Package: UKBAnalytica (via r-universe)

July 1, 2026

Title UK Biobank Data Processing and Survival Analysis Toolkit

Version 1.0.0

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, 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>.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/Hinna0818/UKBAnalytica, https://hinna0818.github.io/UKBAnalytica/

BugReports https://github.com/Hinna0818/UKBAnalytica/issues

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

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

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 2026-06-26 07:43:11 UTC

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

RemoteRef HEAD

RemoteSha f256d7c35f21c3a3e2d8254ac5daecc520fc1c7a

Contents

assess_balance . . . 6
build_survival_dataset . . . 7
calculate_air_pollution . . . 9
calculate_blood_pressure . . . 10
calculate_diet_score . . . 11
calculate_weights . . . 11
classify_metabolites . . . 12
coef.mediation_result . . . 13
combine_disease_definitions . . . 13
compare_data_sources . . . 14
compute_protein_ppi_metrics . . . 14
confint.mediation_result . . . 15
create_baseline_table . . . 15
create_disease_definition . . . 16
create_imputation_list . . . 18
create_medication_definition . . . 18
estimate_propensity_score . . . 19
extract_cases_by_source . . . 20
extract_diabetes_subtype_baseline . . . 21
extract_disease_diagnosis . . . 22
extract_disease_history . . . 23
extract_disease_history_sensitivity . . . 25
extract_medications . . . 25
extract_self_report_medications . . . 26
fit_mi_models . . . 27
get_death_dates . . . 28
get_disease_catalog . . . 28
get_field_info . . . 29
get_field_metadata . . . 30
get_medication_catalog . . . 31
get_pomegranate_diseases . . . 31
get_pomegranate_source_manifest . . . 32

get_predefined_diseases . . . 32
get_predefined_medications . . . 34
get_protein_ppi . . . 34
get_ukb_demo_colnames . . . 35
get_variable_info . . . 36
get_variable_set . . . 36
get_variable_sets . . . 37
load_pomegranate_portal_coding . . . 37
load_ukb_medication_coding . . . 38
load_ukb_metabolite_panel . . . 38
match_propensity . . . 39
metabolite_to_metaboanalyst_name . . . 40
parse_cancer_registry . . . 41
parse_death_records . . . 41
parse_icd10_diagnoses . . . 42
parse_icd9_diagnoses . . . 43
parse_opcs4_procedures . . . 43
parse_self_reported_illnesses . . . 44
plot.ukb_ml_flow . . . 45
plot.ukb_ml_flow_compare . . . 46
plot_balance . . . 46
plot_calibration . . . 47
plot_correlation . . . 48
plot_cox_loghr_correlation . . . 49
plot_cox_sensitivity_correlation . . . 49
plot_enrichment_lollipop . . . 50
plot_forest . . . 51
plot_go_ora_bar . . . 52
plot_heatmap . . . 53
plot_km_curve . . . 54
plot_mediation . . . 55
plot_mediation_forest . . . 56
plot_metabolite_ora_barplot . . . 57
plot_metabolite_ora_dotplot . . . 57
plot_mi_diagnostics . . . 58
plot_mi_pooled . . . 59
plot_ml_calibration . . . 60
plot_ml_compare . . . 60
plot_ml_confusion . . . 61
plot_ml_dca . . . 62
plot_ml_gain . . . 62
plot_ml_importance . . . 63
plot_ml_ks . . . 63
plot_ml_lift . . . 64
plot_ml_pr . . . 64
plot_ml_roc . . . 65
plot_ml_roc_compare . . . 65
plot_participant_flow . . . 66

plot_ps_distribution . . . 67
plot_rcs . . . 68
plot_regression_volcano . . . 69
plot_scatter . . . 70
plot_shap_beeswarm . . . 71
plot_shap_dependence . . . 73
plot_shap_force . . . 73
plot_shap_summary . . . 74
plot_stacked_bar . . . 75
plot_top_hr_bars . . . 76
plot_violin . . . 76
pool_custom_estimates . . . 77
pool_mi_models . . . 78
preprocess_baseline . . . 79
print.mediation_result . . . 80
protein_to_gene_symbol . . . 81
rank_protein_ppi_nodes . . . 82
rap_extract_pheno . . . 82
rap_find_dataset . . . 83
rap_list_fields . . . 84
rap_plan_extract . . . 85
rap_submit_extract . . . 86
run_correlation . . . 87
run_imputation . . . 87
run_mediation . . . 89
run_metabolite_ora . . . 91
run_multi_mediator . . . 92
run_multi_subgroup . . . 93
run_protein_kegg_ora . . . 94
run_protein_ora . . . 95
run_protein_ppi_clustering . . . 97
run_protein_ppi_robustness . . . 97
run_rcs . . . 99
run_regression . . . 100
run_sensitivity_mediation . . . 102
run_subgroup_analysis . . . 102
run_weighted_analysis . . . 104
runmulti_competing . . . 105
runmulti_cox . . . 106
runmulti_cox_lag . . . 106
runmulti_cox_zph . . . 107
runmulti_gam . . . 108
runmulti_glm . . . 109
runmulti_lm . . . 110
runmulti_logit . . . 111
runmulti_negbin . . . 111
runmulti_trend . . . 112
score_protein_ppi_clusters . . . 113

select_incident_by_years . . . 114
sensitivity_exclude_early_events . . . 115
sensitivity_exclude_missing_covariates . . . 116
subset_protein_ppi . . . 117
summary.mediation_result . . . 118
tidy.mi_pooled_result . . . 118
ukb_check_rap_env . . . 119
ukb_clean_missing . . . 120
ukb_compare_cox_results . . . 121
ukb_compare_sensitivity_cox . . . 122
ukb_cox_diagnostics . . . 123
ukb_create_extraction_manifest . . . 123
ukb_decode . . . 124
ukb_decode_column_names . . . 125
ukb_decode_values . . . 126
ukb_demo . . . 127
ukb_dictionary_zh . . . 127
ukb_download_rap_dictionary . . . 128
ukb_extract_fields . . . 129
ukb_field_info . . . 130
ukb_metadata_setup . . . 130
ukb_ml_as_split . . . 132
ukb_ml_calibration . . . 133
ukb_ml_compare . . . 133
ukb_ml_compare_feature_sets . . . 134
ukb_ml_compare_flows . . . 135
ukb_ml_confusion . . . 137
ukb_ml_cv . . . 138
ukb_ml_dca . . . 139
ukb_ml_evaluate_test . . . 139
ukb_ml_feature_select . . . 140
ukb_ml_fit_final . . . 141
ukb_ml_flow . . . 142
ukb_ml_gain_lift . . . 144
ukb_ml_importance . . . 145
ukb_ml_ks . . . 145
ukb_ml_metrics . . . 146
ukb_ml_model . . . 147
ukb_ml_pr . . . 148
ukb_ml_predict . . . 149
ukb_ml_roc . . . 149
ukb_ml_roc_data . . . 150
ukb_ml_split_data . . . 151
ukb_ml_supported_models . . . 152
ukb_ml_survival . . . 153
ukb_ml_survival_as_split . . . 154
ukb_ml_survival_evaluate_test . . . 155
ukb_ml_survival_feature_select . . . 155

ukb_ml_survival_fit_final . . . 156
ukb_ml_survival_importance . . . 157
ukb_ml_survival_predict . . . 158
ukb_ml_survival_shap . . . 158
ukb_ml_survival_split_data . . . 159
ukb_ml_survival_tune . . . 160
ukb_ml_survival_workflow . . . 161
ukb_ml_threshold . . . 163
ukb_ml_tune . . . 163
ukb_ml_workflow . . . 165
ukb_participant_flow . . . 166
ukb_protein_annotation . . . 167
ukb_query_dictionary . . . 168
ukb_scale_with_parameters . . . 169
ukb_search_fields . . . 169
ukb_sensitivity_suite . . . 170
ukb_shap . . . 171
ukb_shap_dependence . . . 172
ukb_shap_force . . . 173
ukb_shap_summary . . . 174
ukb_snapshot . . . 174
ukb_standardize_by_train . . . 175
ukb_time_skeleton . . . 176
ukb_top_hr_results . . . 177
ukb_train_validation_cox . . . 178
ukb_validate_columns . . . 179
ukb_write_extraction_manifest . . . 180
UKBAnalytica . . . 180

Index **182**

`assess_balance` *Assess Covariate Balance*

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

---

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	Disease key used to compute follow-up time and event status (must be in disease_definitions). If NULL, the first disease in the list is used.
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 data.table::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) <= 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 (controlled by primary_disease):

- Prevalent case (primary disease): NA (not at risk)
- Incident case: (diagnosis_date - baseline_date) / 365.25
- Censored: (min(death_date, censor_date) - 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 Event indicator for primary disease (1=event, 0=censored, NA=prevalent case)

outcome_surv_time Follow-up time in years for primary disease (NA for prevalent cases)

`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

df	A data.table containing BP columns
type	Character: "sbp" or "dbp"
prefer	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 *Calculate IPTW Weights*

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.

compare_data_sources *Compare Case Counts Across Data Sources*

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.

`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

data	a data.table containing the baseline characteristics and case/control status.
case_col	the name of the column indicating case/control status (1 for cases, 0 for controls).
factor_cols	a vector of column names that are factors (categorical variables)
continuous_cols	a vector of column names that are continuous variables.
test	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. UKB field ID for the algorithmically-defined outcome date (Category 42). For example, 42016 for COPD, 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	Integer. UKB field ID for the algorithmically-defined outcome source (Category 42). For example, 42017 for COPD source and 42015 for Asthma source. Stored as metadata for source provenance.
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

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

data	A data.frame or data.table containing all variables.
treatment	Character string specifying the treatment variable name (binary 0/1).
covariates	Character vector of covariate names used to estimate propensity scores.
method	Character string specifying the estimation method: "logistic" (default) or "gbm".
formula	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

`dt`	A data.table or data.frame containing complete UKB data.
`disease_definitions`	Named list of disease definitions.
`sources`	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.
`censor_date`	Administrative censoring date.
`baseline_col`	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` *Extract Baseline Diabetes Subtypes (T1DM/T2DM) with HbA1c Support*

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.
sources	Character vector of evidence sources. Valid options are "ICD10", "ICD9", "Self-report", "Death", "OPCS4", "CancerRegistry", "FirstOccurrence", and "Algorithm".
censor_date	Administrative censoring date.
baseline_col	Column name for baseline assessment date.
include_all	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

`dt`	A data.table or data.frame containing complete UKB data.
`diseases`	Character vector of disease names to extract. Must match keys in `disease_definitions` or predefined disease names.
`disease_definitions`	Named list of disease definitions. If NULL, uses `get_predefined_diseases`.
`sources`	Character vector specifying data sources. Default: "ICD10". Options: "ICD10", "ICD9", "Self-report", "Death", "OPCS4", "CancerRegistry", "FirstOccurrence", "Algorithm".
`baseline_col`	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 *Extract medication use from UKB drug fields*

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

Extract self-reported medication indicators from field 20003

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

data	A data.frame or data.table containing field 20003 array columns.
medications	Optional medication definition names to extract. If NULL, all predefined definitions are used.
medication_definitions	Named list of medication definitions. Defaults to `get_predefined_medications()`.
id_col	Participant identifier column.

instance	Optional UKB assessment instance. If NULL, all available instances are searched.
prefix	Prefix for output variable names.
missing_as_zero	Logical. If TRUE, participants with no valid 20003 entries are coded as 0; otherwise they are coded as NA.
return_long	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` *Query the built-in disease code catalog*

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 *Get one UK Biobank field's metadata*

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` *Get structured UK Biobank field metadata*

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 UKBAnalytica 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 UKBAnalytica 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")
)
```

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".

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 *Get information about available variables*

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 *Get one curated UK Biobank variable set*

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 *Curated UK Biobank variable sets for extraction*

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)
```

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)
```

---

match_propensity *Propensity Score Matching*

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

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.
ratio	Numeric matching ratio (1:ratio). Default 1 for 1:1 matching.
caliper	Numeric caliper width in standard deviations of PS. Default 0.2.
method	Character string specifying matching method: "nearest" or "optimal".
replace	Logical; whether to match with replacement. Default FALSE.
exact_match	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 *Parse Death Registry Records*

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_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

`plot.ukb_ml_flow` *Plot a UKB ML Flow Object*

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 assess_balance() for unmatched data.
balance_after	A data.frame from assess_balance() 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` *Plot Calibration Curve*

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 *Visualize correlation matrix as a heatmap*

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 run_correlation() or cor()).
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 *Plot training-validation Cox log(HR) concordance*

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 *Plot sensitivity-analysis Cox log(HR) concordance*

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 `ukb_compare_sensitivity_cox()`.
`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::gglollipop()` 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::gglollipop().

Value

A ggplot2 object.

---

plot_forest *Plot Forest Plot for Subgroup Analysis*

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 *Plot GO ORA results as a bar chart via TCMDATA*

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 `enrichResult` object, or a list containing `ora_result`.
...	Additional arguments passed to `TCMDATA::go_barplot()`.

Value

A `ggplot2` object.

`plot_heatmap` *Plot a publication-style heatmap*

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

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

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

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".
pathway_col	Column containing pathway names. Default "pathway".
fill_color	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 mi_pooled_result from pool_mi_models().
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 *Plot Multiple Imputation Pooled Results*

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 mi_pooled_result from pool_mi_models().
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 mi_result 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 *Plot Calibration Curve*

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 *Plot Model Comparison*

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 *Plot Confusion Matrix*

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 *Plot Decision Curve Analysis*

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 ukb_ml_dca object from ukb_ml_dca()
title	Plot title
...	Additional arguments

Value

A ggplot2 object

plot_ml_gain *Plot Gain Curve*

Description

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

Usage

```
plot_ml_gain(object, title = "Gain 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_importance *Plot Variable Importance*

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 ukb_ml object from ukb_ml_model()
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 *Plot KS Curve*

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 *Plot Lift Curve*

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 *Plot PR Curve*

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 *Plot ROC Curves*

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 *Plot One or More ROC Curves from Tidy ROC Data*

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 theme().
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 ukb_participant_flow object.
show_removed	Logical. If TRUE, annotate removals at each step.
show_events	Logical. If TRUE, include event counts in labels when outcome_col was supplied to ukb_participant_flow().
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 data.frame or data.table 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 *Plot a restricted cubic spline exposure-response curve*

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

x	A ukb_rcs object from run_rcs.
...	Additional arguments (currently unused).
show_distribution	Logical. Whether to overlay an exposure distribution layer. Default TRUE.
distribution	One of "histogram" (default), "density", or "rug".
show_ref	Logical. Whether to mark the reference value with a point. Default TRUE.
show_p	Logical. Whether to annotate P-overall and P-nonlinear. Default TRUE.
show_knots	Logical. Whether to annotate the knot count. Default TRUE.
curve_color	Character. Hex color for the main curve and ribbon. Default "#2166AC" (deep blue).
dist_color	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 -log10(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 *Plot a publication-style scatter plot*

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 *Plot SHAP Beeswarm Summary*

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 ukb_shap object from ukb_shap.
max_features	Maximum number of features to display.
label_map	Optional named vector or data.frame mapping feature names to display labels. For a data.frame, columns are controlled by feature_col and label_col.
feature_col	Feature column in label_map when it is a data.frame.
label_col	Label column in label_map when it is a data.frame.
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 NULL, no title is shown.
xlab	X-axis label.
legend_title	Legend title.
base_size	Base font size.
return_data	Logical. If TRUE, returns a list with plot and data; otherwise returns only the ggplot object.
...	Additional arguments reserved for future use.

Value

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

plot_shap_dependence *Plot SHAP Dependence*

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 *Plot SHAP Force (Waterfall)*

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 *Plot SHAP Summary*

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` *Plot a publication-style stacked bar chart*

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

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

Value

A ggplot object.

plot_top_hr_bars *Plot top positive and inverse Cox associations*

Description

Plot top positive and inverse Cox associations

Usage

```
plot_top_hr_bars(
  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 *Plot a publication-style violin plot*

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

data	A data.frame.
x	Character column name for groups.
y	Character numeric column name.
fill	Optional fill grouping column. Defaults to x.
palette	Optional vector of fill colors.
add_boxplot	Logical. Overlay a narrow boxplot.
add_points	Logical. Overlay jittered observations.
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.

---

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

estimates	A list of numeric vectors containing point estimates from each imputed dataset. All vectors must have the same length.
variances	A list of variance-covariance matrices (or single variances as 1x1 matrices) corresponding to the estimates.
df.complete	Complete-data degrees of freedom. Default Inf.
conf.level	Confidence level for intervals. Default 0.95.
labels	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`.

---

pool_mi_models *Pool Results from Multiple Imputation Models*

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

`models`	A list of fitted model objects (one per imputed dataset). If NULL, models will be fitted using `datasets` and `formula`.
`datasets`	A list of data.frames or an `imputationList` object. Required if `models` is NULL.
`formula`	A formula specifying the model. Required if `datasets` is provided.
`model_type`	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
`family`	A `family` object for GLM. If NULL, inferred from `model_type`.
`df.complete`	Complete-data degrees of freedom for small-sample correction. Default is Inf (large sample approximation).
`conf.level`	Confidence level for intervals. Default 0.95.
`exponentiate`	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

`preprocess_baseline` *Preprocess UKB baseline variables*

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

df	A data.table or data.frame containing UKB data from rap platform export.
variables	Character vector of variable names to process. Use `get_variable_info()` to see available variables.
custom_mapping	Optional named list for user-defined variable mappings. Each element should have: `ukb_col` (required), `description` (optional). Example: `list(my_var = list(ukb_col = "p12345_i0", description = "My custom var"))`
missing_action	Character. How to handle missing values: • "keep": Keep as NA (default) • "drop": Remove rows with any missing values in processed variables
invalid_codes	Numeric vector of UKB codes to treat as missing. Default: c(-1, -3) 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

x	An object of class "mediation_result".
...	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
)
```

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. Identifier type used by Bioconductor when `mapping_table` does not fully resolve the input. 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"`.
`drop_unmapped`	Logical. If `TRUE`, drop rows without a mapped gene symbol. Default is `TRUE`.

Value

A data.frame with columns `protein`, `gene_symbol`, and `mapping_source`.

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 *Extract RAP Phenotype Data Synchronously*

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

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

Value

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

---

rap_find_dataset *Find the RAP Dataset File in the Current Project*

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 *List Approved RAP Dataset Fields*

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` *Plan a RAP Phenotype Extraction*

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_submit_extract` *Submit a RAP Table-Exporter Phenotype Extraction Job*

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

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

Value

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

run_correlation *Calculate correlation between variables*

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_imputation *Multiple imputation and merge back to full data*

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

`mediator_type`
: Character string: "continuous" or "binary". Default "continuous".

`outcome_type`
: 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".

`interaction`
: Logical; whether to include exposure-mediator interaction in the outcome model. Default TRUE.

`boot`
: Logical; whether to use bootstrap for confidence intervals. Default FALSE.

`boot_n`
: Integer; number of bootstrap replicates. Default 1000.

`conf_level`
: 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_ora` *Run metabolite over-representation analysis*

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_ora(
  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 `stats::p.adjust()`. 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, and library.

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")
```

---

run_multi_mediator *Run Multiple Mediator Analysis*

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

data	A data.frame or data.table containing all variables.
exposure	Character string specifying the exposure (treatment) variable name.
mediators	Character vector of mediator variable names.
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.
mediator_type	Character string: "continuous" or "binary". Default "continuous".
outcome_type	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 *Run Multiple Subgroup Analyses*

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` *Run KEGG ORA enrichment for proteomics hits*

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
)
```

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.

Value

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

---

run_protein_ora *Run GO ORA enrichment for proteomics hits*

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
)
```

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.

Value

A list with components gene_symbols, mapping, 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

`ppi`	An `igraph` object, a list returned by `get_protein_ppi()`, or a list containing a `graph` element.
`method`	Clustering algorithm. Options are `"fastgreedy"`, `"louvain"`, `"mcode"`, and `"mcl"`.
`...`	Method-specific arguments passed to the selected clustering helper, such as `n_clusters` for `"fastgreedy"`, `resolution` for `"louvain"`, `vwp` for `"mcode"`, or `inflation` 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",
  rewire_niter = 10L,
  seed = 42L
)
```

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.

Value

A list with components targets, mapping, and robustness.

run_rcs *Fit a restricted cubic spline exposure-response model*

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

`data`	A data.frame containing all required columns.
`exposure`	Character. Name of the continuous exposure variable.
`covariates`	Character vector of covariate names, or `NULL`.
`model_type`	One of `"cox"`, `"logistic"`, `"linear"`.
`endpoint`	Character vector of length 2 giving `c(time, status)` column names. Required when `model_type = "cox"`.
`outcome`	Character. Outcome column name. Required for `"logistic"` and `"linear"`.
`knots`	Integer. Number of knots (3-7). If `NULL`, the knot count with the lowest AIC within `knot_range` is chosen automatically.
`knot_range`	Integer vector of candidate knot counts for AIC selection. Default `3:7`.
`ref`	Numeric. Reference value for the exposure. If `NULL`, `ref_quantile` is used.
`ref_quantile`	Numeric (0-1). Quantile of the exposure used as the reference when `ref` is `NULL`. Default `0.5` (median).

conf_level	Numeric. Confidence level for intervals. Default 0.95.
trim_quantiles	Numeric vector of length 2. Exposure values outside these quantiles are excluded before fitting. Default c(0.01, 0.99).
grid_size	Integer. Number of points in the prediction grid. Default 200.
backend	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 *Run a regression model (unified interface)*

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

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

Description

Perform sensitivity analysis to assess the impact of unmeasured confounding on mediation effect estimates.

Usage

```
run_sensitivity_mediation(
  mediation_result,
  rho_values = seq(-0.9, 0.9, by = 0.1)
)
```

Arguments

mediation_result	An object of class "mediation_result" from `run_mediation()`.
rho_values	Numeric vector of sensitivity parameter values (correlation between unmeasured confounder and mediator/outcome residuals). Default seq(-0.9, 0.9, by = 0.1).

Details

This function evaluates how the indirect effect would change if there were unmeasured confounding of the mediator-outcome relationship. The rho parameter represents the correlation between residuals that would be induced by an unmeasured confounder.

A robust mediation effect should remain significant across a range of plausible rho values.

Value

A data.frame with effect estimates under different rho values.

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 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".
ref_level	Character string specifying the reference level for the subgroup variable. If NULL, 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` *Run Multiple Fine-Gray Competing-Risk Models*

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 *Run multiple Cox proportional hazards models*

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

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

Value

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

runmulti_cox_lag *Run Lagged Cox Sensitivity Analyses*

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 *Run Multiple Cox Models with PH Diagnostics*

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

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

Value

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

---

runmulti_gam *Run multiple generalised additive models*

Description

Fit GAMs (mgcv::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

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

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 stats::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. stats::poisson. • A family object, e.g. stats::poisson(link = "sqrt").
outcome	A character string specifying the outcome column.
covariates	A character vector of covariate names. Default NULL.
...	Additional arguments passed to stats::glm().

Value

A data.frame with columns: variable, family, link, beta, lower95, upper95, pvalue, n. For log- or logit-link families exp(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

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

Value

A data.frame with columns: variable, beta, lower95, upper95, pvalue.

runmulti_logit *Run multiple logistic regression models*

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

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

Value

A data.frame with columns: `variable`, `OR`, `lower95`, `upper95`, `pvalue`.

runmulti_negbin *Run multiple negative-binomial regression models*

Description

Fit negative-binomial GLMs (`MASS::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

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

Value

A data.frame with columns: variable, IRR, lower95, upper95, pvalue, theta, n. IRR is the incidence rate ratio (exp(beta)). theta is the estimated negative-binomial dispersion parameter (larger values indicate less overdispersion).

---

runmulti_trend *Run Grouped-Exposure Trend Tests*

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

data	A data.frame or data.table.
main_var	Character vector of grouped exposure variable names.
outcome	Outcome column for logistic or linear models.
covariates	Optional character vector of covariates.
model_type	One of "cox", "logistic", or "linear".
endpoint	Character vector of length 2 for Cox models.

ref_level	Optional reference level applied to every grouped exposure.
score_method	One of "integer", "median", or "custom".
custom_scores	Optional named list of custom score mappings.
include_level_estimates	Logical; if TRUE, include category-specific estimates.
...	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 get_protein_ppi(), 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

data	A data.frame or data.table.
covariates	Character vector of covariate names to check.
copy	Logical scalar. If TRUE and data is a data.table, work on a copied object before filtering.

stepwise	Logical scalar. If TRUE, apply covariate missingness filters sequentially in the order provided and record a row-level flow table in attr(result, "complete_case_flow").
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_missing_covariates(
  data = mtcars,
  covariates = c("hp", "wt")
)
```

---

subset_protein_ppi *Filter a STRING PPI network via TCMDATA*

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 get_protein_ppi(), or a list containing a graph element.
n	Integer. Number of top-degree nodes to keep. If NULL, 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 TRUE.

Value

An igraph object.

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

x	An mi_pooled_result object.
conf.int	Logical; include confidence intervals? Default TRUE.
conf.level	Confidence level. Default 0.95.
exponentiate	Logical; exponentiate estimates? Default FALSE.
...	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 `dx whoami` and `dx env --bash` to check DNAnexus authentication and the active project context. This check does not inspect participant-level data.
`check_write`	Logical. If `TRUE` and `output_dir` 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 *Clean UK Biobank Missing and Non-response Values*

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 *Compare Cox results between training and validation sets*

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` *Compare sensitivity Cox results against a main analysis*

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 *Diagnose Proportional Hazards Assumptions for a Cox Model*

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

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

Description

Decode UK Biobank RAP exports

Usage

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

Arguments

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

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

Value

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

---

ukb_decode_values *Decode UK Biobank coded values*

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 *Generate a small synthetic UK Biobank-style demo dataset*

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 *Chinese UK Biobank field-path dictionary*

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

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

Value

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

ukb_extract_fields *Extract UK Biobank fields from a search result or field list*

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

x	Optional object returned by ukb_search_fields() or ukb_field_info().
field_id	Optional UKB field IDs. Ignored when x supplies IDs.
metadata	Optional object from ukb_metadata_setup().
mode	"plan" returns a RAP extraction plan, "sync" calls rap_extract_pheno(), and "job" calls rap_submit_extract().
top_n	Optional number of top search-result fields to extract.
dataset	Optional RAP .dataset file name.
entity	RAP entity. Defaults to "participant".
...	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 *Inspect one UK Biobank field*

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_metadata_setup *Set up UK Biobank metadata for search, extraction, and decoding*

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 dx extract_dataset -ddd, 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 rap_list_fields().
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 tools::R_user_dir("UKBAnalytica", "cache").
refresh	Logical. Passed to rap_list_fields() 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

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

Value

A ukb_ml_split object.

ukb_ml_calibration *Calibration Curve Analysis*

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 object
newdata	Optional new data
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

ukb_ml_calibration object

ukb_ml_compare *Compare Multiple ML Models*

Description

Compare performance of multiple trained ML models.

Usage

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

Arguments

...	ukb_ml objects to compare
models	Alternative: list of ukb_ml objects
metrics	Metrics to compare
test_data	Optional common test data
plot	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

split	A ukb_ml_split object.
feature_sets	Named list of character vectors. Each vector contains the feature names used by one model.
outcome	Optional outcome column. Defaults to split$outcome.
model	Model type passed to ukb_ml_tune and ukb_ml_fit_final.
outcome_type	Outcome type. Currently this helper is intended for binary classification.
model_labels	Optional labels for feature sets. Can be a named vector or a vector in the same order as feature_sets.
param_grid	Optional parameter grid. Can be a single grid shared by all models or a named list keyed by feature-set name.
tune_params	Additional arguments passed to ukb_ml_tune.
threshold_method	"none", "fixed", or "youden".
threshold_params	Additional arguments passed to ukb_ml_threshold.
metrics	Optional metric names passed to ukb_ml_evaluate_test.
positive_class	Optional positive class label for binary outcomes.
use_validation_in_refit	Logical passed to ukb_ml_fit_final.
seed	Optional random seed.
verbose	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_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 `feature_sets` is `NULL`.
`data`, `split`, `train_data`, `test_data`, `validation_data`, `id_col`	Passed to `ukb_ml_flow`.
`outcome`	Optional outcome column. Required when `formula` is `NULL`.
`feature_sets`	Optional named list of feature vectors. If `NULL`, one feature set is derived from `formula` or `features`.
`features`	Optional feature names used when `formula` and `feature_sets` are `NULL`.
`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_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 *Confusion Matrix*

Description

Generate confusion matrix for classification model.

Usage

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

Arguments

object	A ukb_ml 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

ukb_ml_cv *Cross-Validation for ML Models*

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 *Decision Curve Analysis*

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 object
newdata	Optional new data
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 with field data containing: threshold, net_benefit_model, net_benefit_all, net_benefit_none

ukb_ml_evaluate_test *Evaluate the Final Model Once on the Frozen Test Set*

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 *Refit the Final ML Model on Training Development Data*

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 *Run a Complete Single-Model UKB ML Flow*

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 ukb_ml_split 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 ukb_ml_tune and ukb_ml_fit_final.
model_id, model_label	Optional model identifier and display label.
outcome_type	Outcome type.
split_params	List passed to ukb_ml_split_data when splitting a full data object.
param_grid	Optional hyperparameter grid.
tune	Logical. Run hyperparameter tuning.
tune_params	Additional arguments passed to ukb_ml_tune.
best_params	Optional final model parameters when tune = FALSE.
threshold_method	"none", "fixed", or "youden".
threshold_params	Additional arguments passed to ukb_ml_threshold.
metrics	Optional metric names passed to ukb_ml_evaluate_test.
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 *Gain and Lift Curve Analysis*

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 ukb_ml 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 *Get Variable Importance*

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 *KS Curve Analysis*

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 *Calculate Model Performance Metrics*

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_pr *Precision-Recall Curve Analysis*

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 object
newdata	Optional new data
plot	Whether to create the PR plot (default TRUE)
n_thresholds	Number of threshold points (default 200)
...	Additional arguments

Value

A ukb_ml_pr object with fields: data (threshold/precision/recall), auprc, prevalence

ukb_ml_predict *Predict from ML Model*

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 *ROC Curve Analysis*

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 ci.auc(); usually "delong" or "bootstrap".
quiet	Logical passed to roc().

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

`df`	A data.frame or data.table.
`outcome`	Outcome column name. If NULL, a legacy random split is returned with `internal_validation` populated.
`outcome_type`	One of `"auto"`, `"binary"`, `"multiclass"`, or `"continuous"`.
`split`	Either `"train_test"` or `"train_valid_test"`.
`train_ratio`	Training proportion.
`validation_ratio`	Validation proportion for train/validation/test.
`test_ratio`	Test proportion.
`split_ratio`	Deprecated compatibility alias for `train_ratio`.
`stratify_by`	`"auto"`, `"outcome"`, `"custom"`, `"none"`, or an older-style column name.

stratify_col	Column used when stratify_by = "custom".
regression_bins	Number of quantile bins for continuous outcome stratification.
seed	Optional random seed.
verbose	Logical. Print split summary.

Value

A ukb_ml_split object.

---

ukb_ml_supported_models

List Supported Machine Learning Models

Description

Returns the machine-learning algorithms supported by the UKBAnalytica 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 *Train Survival Machine Learning Model*

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., Surv(time, event) ~ x1 + x2)
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

train_data	Training/development data.
test_data	Frozen test data.
validation_data	Optional validation data.
time	Survival time column.
event	Event indicator column coded 0/1.
id_col	Optional participant ID column used to check overlap.
check_overlap	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. Naive time-specific Brier scores are also reported for requested prediction times without IPCW adjustment.

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

split	A ukb_ml_survival_split object.
formula	Survival formula.
model	Survival model type.
best_params	Model parameters.
feature_spec	Optional feature-selection result.
seed	Optional random seed.
verbose	Logical.
...	Additional arguments passed to the fitter.

Value

A ukb_ml_survival_final object.

---

ukb_ml_survival_importance

Get Variable Importance for Survival Model

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 *Predict from Survival ML Model*

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 *SHAP Values for Survival Models*

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,
  ...
)
```

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
...	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

df	A data.frame or data.table.
time	Survival time column.
event	Event indicator column coded 0/1.
split	Either "train_test" or "train_valid_test".
train_ratio	Training proportion.
validation_ratio	Validation proportion for train/validation/test.
test_ratio	Test proportion.
stratify_by	"event", "custom", "none", or a column name.
stratify_col	Column used when stratify_by = "custom".
seed	Optional random seed.
verbose	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

formula	Survival formula, for example Surv(time, event) ~ x1 + x2.
data	Optional full dataset. Required when split is NULL.
split	Optional ukb_ml_survival_split object.
model	"cox", "rsf", "gbm_surv", or "coxnet".
split_params	List passed to ukb_ml_survival_split_data.
feature_select	"none", "filter", or "glmnet".
feature_params	List passed to ukb_ml_survival_feature_select.
tune	Logical. Run hyperparameter tuning.
tune_params	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 *Learn a Binary Classification Threshold*

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

split	A ukb_ml_split object.
formula	Model formula.
model	Model type.
outcome_type	Outcome type.
search	"grid", "random", or "bayes". Bayesian search currently requires rBayesianOptimization; if unavailable it falls back to random search with the same parameter space.
param_grid	List or data.frame of candidate parameters.
param_space	Parameter space for random or Bayesian search.
n_iter	Number of random/Bayesian iterations.
resampling	"cv" or "validation".
folds	Number of CV folds.
metric	Metric to optimize.
maximize	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 *Run a Frozen-Test UKB ML Workflow*

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,
  ...
)
```

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.

Value

A ukb_ml_workflow object.

---

ukb_participant_flow *Build a participant flow table*

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

data	A data.frame or data.table.
steps	A named list of rules. Each rule can be: a one-sided formula such as ~ !is.na(age), a function returning a logical vector, a logical vector, or a character vector of variable names to retain complete cases.
id_col	Optional participant identifier column. If supplied, duplicate non-missing IDs are reported as an error.
outcome_col	Optional 0/1 outcome column used to count events after each step.
event_value	Value in outcome_col indicating an event. Defaults to 1.
start_label	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_protein_annotation

Annotate Olink-style protein variables

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_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

`query`	Character vector of query terms, Chinese variable names, English names, UKB field IDs, or UKB/RAP column names.
`official_dict`	Optional official RAP data dictionary CSV. If `NULL`, `ukb_download_rap_dictionary` is called.
`zh_dict`	Optional Chinese dictionary CSV. Defaults to the UKBAnalytica built-in Chinese dictionary.
`dataset`	RAP `.dataset` file used when `official_dict` is `NULL`.
`output_dir`	Directory used when downloading the official dictionary.
`language`	Query language. `"auto"` detects Chinese, field IDs, and column names.
`translation_map`	Optional data.frame with columns `zh` and `en`, or a named character vector mapping Chinese terms to English query terms.
`max_results`	Maximum official dictionary matches returned per query.
`min_score`	Minimum internal matching score for official dictionary matches.
`require_rap`	Logical. Require a RAP-like environment before querying.
`timeout`	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_scale_with_parameters *Standardize variables using existing scaling parameters*

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 *Search UK Biobank fields*

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 ukb_metadata_setup().
max_results	Maximum number of rows to return.
search_in	Columns to search.
...	Arguments passed to ukb_metadata_setup() 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 *Compute SHAP Values*

Description

Calculate SHAP values for model interpretation. SHAP values explain each feature's contribution to individual predictions.

Usage

```
ukb_shap(
  object,
  data = NULL,
  nsim = 100,
  sample_n = NULL,
  seed = NULL,
```

```
  verbose = TRUE,
  class_level = NULL,
  method = c("auto", "permutation", "xgboost"),
  ...
)
```

Arguments

object	A ukb_ml_workflow, ukb_ml_final, or legacy ukb_ml object.
data	Data for SHAP computation. If object is a ukb_ml_workflow and data = NULL, the frozen test set is used. If object is a ukb_ml_final, data is required.
nsim	Number of Monte Carlo samples for SHAP estimation (default 100). Ignored when method = "xgboost".
sample_n	Optional; subsample observations for large datasets
seed	Random seed
verbose	Print progress
class_level	Optional class to explain for multiclass ukb_ml_workflow/ukb_ml_final objects.
method	SHAP backend. "auto" uses the native XGBoost contribution backend for XGBoost models and an internal permutation approximation otherwise.
...	Additional arguments

Value

A ukb_shap object containing:

- shap_values: Matrix of SHAP values (n x p)
- baseline: Model baseline (expected) value
- feature_names: Names of features
- feature_values: Original feature values

---

ukb_shap_dependence *SHAP Dependence Values*

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 *SHAP Force Plot Data*

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 *SHAP Summary Statistics*

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 *Record or Retrieve UKB Cohort Snapshots*

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

data	Optional data.frame or data.table. If supplied, records a new snapshot.
label	Snapshot label. Required when recording a new snapshot.
id	Snapshot stream identifier. Use separate IDs for independent pipelines in the same R session.
reset	Logical. If TRUE, clears the snapshot history for id.
verbose	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

train_data	Training data.
validation_data	Optional validation data.
variables	Character vector of variables to standardize.
center	Logical. If TRUE, subtract the training-set mean.
scale	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` *Build a UK Biobank follow-up time skeleton*

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

`data`	A data.frame or data.table containing UK Biobank columns.
`id_col`	Participant identifier column. Default `"eid"`.
`baseline_col`	Baseline assessment date column. Default `"p53_i0"`.
`birth_year_col`	Year-of-birth column. Default `"p34"`.
`birth_month_col`	Month-of-birth column. Default `"p52"`.
`age_col`	Age-at-baseline column. Default `"p21022"`. If missing, age is approximated from baseline date and birth year/month when available.
`death_date_cols`	Death date columns or a regular expression used to identify them. Default `"^(participant\.)?p40000_i[0-9]+$"`.
`lost_to_followup_col`	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 runmulti_cox().

standardize_main_vars
: Logical. If TRUE, standardize main_vars using training-set means and SDs.

add_protein_annotation
: Logical. If TRUE, add parsed protein names and gene symbols for Olink-style protein columns.

protein_prefix
: Regular expression prefix removed from protein columns.

train_label
: Training-set label.

validation_label
: Validation-set label.

comparison_train_prefix
: Prefix for training columns in the comparison table.

comparison_validation_prefix
: Prefix for validation columns in the comparison table.

p_adjust_methods
: P-value adjustment methods.

alpha
: Significance threshold.

...
: Additional arguments passed to runmulti_cox().

Value

A list containing scaled data, scaling parameters, Cox results, and comparison summaries.

---

ukb_validate_columns *Validate requested columns against a data object*

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

data
: A data.frame/data.table or a character vector of available column names.

columns
: Character vector of requested column names.

ignore_entity_prefix
: Logical. If TRUE, compare both original names and names with a leading "participant." prefix removed.

error
: 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)
```

`UKBAnalytica` *UKBAnalytica: UK Biobank Data Processing and Survival Analysis Toolkit*

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/UKBAnalytica
- https://hinna0818.github.io/UKBAnalytica/
- Report bugs at https://github.com/Hinna0818/UKBAnalytica/issues

Index

∗ **datasets**
ukb_dictionary_zh, 127

assess_balance, 6

build_survival_dataset, 7, *22*, *176*, *181*
build_survival_dataset(), *114*

calculate_air_pollution, 9
calculate_blood_pressure, 10
calculate_diet_score, 11
calculate_weights, 11
classify_metabolites, 12
coef.mediation_result, 13
combine_disease_definitions, 13
compare_data_sources, 14
compute_protein_ppi_metrics, 14
confint.mediation_result, 15
create_baseline_table, 15
create_disease_definition, *8*, 16
create_imputation_list, 18
create_medication_definition, 18

estimate_propensity_score, 19
extract_cases_by_source, 20, *23*, *181*
extract_diabetes_subtype_baseline, 21
extract_disease_diagnosis, 22
extract_disease_history, 23
extract_disease_history_sensitivity, 25
extract_medications, 25
extract_self_report_medications, 26

fit_mi_models, 27

get_death_dates, 28
get_disease_catalog, 28
get_field_info, 29
get_field_metadata, 30
get_medication_catalog, 31
get_pomegranate_diseases, 31
get_pomegranate_source_manifest, 32
get_predefined_diseases, *22–24*, 32
get_predefined_diseases(), *28*, *34*
get_predefined_medications, 34
get_protein_ppi, 34
get_ukb_demo_colnames, 35
get_variable_info, 36
get_variable_info(), *37*, *124*
get_variable_set, 36
get_variable_sets, 37
get_variable_sets(), *124*

load_pomegranate_portal_coding, 37
load_ukb_medication_coding, 38
load_ukb_metabolite_panel, 38

match_propensity, 39
metabolite_to_metaboanalyst_name, 40

parse_cancer_registry, 41
parse_death_records, 41, *181*
parse_icd10_diagnoses, 42, *181*
parse_icd9_diagnoses, 43, *181*
parse_opcs4_procedures, 43, *181*
parse_self_reported_illnesses, 44, *181*
plot.ukb_ml_flow, 45
plot.ukb_ml_flow_compare, 46
plot.ukb_rcs *(*plot_rcs*)*, 68
plot_balance, 46
plot_calibration, 47
plot_correlation, 48
plot_cox_loghr_correlation, 49
plot_cox_sensitivity_correlation, 49
plot_enrichment_lollipop, 50
plot_forest, 51
plot_go_ora_bar, 52
plot_heatmap, 53
plot_km_curve, 54
plot_mediation, 55
plot_mediation_forest, 56

plot_metabolite_ora_barplot, 57
plot_metabolite_ora_dotplot, 57
plot_mi_diagnostics, 58
plot_mi_pooled, 59
plot_ml_calibration, 60
plot_ml_compare, 60
plot_ml_confusion, 61
plot_ml_dca, 62
plot_ml_gain, 62
plot_ml_importance, 63
plot_ml_ks, 63
plot_ml_lift, 64
plot_ml_pr, 64
plot_ml_roc, 65
plot_ml_roc_compare, *46*, 65
plot_participant_flow, 66
plot_ps_distribution, 67
plot_rcs, 68
plot_regression_volcano, 69
plot_scatter, 70
plot_shap_beeswarm, 71
plot_shap_dependence, 73
plot_shap_force, 73
plot_shap_summary, 74
plot_stacked_bar, 75
plot_top_hr_bars, 76
plot_violin, 76
pool_custom_estimates, 77
pool_mi_models, 78
preprocess_baseline, 79
preprocess_baseline(), *37*
print.mediation_result, 80
protein_to_gene_symbol, 81
protein_to_gene_symbol(), *167*

rank_protein_ppi_nodes, 82
rap_extract_pheno, 82
rap_find_dataset, 83, *128*
rap_list_fields, 84
rap_plan_extract, 85
rap_submit_extract, 86
run_correlation, 87
run_imputation, 87
run_mediation, 89
run_metabolite_ora, 91
run_multi_mediator, 92
run_multi_subgroup, 93
run_protein_kegg_ora, 94
run_protein_ora, 95
run_protein_ppi_clustering, 97
run_protein_ppi_robustness, 97
run_rcs, *68*, 99
run_regression, 100
run_sensitivity_mediation, 102
run_subgroup_analysis, 102
run_weighted_analysis, 104
runmulti_competing, 105
runmulti_cox, 106
runmulti_cox(), *69*, *178*, *179*
runmulti_cox_lag, 106
runmulti_cox_zph, 107
runmulti_gam, 108
runmulti_glm, *94*, *101*, *103*, 109, *109*
runmulti_lm, 110
runmulti_logit, 111
runmulti_logit(), *69*
runmulti_negbin, 111
runmulti_trend, 112

score_protein_ppi_clusters, 113
select_incident_by_years, 114
sensitivity_exclude_early_events, 115
sensitivity_exclude_missing_covariates, 116
stats::p.adjust(), *91*
subset_protein_ppi, 117
summary.mediation_result, 118

tidy.mi_pooled_result, 118

ukb_check_rap_env, 119
ukb_clean_missing, 120
ukb_compare_cox_results, 121
ukb_compare_cox_results(), *49*
ukb_compare_sensitivity_cox, 122
ukb_compare_sensitivity_cox(), *50*
ukb_cox_diagnostics, 123
ukb_create_extraction_manifest, 123
ukb_decode, 124
ukb_decode_column_names, 125
ukb_decode_values, 126
ukb_demo, 127
ukb_dictionary_zh, 127
ukb_download_rap_dictionary, 128, *168*
ukb_extract_fields, 129
ukb_field_info, 130
ukb_metadata_setup, 130
ukb_ml_as_split, 132, *154*

ukb_ml_calibration, 133
ukb_ml_compare, 133
ukb_ml_compare_feature_sets, 134
ukb_ml_compare_flows, 135
ukb_ml_confusion, 137
ukb_ml_cv, 138
ukb_ml_dca, 139
ukb_ml_evaluate_test, *135*, 139, *143*
ukb_ml_feature_select, 140, *165*
ukb_ml_fit_final, *135*, 141, *143*, *144*
ukb_ml_flow, *135–137*, 142
ukb_ml_gain_lift, 144
ukb_ml_importance, 145
ukb_ml_ks, 145
ukb_ml_metrics, 146
ukb_ml_model, 147
ukb_ml_pr, 148
ukb_ml_predict, 149
ukb_ml_roc, 149
ukb_ml_roc_data, *65*, *66*, 150
ukb_ml_split_data, *143*, 151, *165*
ukb_ml_supported_models, *136*, 152
ukb_ml_survival, 153
ukb_ml_survival_as_split, 154
ukb_ml_survival_evaluate_test, 155
ukb_ml_survival_feature_select, 155, *162*
ukb_ml_survival_fit_final, 156
ukb_ml_survival_importance, 157
ukb_ml_survival_predict, 158
ukb_ml_survival_shap, 158
ukb_ml_survival_split_data, 159, *162*
ukb_ml_survival_tune, 160, *162*
ukb_ml_survival_workflow, *153*, 161
ukb_ml_threshold, *135*, *137*, *143*, 163, *166*
ukb_ml_tune, *135*, *136*, *143*, 163, *165*
ukb_ml_workflow, 165
ukb_participant_flow, 166
ukb_protein_annotation, 167
ukb_query_dictionary, *127*, 168
ukb_scale_with_parameters, 169
ukb_search_fields, 169
ukb_sensitivity_suite, 170
ukb_shap, *72*, *144*, 171
ukb_shap_dependence, 172
ukb_shap_force, 173
ukb_shap_summary, 174
ukb_snapshot, 174
ukb_standardize_by_train, 175
ukb_standardize_by_train(), *169*
ukb_time_skeleton, *8*, 176
ukb_top_hr_results, 177
ukb_top_hr_results(), *76*
ukb_train_validation_cox, 178
ukb_validate_columns, 179
ukb_write_extraction_manifest, 180
UKBAnalytica, 180
UKBAnalytica-package (UKBAnalytica), 180