- calculate_diet_score, [14](#)
- calculate_weights, [15](#)
- classify_metabolites, [16](#)
- coef.mediation_result, [16](#)
- combine_disease_definitions, [17](#)
- combine_prs_weights, [17](#)
- combine_prs_weights(), [192](#)
- compare_data_sources, [18](#)
- compute_protein_ppi_metrics, [19](#)
- confint.mediation_result, [19](#)
- convert_gwas_datatype, [20](#), [190](#), [199](#)
- create_baseline_table, [21](#)
- create_disease_definition, [11](#), [22](#)
- create_imputation_list, [24](#)
- create_medication_definition, [25](#)
- estimate_propensity_score, [26](#)
- extract_cases_by_source, [26](#), [30](#), [215](#)
- extract_diabetes_subtype_baseline, [28](#)
- extract_disease_diagnosis, [29](#)
- extract_disease_history, [30](#)
- extract_disease_history_sensitivity, [31](#)
- extract_medications, [32](#)
- extract_self_report_medications, [32](#)
- fit_mi_models, [34](#)
- get_death_dates, [34](#)
- get_disease_catalog, [35](#)
- get_field_info, [35](#)
- get_field_metadata, [36](#)
- get_medication_catalog, [37](#)
- get_pomegranate_diseases, [38](#)
- get_pomegranate_source_manifest, [39](#)
- get_predefined_diseases, [28–30](#), [39](#)
- get_predefined_diseases(), [35](#), [41](#)
- get_predefined_medications, [41](#)
- get_protein_ppi, [41](#)
- get_ukb_demo_colnames, [42](#)
- get_variable_info, [43](#)
- get_variable_info(), [44](#), [144](#)
- get_variable_set, [43](#)
- get_variable_sets, [44](#)
- get_variable_sets(), [144](#)
- harmonize_prs_weights, [45](#)
- harmonize_prs_weights(), [93](#)
- integrate_gp_results, [46](#)
- load_pomegranate_portal_coding, [47](#)
- load_ukb_medication_coding, [47](#)
- load_ukb_metabolite_panel, [48](#)
- load_ukb_prs, [48](#)
- match_propensity, [49](#)
- metabolite_to_metaboanalyst_name, [50](#)
- p.adjust(), [109](#)
- parse_cancer_registry, [51](#)
- parse_death_records, [52](#), [215](#)
- parse_gp_clinical, [53](#)
- parse_gp_clinical(), [137](#)
- parse_gp_query, [53](#)
- parse_gp_query(), [100](#), [103](#), [104](#)
- parse_gp_registrations, [54](#)
- parse_gp_registrations(), [8](#), [136](#)
- parse_icd10_diagnoses, [10](#), [55](#), [215](#)

parse_icd9_diagnoses, 56, 215
parse_opcs4_procedures, 56, 215
parse_self_reported_illnesses, 57, 215
plot.ukb_ml_flow, 58
plot.ukb_ml_flow_compare, 59
plot.ukb_rcs(plot_rcs), 81
plot_balance, 59
plot_calibration, 60
plot_correlation, 61
plot_cox_loghr_correlation, 62
plot_cox_sensitivity_correlation, 62
plot_enrichment_lollipop, 63
plot_forest, 64
plot_go_ora_bar, 65
plot_heatmap, 66
plot_km_curve, 67
plot_mediation, 68
plot_mediation_forest, 69
plot_metabolite_ora_barplot, 70
plot_metabolite_ora_dotplot, 70
plot_mi_diagnostics, 71
plot_mi_pooled, 72
plot_ml_calibration, 73
plot_ml_compare, 73
plot_ml_confusion, 74
plot_ml_dca, 75
plot_ml_gain, 75
plot_ml_importance, 76
plot_ml_ks, 76
plot_ml_lift, 77
plot_ml_pr, 77
plot_ml_roc, 78
plot_ml_roc_compare, 59, 78
plot_participant_flow, 79
plot_ps_distribution, 80
plot_rcs, 81
plot_regression_volcano, 82
plot_scatter, 83
plot_shap_beeswarm, 84
plot_shap_dependence, 86
plot_shap_force, 86
plot_shap_summary, 87
plot_stacked_bar, 88
plot_top_hrBars, 89
plot_violin, 89
pool_custom_estimates, 90
pool_mi_models, 91
prepare_prs_weights, 92
prepare_prs_weights(), 45, 192
preprocess_baseline, 93
preprocess_baseline(), 44
print.mediation_result, 94
protein_to_gene_symbol, 95
protein_to_gene_symbol(), 42, 113, 114, 117, 196
rank_protein_ppi_nodes, 96
rap_extract_pheno, 97
rap_extract_pheno(), 48, 49
rap_find_dataset, 98, 148
rap_list_fields, 98
rap_plan_extract, 99
rap_plan_gp_query, 100
rap_plan_gp_query(), 101
rap_run_gp_query, 100
rap_run_gp_query(), 53
rap_submit_extract, 101
read_prs_scores, 102
run_correlation, 103
run_gp_workflow, 103
run_imputation, 105
run_mediation, 107
run_metabolite_ora, 109
run_multi_mediator, 110
run_multi_subgroup, 111
run_protein_kegg_ora, 112
run_protein_ora, 113
run_protein_ppi_clustering, 115
run_protein_ppi_robustness, 116
run_rcs, 81, 117
run_regression, 119
run_subgroup_analysis, 120
run_weighted_analysis, 121
runmulti_competing, 122
runmulti_cox, 123
runmulti_cox(), 82, 211, 212
runmulti_cox_lag, 124
runmulti_cox_zph, 124
runmulti_gam, 125
runmulti_glm, 112, 119, 121, 126, 126
runmulti_lm, 127
runmulti_logit, 128
runmulti_logit(), 82
runmulti_negbin, 129
runmulti_trend, 129
score_protein_ppi_clusters, 130

- [select_incident_by_years](#), 131
- [sensitivity_exclude_early_events](#), 132
- [sensitivity_exclude_missing_covariates](#), 133
- [split_prs_weights](#), 134
- [split_prs_weights\(\)](#), 192
- [subset_protein_ppi](#), 135
- [summarise_gp_coverage](#), 136
- [summarise_gp_coverage\(\)](#), 46
- [summarise_gp_diagnoses](#), 137
- [summarise_gp_diagnoses\(\)](#), 46
- [summary.mediation_result](#), 137
- [tidy.mi_pooled_result](#), 138
- [ukb_check_rap_env](#), 138
- [ukb_clean_missing](#), 139
- [ukb_compare_cox_results](#), 140
- [ukb_compare_cox_results\(\)](#), 62
- [ukb_compare_sensitivity_cox](#), 141
- [ukb_compare_sensitivity_cox\(\)](#), 63
- [ukb_cox_diagnostics](#), 142
- [ukb_create_extraction_manifest](#), 143
- [ukb_decode](#), 144
- [ukb_decode_column_names](#), 145
- [ukb_decode_values](#), 146
- [ukb_demo](#), 147
- [ukb_dictionary_zh](#), 147
- [ukb_download_rap_dictionary](#), 148, 197
- [ukb_extract_fields](#), 149
- [ukb_field_info](#), 150
- [ukb_gwas_phewas_tools](#), 150
- [ukb_gwas_qc_profile](#), 10, 151
- [ukb_metadata_setup](#), 152
- [ukb_ml_as_split](#), 153, 176
- [ukb_ml_calibration](#), 154
- [ukb_ml_compare](#), 154
- [ukb_ml_compare_feature_sets](#), 155
- [ukb_ml_compare_flows](#), 157
- [ukb_ml_confusion](#), 158
- [ukb_ml_cv](#), 159
- [ukb_ml_dca](#), 160
- [ukb_ml_evaluate_test](#), 156, 160, 164
- [ukb_ml_feature_select](#), 161, 170, 188
- [ukb_ml_fit_final](#), 156, 162, 164, 165
- [ukb_ml_flow](#), 157, 158, 163
- [ukb_ml_gain_lift](#), 165
- [ukb_ml_importance](#), 166
- [ukb_ml_ks](#), 166
- [ukb_ml_metrics](#), 167
- [ukb_ml_model](#), 168
- [ukb_ml_nested_cv](#), 169, 188
- [ukb_ml_pr](#), 171
- [ukb_ml_predict](#), 171
- [ukb_ml_roc](#), 172
- [ukb_ml_roc_data](#), 78, 79, 173
- [ukb_ml_split_data](#), 164, 174, 188
- [ukb_ml_supported_models](#), 157, 175
- [ukb_ml_survival](#), 175
- [ukb_ml_survival_as_split](#), 176
- [ukb_ml_survival_evaluate_test](#), 177
- [ukb_ml_survival_feature_select](#), 178, 184
- [ukb_ml_survival_fit_final](#), 179
- [ukb_ml_survival_importance](#), 180
- [ukb_ml_survival_predict](#), 180
- [ukb_ml_survival_shap](#), 181
- [ukb_ml_survival_split_data](#), 182, 184
- [ukb_ml_survival_tune](#), 183, 184
- [ukb_ml_survival_workflow](#), 175, 184
- [ukb_ml_threshold](#), 156, 158, 164, 185, 188
- [ukb_ml_tune](#), 156, 158, 164, 170, 186, 188
- [ukb_ml_workflow](#), 170, 187
- [ukb_participant_flow](#), 188
- [ukb_plan_plink2](#), 21, 190, 191, 199
- [ukb_plan_plink2_dosage](#), 190, 190, 199
- [ukb_plan_prs](#), 191
- [ukb_plan_prs\(\)](#), 93, 134
- [ukb_plan_regenie](#), 193, 195, 201
- [ukb_plan_regenie_command](#), 194, 195, 201
- [ukb_protein_annotation](#), 195
- [ukb_prs_catalog](#), 196
- [ukb_prs_catalog\(\)](#), 49
- [ukb_query_dictionary](#), 147, 197
- [ukb_run_phewas](#), 198
- [ukb_run_plink2](#), 20, 199
- [ukb_run_prs](#), 200
- [ukb_run_regenie](#), 201
- [ukb_scale_with_parameters](#), 202
- [ukb_search_fields](#), 202
- [ukb_sensitivity_suite](#), 203
- [ukb_shap](#), 85, 165, 204
- [ukb_shap_dependence](#), 205
- [ukb_shap_force](#), 206
- [ukb_shap_summary](#), 207
- [ukb_snapshot](#), 207
- [ukb_standardize_by_train](#), 208

ukb_standardize_by_train(), [202](#)
ukb_time_skeleton, [12](#), [209](#)
ukb_top_hr_results, [210](#)
ukb_top_hr_results(), [89](#)
ukb_train_validation_cox, [211](#)
ukb_validate_columns, [212](#)
ukb_write_extraction_manifest, [213](#)
ukb_write_gwas_phewas_phenotype, [9](#), [193](#),
[213](#)
UKBAntalytica, [214](#)
UKBAntalytica-package (UKBAntalytica), [214](#)