# Package: leo.gwas (via r-universe)

June 10, 2026

**Title** Layered Exploratory Omics (LEO)

**Version** 0.0.2

**Description** Although in early development, the Layered Exploratory Omics (LEO) is a tool specifically designed for deep analysis of genomic data. It employs a multi-layered exploratory approach to help researchers uncover the complex biological information hidden behind their data.

**URL** https://laleoarrow.github.io/leo.gwas/

**License** file LICENSE

**Imports** annotables (>= 0.2.0), AnnotationDbi, biomaRt, BSgenome, cli, data.table, dplyr, GenomicFeatures, GenomicRanges, ggrastr, ggplot2, ggsci, glmnet, glue, ieugwasr, LDlinkR, leo.basic (>= 0.0.1), magrittr, pROC, purrr, RColorBrewer, rlang, rtracklayer, stringr, tibble, tidyr, TwoSampleMR, vroom, S4Vectors, ggbeeswarm, locuszoomr, org.Hs.eg.db, plyr, scales

**Suggests** caret, catboost, knitr, minfi, MRlap, pacman, patchwork, plinkbinr, Rmpfr, rmarkdown, SNPlocs.Hsapiens.dbSNP155.GRCh37, SNPlocs.Hsapiens.dbSNP155.GRCh38, TxDb.Hsapiens.UCSC.hg19.knownGene, TxDb.Hsapiens.UCSC.hg38.knownGene, txdbmaker

**Remotes** github::MRCIEU/ieugwasr, github::MRCIEU/TwoSampleMR, github::catboost/catboost/catboost/R-package, github::laleoarrow/leo.basic, github::stephenturner/annotables, github::n-mounier/MRlap

**Config/leo.gwas/suggests_usage** caret -> dr.prs(), catboost_prs(), lasso_prs(); catboost -> dr.prs(), catboost_prs(), catboost_prs_target(), catboost_prs_rank(); minfi -> annotate_cpg_sites(); SNPlocs.Hsapiens.dbSNP155.GRCh37 -> add_rsid(), add_rsid_using_ref(); SNPlocs.Hsapiens.dbSNP155.GRCh38 -> add_rsid(), add_rsid_using_ref(); Rmpfr -> chisq_p_value(); TxDb.Hsapiens.UCSC.hg19.knownGene -> map_gene_to_chrbp_using_TxDb();

TxDb.Hsapiens.UCSC.hg38.knownGene ->
map_gene_to_chrbp_using_TxDb()

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**Config/pak/sysreqs** libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libbz2-dev libicu-dev libjpeg-dev liblzma-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

**Repository** https://laleoarrow.r-universe.dev

**Date/Publication** 2026-03-15 09:04:55 UTC

**RemoteUrl** https://github.com/laleoarrow/leo.gwas

**RemoteRef** HEAD

**RemoteSha** 2d18ab01483460a980bafe5f015ede4a500bdd66

# Contents

---

`across_df_na` *Across a df to count na*

---

## Description

This function summarize NA values in each column of a data frame.

## Usage

```
across_df_na(df)
```

### Arguments

| | |
|---|---|
| df | a data frame |

### Value

a data frame with the number of NA values in each column

### Examples

```
df <- data.frame(a = c(1, 2, NA, 4), b = c(NA, 2, 3, 4))
across_df_na(df)
```

---

## across_df_TF *Across a df to count TRUE and FALSE*

---

### Description

This function summarizes both TRUE and FALSE values in each column of a data frame.

### Usage

```
across_df_TF(df, type = "T")
```

### Arguments

| | |
|---|---|
| df | a data frame |
| type | "T" for TRUE counts (default), "F" for FALSE counts |

### Value

a data frame with the number of TRUE or FALSE values in each column

### Examples

```
df <- data.frame(a = c(TRUE, FALSE, TRUE, TRUE), b = c(FALSE, TRUE, TRUE, TRUE))
across_df_TF(df) # Count TRUE (default)
across_df_TF(df, "F") # Count FALSE
```

## add_chrpos *Convert rsID to CHR & BP*

### Description

This function takes a dataset with a column containing rsIDs (SNP IDs) and adds the corresponding chromosome (CHR) and position (POS) information. It queries the SNPlocs.Hsapiens.dbSNP155.GRCh37 database (or GRCh38 if specified) to retrieve the genomic positions. The function returns a dataframe with the additional 'CHR' and 'POS' columns appended.

### Usage

```
add_chrpos(dat, snp_col = "SNP", ref = "GRCh37")
```

### Arguments

| | |
|---|---|
| dat | A dataframe containing at least a column with SNP IDs (rsIDs). |
| snp_col | A string indicating the column name containing SNP IDs (default is "SNP"). |
| ref | A string indicating the reference genome version. Default is "GRCh37", can also use "GRCh38". |

### Value

A dataframe with additional 'CHR' and 'POS' columns.

### Examples

```
## Not run:
pacman::p_load(data.table, dplyr, BSgenome, SNPlocs.Hsapiens.dbSNP155.GRCh37)
zuo_ref <- fread("/path/to/1KG-EAS-EAF.txt.gz") # Input dataset with rsID (SNP column)
result <- add_chrpos(zuo_ref, snp_col = "SNP", ref = "GRCh37")

## End(Not run)
```

## add_rsid *Convert CHR:BP to rsID (not recommended)*

### Description

This function takes a dataframe that must include columns 'CHR' and 'BP', and it appends the corresponding rsID by querying the SNPlocs.Hsapiens.dbSNP155.GRCh37 database. The function returns a dataframe with the rsIDs included.

### Usage

```
add_rsid(dat, ref = "GRCh37")
```

### Arguments

| | |
|---|---|
| dat | • "GRCh38" for GRCh38 Ref panel |
| ref | str indicating ref version. |
| | • "GRCh37" for GRCh37 Ref panel |

### Value

A dataframe with an additional 'RefSNP_id' column which contains the rsIDs.

### Examples

```
## Not run:
pacman::p_load(data.table, BSgenome, leo.gwas)
library("SNPlocs.Hsapiens.dbSNP155.GRCh37") # for GRCh37
library("SNPlocs.Hsapiens.dbSNP155.GRCh38") # for GRCh38
df <- data.frame(
  CHR = c(1, 1),
  BP = c(15211, 15820)
)
result <- add_rsid(df); result

## End(Not run)
```

---

## add_rsid_using_ref *Add rsID based on local reference file (recommended)*

---

### Description

This function takes a data frame with at least chromosome, position, and allele columns, and matches rsID based on a local reference file, considering reversed/complement alleles. It allows for flexibility in the names of the allele columns (e.g., 'REF'/'ALT' or 'A1'/'A2'). The function returns the original data frame with rsID added in the first column.

### Usage

```
add_rsid_using_ref(
  dat,
  local_ref,
  chr_col = "CHR",
  pos_col = "BP",
  a1_col = "A1",
  a2_col = "A2"
)
```

## Arguments

| | |
|---|---|
| dat | A data frame containing at least chromosome, position, and allele columns. |
| local_ref | A data frame containing at least 'ID' and 'SNP' columns. |
| chr_col | Name of the chromosome column in 'dat'. Default is 'CHR'. |
| pos_col | Name of the position column in 'dat'. Default is 'BP'. |
| a1_col | Name of the first allele column in 'dat'. Default is 'A1' (e.g., 'A1' or 'ALT'). |
| a2_col | Name of the second allele column in 'dat'. Default is 'A2' (e.g., 'A2' or 'REF'). |

## Value

The original data frame with an added 'rsID' column as the first column.

## Examples

```
## Not run:
library(dplyr)
# Example data with REF and ALT
dat <- data.frame(
  CHR = c(7, 12, 4),
  BP = c(6013153, 126890980, 40088896),
  REF = c("G", "A", "T"),
  ALT = c("A", "G", "A")
)
# Reference data
local_ref <- data.frame(
  ID = c("7:6013153:A:G", "12:126890980:G:A", "4:40088896:A:T"),
  SNP = c("rs10000", "rs1000000", "rs10000000")
)
result <- add_rsid_using_ref(dat, local_ref, a1_col = "ALT", a2_col = "REF")
print(result)

# Example data with A1 and A2
dat2 <- data.frame(
  CHR = c(7, 12, 4),
  POS = c(6013153, 126890980, 40088896),
  A1 = c("A", "G", "A"),
  A2 = c("G", "A", "T")
)
result2 <- add_rsid_using_ref(dat2, local_ref, pos_col = "POS")
print(result2)

## End(Not run)
```

`annotate_cpg_sites` *Annotate CpG Sites with Gene Information*

## Description

This function annotates a vector of CpG site probe IDs by retrieving corresponding gene names from the Illumina 450k annotation package. It returns a data frame with the original CpG sites and their associated gene names.

## Usage

```
annotate_cpg_sites(
  cpg_vector,
  annotation_package = "IlluminaHumanMethylation450kanno.ilmn12.hg19"
)
```

## Arguments

| | |
|---|---|
| `cpg_vector` | A character vector of CpG site probe IDs to be annotated. |
| `annotation_package` | A character string specifying the Illumina annotation package to use. It can be one of the following:<br>• "IlluminaHumanMethylation450kanno.ilmn12.hg19" (default)<br>• "IlluminaHumanMethylationEPICanno.ilm10b4.hg19" |

## Value

A data frame with two columns:

- `CpG_Site`: The original CpG site probe IDs.
- `Gene`: The associated gene names. `NA` if no gene annotation is found.

## Examples

```
## Not run:
library(IlluminaHumanMethylation450kanno.ilmn12.hg19)
library(IlluminaHumanMethylationEPICanno.ilm10b4.hg19)
# Example CpG probe IDs
cpg_ids <- c("cg00000029", "cg00000108", "cg00000109")
# Annotate CpG sites
annotate_cpg_sites(cpg_ids, "IlluminaHumanMethylation450kanno.ilmn12.hg19")

## End(Not run)
```

catboost_prs *CatBoost PRS utilities*

## Description

Train, apply, and rank CatBoost polygenic risk score (PRS) models.

## Usage

```
catboost_prs(a1_matrix, divide = FALSE, divide_ratio = 0.5)

catboost_prs_target(a1_matrix, model)

catboost_prs_rank(
  model,
  pool,
  pool_df,
  types = c("FeatureImportance", "ShapValues", "Interaction"),
  top_k = NULL
)
```

## Arguments

| | |
|---|---|
| a1_matrix | A1 matrix (with PHENOTYPE and ID columns). |
| divide | Logical; split into train/val sets. Default FALSE. |
| divide_ratio | Numeric; train proportion if divide=TRUE. Default 0.5. |
| model | Trained CatBoost model. |
| pool | CatBoost pool. |
| pool_df | Data used to build pool. |
| types | Character; any of "FeatureImportance","ShapValues","Interaction". |
| top_k | Integer; top features to keep. Default NULL. |

## Details

These functions require installed **catboost**. Training with divide = TRUE also requires **caret** for stratified data splitting.

## Value

Each function returns a list; contents depend on task (training, target prediction, ranking).

## See Also

catboost::catboost.train(), catboost::catboost.get_feature_importance()

check_significant_SNP *locate the significant SNP for conditional analysis*

## Description

locate the significant SNP for conditional analysis

## Usage

```
check_significant_SNP(x, environment.list, significance_level = 5e-08)
```

## Arguments

| | |
|---|---|
| x | data.frame of the SNP information |
| environment.list | tibble of the environment list (see the example for usage) |
| significance_level | numeric, significance level, default is 5e-8 |

## Value

message that informs the user of the independant SNP for the following conditional analysis

## Examples

```
## Not run:
# specify the directory to store the HLA original data and subsequent
con_dir <- "/Users/leoarrow/project/VKH2024/data/zuo/con_su"
files <- list.files(con_dir,full.names = T) %>% as.vector(); files # update it each time
# and sort it by CHISQ/P value
x1 <- fread(files[1]) %>% arrange(desc(CHISQ)); head(x1) # for the 1st, just read the data
x2 <- fread(files[2]) %>% arrange(P) # repeat until no more independent signal
x3 <- fread(files[3]) %>% arrange(P)
x4 <- fread(files[4]) %>% arrange(P)
x5 <- fread(files[5]) %>% arrange(P)
x6 <- fread(files[6]) %>% arrange(P)
x7 <- fread(files[7]) %>% arrange(P)
x8 <- fread(files[8]) %>% arrange(P)

env <- ls() # get the environment; this line if were put in main func will lead to error.
# load the environment
environment.list <- tibble(item = as.vector(grep("^x[0-9]+$", x = env, value = TRUE)))
check_significant_SNP(x8, environment.list, significance_level = 5e-8)

# You can mannually check the p-value of one SNP in previous environment.list

## End(Not run)
```

## chisq_p_value *Give precise p-value for chi-square test*

### Description

Sometimes you get a p-value of 0 when you perform a chi-square test or other analysis. This is because the p-value is so small that R rounds it to 0. This function gives you a more precise p-value.

### Usage

```
chisq_p_value(chisq_value, df, digits = 4, prec = 100)
```

### Arguments

| | |
|---|---|
| chisq_value | numeric, chi-square value |
| df | numeric, degree of freedom |
| digits | numeric, digits for output to illustrate |
| prec | numeric, precision for mpfr() function |

### Value

A precise p-value in scientific format

### Examples

```
## Not run:
# install.packages("Rmpfr")
library(Rmpfr)
chisq_value <- 2629; df <- 1
p_value <- chisq_p_value(chisq_value, df)
print(p_value)

## End(Not run)
```

## clump_data_local *Perform LD Clumping Locally or via Reference Panel*

### Description

This function performs LD clumping using either a local PLINK binary file (bfile) or a 1000 Genomes super population panel. Requires the ieugwasr and plinkbinr packages.

## Usage

```
clump_data_local(
  dat,
  pop = NULL,
  bfile = NULL,
  clump_kb = 10000,
  clump_r2 = 0.001,
  plink_bin = plinkbinr::get_plink_exe()
)
```

## Arguments

| | |
|---|---|
| dat | Data frame with columns SNP, pval.exposure; optional id.exposure. |
| pop | 1000G super-pop for online clumping (AFR/AMR/EAS/EUR/SAS). Used when bfile is NULL. |
| bfile | PLINK reference panel prefix for local clumping (without .bed/.bim/.fam). If set, pop is ignored. |
| clump_kb | Window size in kb. Default 10000. |
| clump_r2 | r^2 threshold. Default 0.001. |
| plink_bin | Path to PLINK executable (auto-detect via plinkbinr if NULL and bfile is set). |

## Examples

```
## Not run:
# Reference:
# - https://github.com/MRCIEU/TwoSampleMR/issues/173
# - https://blog.csdn.net/xiaozheng1213/article/details/126269969
library(ieugwasr)
library(plinkbinr) # devtools::install_github("explodecomputer/plinkbinr")
plinkbinr::get_plink_exe()

# Note: after using this, please check `leo_clump` column to see if they are all TRUE
# If it's contains F, it means no SNPs remained after clumping or something bad happened

## End(Not run)
```

---

# combine_smr_res_1outcome

*Combine* .fdr *files for one or multiple outcomes (seperately)*

---

## Description

Combine .fdr files for one or multiple outcomes (seperately)

## Usage

```
combine_smr_res_1outcome(
  dir,
  outcome,
  out_dir = file.path(dir, "combine_1outcome")
)
```

## Arguments

| | |
|---|---|
| dir | Character. The parent folder that contains subfolders with fdr files. |
| outcome | Character or character vector. One or more outcomes to search in file names. |
| out_dir | Character. Output folder; default is to create "combine_1outcome" under dir. |

## Value

NULL. This function writes combined files to disk directly.

## Examples

```
## Not run:
combine_smr_res_1outcome("/Users/leoarrow/project/iridocyclitis/output/smr", "iri3")

## End(Not run)
```

---

# combine_smr_res_chr *Combine SMR Results for All Chromosomes*

---

## Description

This function combines SMR (Summary-data-based Mendelian Randomization) result files across all chromosomes for each unique exposure and outcome pair (Yes, we can deal with multiple exposure in one dir (for say, the 49 sqtl from GTEx)). The combined results are saved to a specified output directory.

## Usage

```
combine_smr_res_chr(dir, out_dir = "")
```

## Arguments

| | |
|---|---|
| dir | Character. The directory containing SMR result files. Files should follow the naming convention exposure_chrX@outcome.smr, where X represents the chromosome number. |
| out_dir | Character. The output directory where combined SMR files will be saved. If not specified, defaults to a subdirectory named chr_combined within dir. |

## Examples

```
## Not run:
# Combine SMR results in the "data/smr_results" directory and save to default output directory
combine_smr_res_chr(dir = "data/smr_results")

# Combine SMR results and specify a custom output directory
combine_smr_res_chr(dir = "data/smr_results", out_dir = "data/combined_results")

## End(Not run)
```

---

# correlation_calculate *Calculate Correlation between Two Vectors*

---

## Description

This function calculates the Spearman (default) or Pearson correlation coefficient and its associated p-value between two vectors. It automatically handles missing values.

## Usage

```
correlation_calculate(vector_x, vector_y, method = "spearman", ...)
```

## Arguments

| | |
|---|---|
| vector_x | A numeric vector. |
| vector_y | A numeric vector of the same length as vector_x. |
| method | A character string specifying the correlation method ("spearman" or "pearson"). Defaults to "spearman". |
| ... | Pass to cor.test. |

## Value

A list with the correlation coefficient and p-value.

## See Also

correlation_draw for plotting correlation results.

## Examples

```
vector_x <- c(1, 2, 2, 4, 5)
vector_y <- c(5, 6, 7, 8, 7)
result <- correlation_calculate(vector_x, vector_y, method = "pearson")
```

`correlation_draw` *Draw Correlation between Two Vectors*

## Description

This function creates a scatter plot to visualize the correlation between two vectors, displaying the correlation coefficient and p-value on the plot.

## Usage

```
correlation_draw(
  vector_x,
  vector_y,
  method = "spearman",
  color_palette = "npg",
  title = "Correlation Plot",
  xlab = "Vector X",
  ylab = "Vector Y",
  point_size = 1.5,
  point_color = "#BB7CD8",
  point_stroke = 1,
  alpha = 0.75,
  line_color = "#BB7CD8",
  line_type = "dashed",
  line_size = 1.2,
  ci_alpha = 0.2,
  title_size = 16,
  xlab_size = 14,
  ylab_size = 14,
  axis_text_size = 14,
  ...
)
```

## Arguments

| | |
|---|---|
| `vector_x` | Numeric vector. |
| `vector_y` | Numeric vector of the same length as `vector_x`. |
| `method` | Correlation method: `"spearman"` or `"pearson"`. Default `"spearman"`. |
| `color_palette` | Character scalar or vector for color palette (kept for compatibility). |
| `title` | Plot title. Default `"Correlation Plot"`. |
| `xlab, ylab` | Axis labels. Defaults `"Vector X"`, `"Vector Y"`. |
| `point_size` | Point size. Default `1.5`. |
| `point_color` | Fill color for points (shape 21). Default `"#BB7CD8"`. |
| `point_stroke` | Numeric stroke width for point outline. If `NA`, treated as `0`. Default `1`. |

| | |
|---|---|
| alpha | Point transparency. Default 0.75. |
| line_color, line_type, line_size | Trend line color, type, size. Defaults "#BB7CD8", "dashed", 1.2. |
| ci_alpha | Confidence ribbon alpha. Default 0.2. |
| title_size, xlab_size, ylab_size, axis_text_size | Font sizes. Defaults 16, 14, 14, 14. |
| ... | Additional arguments passed to correlation_calculate(). |

## Value

A **ggplot2** object.

## See Also

correlation_calculate, leo_scale_color

## Examples

```
vector_x <- c(10, 2, 3, 4, 5)
vector_y <- c(5, 6, 7, 8, 7)
correlation_draw(vector_x, vector_y, method = "pearson", point_size = 10, color_palette = "npg")
```

---

# count_duplicate_element

*Count or Identify Duplicates in a Vector*

---

## Description

This function counts the number of duplicate elements in a vector, or returns a logical vector indicating which elements are duplicates.

## Usage

```
count_duplicate_element(vector, return = "count")
```

## Arguments

| | |
|---|---|
| vector | A vector to be checked for duplicates. |
| return | A string specifying the return type: "count" for number of duplicates, or "logi" for a logical vector. |

## Value

An integer representing the number of duplicates if return is "count", or a logical vector indicating which elements are duplicates if return is "logi".

## Examples

```
vec <- c("a", "b", "c", "a", "b", "d")
count_duplicate_element(vec, return = "count")  # Returns 4
count_duplicate_element(vec, return = "logi")  # Returns c(TRUE, TRUE, FALSE, TRUE, TRUE, FALSE)
```

---

`count_matching_elements`

*Count or Identify Matches of a Pattern in a Vector*

---

## Description

This function counts the number of elements in a vector that contain a given pattern, or returns a logical vector indicating which elements match the pattern.

## Usage

```
count_matching_elements(vector, pattern, return = "count")
```

## Arguments

| | |
|---|---|
| `vector` | A character vector to be searched. |
| `pattern` | The pattern to match (regular expression). |
| `return` | A string specifying the return type: "count" for number of matches, or "logi" for a logical vector. |

## Value

An integer representing the number of matches if return is "count", or a logical vector indicating which elements match the pattern if return is "logi".

## Examples

```
vec <- c("abc", "def", "xyz", "abcd")
count_matching_elements(vec, "abc", return = "count")  # Returns 2
count_matching_elements(vec, "abc", return = "logi")  # Returns c(TRUE, FALSE, FALSE, TRUE)
```

csMR_env *Configure Conda Environment for csMR*

## Description

csMR_env() prepares the official csMR environment. It clones the official repository if missing, tries envs/envpy3.yml first, falls back to a temporary compatibility env file only when the official file cannot solve, installs required R packages into the csMR conda library, checks PLINK 1.90, and returns the runtime paths required by the csMR README.

## Usage

```
csMR_env(
  repo_dir = "~/Project/software/csMR",
  repo_url = "https://github.com/rhhao/csMR.git",
  env_name = "csMR",
  conda = Sys.which("conda"),
  mamba = Sys.which("mamba"),
  plink_path = Sys.which("plink"),
  ref = "main",
  overwrite = FALSE
)
```

## Arguments

| | |
|---|---|
| repo_dir | Path to the csMR repository. |
| repo_url | Official csMR GitHub URL. |
| env_name | Conda environment name. |
| conda | Path to conda. |
| mamba | Path to mamba. |
| plink_path | Optional PLINK 1.90 path. If empty, PATH and plinkbinr::get_plink_exe() are tried. |
| ref | Git branch or tag. |
| overwrite | Whether to rebuild an existing env. |

## Value

A list with repo_dir, env_name, env_prefix, env_file, r_home, r_library, and plink.

## Examples

```
## Not run:
cfg <- csMR_env()
cfg$r_library

## End(Not run)
```

`csMR_step1_prep` *Prepare csMR Step1 Input*

## Description

Convert GWAS or QTL summary statistics to the csMR-required `.ma` format using explicit column mappings. This function only performs thin formatting and basic QC; it does not guess genome build or map non-rsID SNPs.

## Usage

```
csMR_step1_prep(
  data,
  type = c("gwas", "qtl"),
  output,
  SNP_col = "SNP",
  GENE_col = "GENE",
  A1_col = "A1",
  A2_col = "A2",
  MAF_col = "MAF",
  BETA_col = "BETA",
  SE_col = "SE",
  P_col = "P",
  N_col = "N",
  n = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | GWAS/QTL data.frame, file path, file vector, or a QTL directory. |
| `type` | `"gwas"` or `"qtl"`. |
| `output` | Output `.ma` file path for GWAS or single-table QTL, or output directory for multi-file QTL input. |
| `SNP_col` | Input SNP column name. |
| `GENE_col` | Input gene column name for QTL. |
| `A1_col` | Input effect allele column name. |
| `A2_col` | Input other allele column name. |
| `MAF_col` | Input MAF column name. |
| `BETA_col` | Input beta column name. |
| `SE_col` | Input SE column name. |
| `P_col` | Input P column name. |
| `N_col` | Input sample size column name. |
| `n` | Optional fixed sample size used to fill missing `N` in GWAS. |

## Value

For GWAS, a list with `output`, `n_input`, `n_output`, and `n_missing_filled`. For QTL, a manifest data.frame with `id`, `input`, `output`, `n_input`, and `n_output`.

## Examples

```
## Not run:
csMR_step1_prep(
  data = "~/Project/iridocyclitis/data/diabete/1/GCST90014023_buildhg19.tsv",
  type = "gwas",
  output = "~/Project/iridocyclitis/output/csMR/step1/exposure.ma",
  SNP_col = "rsID",
  A1_col = "effect_allele",
  A2_col = "other_allele",
  MAF_col = "EAF",
  BETA_col = "beta",
  SE_col = "se",
  P_col = "pval",
  N_col = "N"
)

## End(Not run)
```

---

# `csMR_step2_config.yml` *Build csMR config.yml*

---

## Description

Write an official-style csMR `config.yml` for Step 3.

## Usage

```
csMR_step2_config.yml(
  repo_dir = "~/Project/software/csMR",
  config.yml_to = "./output/csMR/step2_config/config.yml",
  BASE_OUTPUT_DIR = "./output/csMR/step3_run",
  qtl_input_dir = "./output/csMR/step1_data_preparation/sc_qtl_dir1",
  exposure_ma = "./output/csMR/step1_data_preparation/gwas_exp_p1.ma",
  exposure_id = "exp_p1",
  exposure_type = "cc",
  exposure_ma_dir = NULL,
  outcome_ma_dir = "./output/csMR/step1_data_preparation/outcome",
  GWAS_REFERENCE_GENOTYPE = NULL,
  eQTL_REFERENCE_GENOTYPE = NULL,
  duplicated_snp_path = "None",
  coloc_window_size_bp = 100000L,
  coloc_coverages = 0.9,
  coloc_threads = 5L,
```

```
  coloc_cutoff = 0.8
)
```

## Arguments

| | |
|---|---|
| repo_dir | csMR repository directory. |
| config.yml_to | Output config file path. |
| BASE_OUTPUT_DIR | Output root used by csMR Step 3. |
| qtl_input_dir | Directory of QTL .ma files (or any input supported by .csmr_qtl_paths()). |
| exposure_ma | Exposure GWAS .ma file path. |
| exposure_id | Exposure id in GWAS_SUMSTATS. |
| exposure_type | Exposure type, either "cc" or "quant". |
| exposure_ma_dir | Optional directory of multiple exposure .ma files. If provided, exposure_ma is ignored and ids are generated from file names. |
| outcome_ma_dir | Outcome .ma directory (used as OUTCOME_DIR). Must contain **only** .ma files — Snakemake's os.listdir() picks up every item (incl. .DS_Store, subdirectories) as an outcome wildcard. |
| GWAS_REFERENCE_GENOTYPE | GWAS reference genotype directory. If NULL, default to repo_dir/data/reference_genome_1000G_EUR |
| eQTL_REFERENCE_GENOTYPE | eQTL reference genotype directory. If NULL, default to repo_dir/data/reference_genome_1000G_EUR. |
| duplicated_snp_path | Path to duplicated SNP file for the reference genotype. Use "None" if not applicable (official csMR default). |
| coloc_window_size_bp | Coloc window size. |
| coloc_coverages | Coloc coverage. |
| coloc_threads | Coloc threads. |
| coloc_cutoff | Coloc cutoff. |

## Details

Uppercase arguments such as BASE_OUTPUT_DIR, GWAS_REFERENCE_GENOTYPE, and eQTL_REFERENCE_GENOTYPE intentionally mirror official csMR config.yml keys to keep mapping explicit and reduce confusion when debugging Step 3.

## Value

Absolute path to the written config file.

## Examples

```
## Not run:
out_cfg <- csMR_step2_config.yml(
  repo_dir = "~/Project/software/csMR",
  config.yml_to = "./output/csMR/step2_config/config.yml",
  BASE_OUTPUT_DIR = "./output/csMR/step3_run", # where to store the final csMR output
  qtl_input_dir = "./output/csMR/step1_data_preparation/sc_qtl_dir1",
  exposure_ma = "./output/csMR/step1_data_preparation/gwas_exp_p1.ma",
  exposure_id = "exp_p1",
  exposure_type = "cc",
  outcome_ma_dir = "./output/csMR/step1_data_preparation/outcome"
)
# Example console messages:
# i [23:36:24] Writing csMR config.yml ...
# v [23:36:24] csMR step2 config written: ./output/csMR/step2_config/config.yml
out_cfg
# [1] "/Users/leoarrow/Project/iridocyclitis/output/csMR/step2_config/config.yml"

## End(Not run)
```

---

`csMR_step3_run` *Run csMR Step 3*

---

## Description

Run the csMR Snakemake workflow with `R_HOME` and `R_LIBS` explicitly set from the target csMR conda environment.

## Usage

```
csMR_step3_run(
  repo_dir = "~/Project/software/csMR",
  config_file = "./output/csMR/step2_config/config.yml",
  jobs = NULL,
  forcerun = NULL,
  work_flow.snakefile = NULL,
  env_name = "csMR",
  conda = Sys.which("conda"),
  dry_run = FALSE,
  log_file = NULL
)
```

## Arguments

| | |
|---|---|
| `repo_dir` | csMR repository directory. |
| `config_file` | Config file path. |
| `jobs` | Number of Snakemake jobs. If `NULL`, run with bare `-j` and let Snakemake use all available cores. |

| | |
|---|---|
| forcerun | Optional Snakemake targets/rules to force-run. Default NULL means no --forcerun is added. |
| work_flow.snakefile | Snakemake workflow file path. If NULL, use file.path(repo_dir, "work_flow.snakefile"). |
| env_name | Conda env name. |
| conda | Path to conda. |
| dry_run | Whether to run --dry-run. |
| log_file | Optional log file path. |

## Details

The first four arguments (repo_dir, config_file, jobs, forcerun) are the most commonly adjusted by users in routine runs.

## Value

A list with command (copyable shell command), status, log_file, r_lib, and r_home. Output streams to console in real time and is also saved to log_file.

## Examples

```
## Not run:
csMR_step3_run(repo_dir = "~/Project/software/csMR", jobs = NULL, dry_run = TRUE)

## End(Not run)
```

---

# dr.prs *DuoRank PRS (Dr.PRS)*

---

## Description

Dr.PRS is designed to rank the importance for PRS inputs and futher optimazation. It takes two none-overlap stage plink file (bfile) for machine learning modeling:

- lasso (to capture linear relationship)
- catboost (to capture non-linear relationship) It also calculated PRS with plink using traditional additive model.

## Usage

```
dr.prs(
  stage1_bfile = "./data/zuo/bed/SNP_for_PRS/VKH-zhonghua-for_prs_snplist",
  stage2_bfile = "./data/zuo/bed/SNP_for_PRS/VKH-ASA-for_prs_snplist",
  summary_file = "./output/part1-gwas/prs/snp_for_prs.txt",
  output_dir = "./output/part1-gwas/prs/b1t2",
  method = c(1, 2, 3),
  dr_optimazation = TRUE,
```

```
  snp_col = "SNP.meta",
  a1_col = "A1",
  weight_col = "OR_Final",
  weight_type = "OR",
  divide_ratio = 0.7,
  iLasso_iteration = 1000,
  nfolds = 10,
  plink_bin = plinkbinr::get_plink_exe(),
  clump = T,
  clump_include_hla = F,
  clump_param = NULL,
  seed = 725
)

combine_rank(rank1, rank2, auc1 = NULL, auc2 = NULL)
```

## Arguments

| | |
|---|---|
| stage1_bfile | Path to the stage1 PLINK binary files. Normally it is the one generates summary data. If only 1 source of individual data is available, you can split it into 2 non-overlap datasets. |
| stage2_bfile | Path to the stage2 PLINK binary files, i.e., the target dataset for PRS calculation. |
| summary_file | Path to the summary statistics file or a data frame containing SNPs, alleles, and weights. Note this summary only contains refined SNP for PRS calculation. |
| output_dir | Directory to save output files. |
| method | Integer vector indicating which methods to use: 1=PLINK, 2=CatBoost, 3=iLasso. Default is all three methods. |
| dr_optimazation | Logical, whether to perform greedy search to optimize PRS based on combined rank from CatBoost and iLasso (default: TRUE). |
| snp_col | Column name in the summary statistics for SNP IDs (default: "SNP.meta"). |
| a1_col | Column name in the summary statistics for effect allele (default: "A1"). |
| weight_col | Column name in the summary statistics for weights (default: "OR_Final"). |
| weight_type | Type of weights, either "OR" (default) or "Beta". If "OR", it will calculate Beta values. |
| divide_ratio | Proportion of stage1 data in iLasso (default: 0.7). |
| iLasso_iteration | Number of iterations for iLasso model (default: 1000). |
| nfolds | Number of folds for cross-validation (default: 10). |
| plink_bin | Path to the PLINK binary executable (default: `plinkbinr::get_plink_exe()`). |
| clump | Place holder for future clumping function. |
| clump_include_hla | Place holder for future clumping function. |
| clump_param | Place holder for future clumping function. |

| | |
|---|---|
| seed | Random seed (Default: 725). |
| rank1 | rank1 |
| rank2 | rank2 |
| auc1 | auc1 |
| auc2 | auc2 |

## Value

A list with:

- lasso_importance, catboost_importance, combined_rank
- greedy_auc_curve (data.frame: kept_snps, auc)
- final_subset (character vector of SNPs)
- plink_prs_file (path), logs

---

`extract_instruments_local` *Extract instruments locally for MR Analysis*

---

## Description

Filters SNPs by p-value threshold and performs LD clumping with flexible column mapping.

## Usage

```
extract_instruments_local(
  dat,
  p = 5e-08,
  pop = NULL,
  phenotype_col = "Phenotype",
  snp_col = "SNP",
  chr_col = "CHR",
  pos_col = "POS",
  effect_allele_col = "A1",
  other_allele_col = "A2",
  beta_col = "BETA",
  se_col = "SE",
  pval_col = "P",
  eaf_col = "EAF",
  N = "Neff",
  bfile = NULL,
  plink_bin = NULL
)
```

## Arguments

| | |
|---|---|
| dat | Dataframe containing GWAS summary statistics; change it to dataframe if it is data.table. |
| p | P-value cutoff (default = 5e-8) |
| pop | Super-population code for LD reference (default = NULL, as we recommend using loca bfile) |
| phenotype_col | Column name for phenotype (default = "Phenotype") |
| snp_col | Column name for SNP IDs (default = "SNP") |
| chr_col | Column name for chromosome (default = "CHR") |
| pos_col | Column name for position (default = "POS") |
| effect_allele_col | Column name for effect allele (default = "A1") |
| other_allele_col | Column name for non-effect allele (default = "A2") |
| beta_col | Column name for effect size (default = "BETA") |
| se_col | Column name for standard error (default = "SE") |
| pval_col | Column name for p-values (default = "P") |
| eaf_col | Column name for effect allele frequency (default = "EAF") |
| N | Column name for sample size (default = "Neff", set NULL to exclude) |
| bfile | Path to PLINK binary reference panel (default = NULL) |
| plink_bin | Path to PLINK executable (default = NULL) |

## Value

Formatted exposure data ready for MR analysis

## Examples

```
## Not run:
# Custom column names example
extract_instruments_local(
  dat = gwas_data,
  phenotype_col = "Trait",
  snp_col = "rsID",
  chr_col = "Chromosome",
  pos_col = "Position"
)

## End(Not run)
```

filter_chr_basedonSNP_p *Filter Chromosomes Based on SNP P-value Threshold*

## Description

This function filters chromosomes out if no SNP within the chromosome meets threshold. That is, if a chromosome has at least one SNP that meets the threshold, all SNPs on that chromosome are retained.

## Usage

```
filter_chr_basedonSNP_p(
  df,
  chr_col = "CHR",
  snp_col = "Variant_ID",
  p_val_col = "nominal_P_value",
  threshold = 0.00157
)
```

## Arguments

| | |
|---|---|
| df | A data frame containing SNP data. |
| chr_col | Character string specifying the name of the chromosome column. Default is "CHR". |
| snp_col | Character string specifying the name of the SNP identifier column. Default is "Variant_ID". |
| p_val_col | Character string specifying the name of the p-value column. Default is "nominal_P_value". |
| threshold | Numeric value specifying the p-value threshold. Default is 1.57e-3 (Thresh hold for the HEIDI test). |

## Value

A filtered data frame .

## Examples

```
library(dplyr)
eqtl_data <- data.frame(
CHR = c("1", "1", "2", "2", "3"),
Variant_ID = c("rs1", "rs2", "rs3", "rs4", "rs5"),
nominal_P_value = c(1e-9, 0.05, 0.2, 1e-7, 0.3)
);eqtl_data
df_filtered <- filter_chr_basedonSNP_p(
  df = eqtl_data,
  chr_col = "CHR",
  snp_col = "Variant_ID",
```

```
  p_val_col = "nominal_P_value",
  threshold = 5e-8
);df_filtered
```

---

`filter_chr_basedonSNP_p_qtltools` *Filter Chromosomes Based on SNP P-value Threshold for* `.qtltoolsnomi` *files*

---

## Description

This function adds an index column to the input data frame and filters chromosomes based on whether any SNP within the chromosome crosses a specified threshold. If a chromosome has at least one SNP that meets the threshold, all SNPs on that chromosome are retained. Otherwise, all SNPs on that chromosome are removed.

## Usage

```
filter_chr_basedonSNP_p_qtltools(
  df,
  chr_col = "CHR",
  snp_col = "Variant_ID",
  gene_col = "Gene",
  p_val_col = "nominal_P_value",
  threshold = 0.00157
)
```

## Arguments

| | |
|---|---|
| `df` | A data frame containing SNP data. |
| `chr_col` | Character string specifying the name of the chromosome column. Default is `"CHR"`. |
| `snp_col` | Character string specifying the name of the SNP identifier column. Default is `"Variant_ID"`. |
| `gene_col` | Character string specifying the name of the gene column. Default is `"Gene"`. |
| `p_val_col` | Character string specifying the name of the p-value column. Default is `"nominal_P_value"`. |
| `threshold` | Numeric value specifying the p-value threshold. Default is `1.57e-3` (Thresh hold for the HEIDI test). |

## Value

A filtered data frame with an added `index` column.

## Examples

```
library(dplyr)
eqtl_data <- data.frame(
Gene = c("G1", "G1", "G2", "G2", "G3"),
CHR = c("1", "1", "2", "2", "3"),
Variant_ID = c("rs1", "rs2", "rs3", "rs4", "rs5"),
nominal_P_value = c(1e-9, 0.05, 0.2, 1e-7, 0.3)
);eqtl_data
df_filtered <- filter_chr_basedonSNP_p_qtltools(
  df = eqtl_data,
  chr_col = "CHR",
  snp_col = "Variant_ID",
  p_val_col = "nominal_P_value",
  threshold = 5e-8
);df_filtered
```

---

`find_proxy` *find_proxy*

---

## Description

`find_proxy` finds the proxy snp for the miss iv

## Usage

```
find_proxy(
  miss_iv,
  miss_snp,
  outcome_snp,
  proxy_file = NULL,
  proxy_output_path = NULL,
  pop = "EUR",
  gb = "grch38",
  token = ""
)
```

## Arguments

| | |
|---|---|
| `miss_iv` | iv datafram in tsmr exposure format, which can not locate snp in the outcome |
| `miss_snp` | snp in the miss_iv; can be inferred using miss_iv$SNP |
| `outcome_snp` | a str vector containing all snp in the outcome; this NOT the entire outcome gwas summary statistics |
| `proxy_file` | pre-calculated proxy file path (full); do provide this if the proxy file is already generated !!! |
| `proxy_output_path` | a full path to save the proxy file when using ldlink |

| | |
|---|---|
| pop | reference panel from 1kg (LDlinkR param) |
| gb | genome build (LDlinkR param) |
| token | token of LDlinkR |

## Value

a updated missiv with proxy.snp proxy.effect.allele proxy.other.allele r2 col

## Examples

```
# This function can be used when many iv can not locate corresponding snp in the outcome
# in tsmr analysis
## Not run:
# iv is estracted iv via tsmr package;dat_h is a standard output of harmonise_data()
miss_iv <- iv[!iv$SNP %in% dat_h$SNP,]
miss_snp <- miss_iv$SNP
outcome_snp <- iri_nc$SNP
proxy_output_path <- "Full path to where you wanna store the LDlinkR output"
 proxy_iv <- find_proxy(miss_iv, miss_snp, outcome_snp,
                        proxy_file = "./combined_query_snp_list_grch38.txt",
                        proxy_output_path = NULL)
 # bak
 proxy_iv$target.snp <- proxy_iv$SNP # target snp
 proxy_iv$target.A1 <- proxy_iv$effect_allele.exposure
 proxy_iv$target.A2 <- proxy_iv$other_allele.exposure
 # replace for tsmr
 proxy_iv$SNP <- proxy_iv$proxy.snp
 proxy_iv$effect_allele.exposure <- proxy_iv$proxy.A1
 proxy_iv$other_allele.exposure <- proxy_iv$proxy.A2
 iv_f <- bind_rows(non_miss_iv, proxy_iv) # f for final
 dat_h_proxy <- harmonise_data(iv_f, out_nc_proxy)
 mr(dat_h_proxy) # nailed it!

## End(Not run)
```

---

# format_outcome *format outcome data*

---

## Description

format outcome data

## Usage

```
format_outcome(dat, snp = iv$SNP, N = "Neff")
```

### Arguments

| | |
|---|---|
| dat | a dataframe for outcome with SNP, CHR, POS, A1, A2, EAF, BETA, SE, P, Phenotype, samplesize columns |
| snp | a str vector out of iv$SNP |
| N | N column name for sample size (effective or observed) |

### Value

a tsmr format outcome dataframe

---

## get_id *Get Unique Identifier for Genetic Data*

---

### Description

Get ID using CHR, BP, A2 (REF/Non-effect), A1 (ALT/Effect)

### Usage

```
get_id(x, count_A1_A2 = FALSE)
```

### Arguments

| | |
|---|---|
| x | A data.frame that must contain the columns CHR, BP, A2, and A1. Each column represents:<br>• CHR: Chromosome (It can be any in c("chrom", "CHR", "Chromosome", "chromosome", "Chr"))<br>• BP/POS: Base pair position (It can be any in c("pos", "POS", "position", "BP", "Position", "Bp"))<br>• A2: Reference allele/non-effect allele (It can be any in c("A2", "Allele2", "allele2", "a2", "REF", "Ref", "ref", "Non-effect"))<br>• A1: Alternative allele/effect allele (It can be any in c("A1", "Allele1", "allele1", "a1", "ALT", "Alt", "alt", "Effect")) |
| count_A1_A2 | If TRUE, will count the number of characters in A1 and A2 |

### Value

A data.frame with an additional 'ID' column (if count_A1_A2=TRUE, containing unique identifiers and character counts of A1 and A2)

### Examples

```
df <- data.frame(chrom = c(1, 1, 2), pos = c(12345, 54321, 11111),
                 A1 = c("A", "T", "G"), A2 = c("G", "C", "A"))
get_id(df); get_id(df, count_A1_A2 = TRUE)
```

`group_comparison_draw` *Draw Group Comparison for a Continuous Variable*

## Description

This function creates a violin + boxplot to visualize the distribution of a continuous variable across groups of a categorical variable, with optional jitter points and annotated p-value.

## Usage

```
group_comparison_draw(
  df,
  x_col,
  y_col,
  vector_x = NULL,
  vector_y = NULL,
  test_method = "wilcox",
  title = "Group Comparison",
  xlab = NULL,
  ylab = NULL,
  alpha = 0.8,
  main_color = "#BB7CD8",
  violin_fill = main_color,
  box_fill = main_color,
  box_color = "black",
  jitter = T,
  jitter_size = 2,
  jitter_color = "black",
  drop_na = F,
  annotate_n = T,
  title_size = 16,
  xlab_size = 14,
  ylab_size = 14,
  axis_text_size = 14
)
```

## Arguments

| | |
|---|---|
| `df` | A data frame (optional if vectors provided). |
| `x_col` | Name of categorical variable in df. |
| `y_col` | Name of numeric variable in df. |
| `vector_x` | Optional categorical vector. |
| `vector_y` | Optional numeric vector. |
| `test_method` | Statistical test: `"wilcox"` (default, Mann–Whitney U test) or `"t.test"`. |
| `title` | Plot title. |

| | |
|---|---|
| xlab, ylab | Axis labels. |
| alpha | Transparency for violin, boxplot and jitter. |
| main_color | Easy way to set the vibe. Good luck trying! |
| violin_fill | Fill color for violin. |
| box_fill | Fill color for boxplot. |
| box_color | Outline color for boxplot. |
| jitter | Logical, whether to show jitter. Default TRUE. |
| jitter_size | Size of jitter points. Default 2. |
| jitter_color | Color of jitter points. Default "black". |
| drop_na | Logical, whether to drop NA group from x variable. Default FALSE. |
| annotate_n | Logical, whether to annotate sample size N for each group. Default TRUE. |
| title_size | Font size for plot title. Default 16. |
| xlab_size | Font size for x-axis label. Default 14. |
| ylab_size | Font size for y-axis label. Default 14. |
| axis_text_size | Font size for axis text. Default 14. |

## Value

A ggplot2 object.

## Examples

```
df <- tibble::tibble(group = rep(c(0,1), each=50), prs = rnorm(100))
group_comparison_draw(df, "group", "prs")
group_comparison_draw(df, "group", "prs", annotate_n = TRUE)
group_comparison_draw(vector_x = rep(c(0,1), each=50), vector_y = rnorm(100))
group_comparison_draw(vector_x = rep(c(0,1), each=50), vector_y = rnorm(100), jitter = FALSE)
```

---

# HLA_exclude *Exclude HLA region from genomic summary data*

---

## Description

This function filters out entries within the HLA region on a specified chromosome and position range. The default HLA region is set to chromosome 6, between 25Mb and 34Mb. Custom chromosome and position bounds can be specified.

## Usage

```
HLA_exclude(
  data,
  chromosome_col = "CHR",
  position_col = "BP",
  lower_bound = 2.5e+07,
  upper_bound = 3.4e+07
)
```

### Arguments

| | |
|---|---|
| data | A data frame containing genomic data. |
| chromosome_col | The name of the column representing chromosome numbers (default is CHR). |
| position_col | The name of the column representing genomic positions (default is BP). |
| lower_bound | The lower boundary of the HLA region in base pairs (default is 25e6). |
| upper_bound | The upper boundary of the HLA region in base pairs (default is 34e6). |

### Value

A data frame excluding rows that fall within the specified HLA region.

### Examples

```
example_data <- data.frame(
  SNP = c("rs1", "rs2", "rs3", "rs4"),
  CHR = c(6, 6, 6, 7),
  POS = c(26000000, 33000000, 35000000, 29000000)
)
result_data <- HLA_exclude(example_data, chromosome_col="CHR", position_col="POS")
print(result_data)
```

---

## HLA_get *Extract HLA region from genomic summary data*

---

### Description

This function extracts entries within the HLA region on a specified chromosome and position range. The default HLA region is chromosome 6, between 25Mb and 34Mb. Users may provide custom chromosome and region boundaries.

### Usage

```
HLA_get(
  data,
  chromosome_col = "CHR",
  position_col = "BP",
  lower_bound = 2.5e+07,
  upper_bound = 3.4e+07
)
```

### Arguments

| | |
|---|---|
| data | A data frame containing genomic data. |
| chromosome_col | The name of the column representing chromosome numbers (default is CHR). |
| position_col | The name of the column representing genomic positions (default is BP). |
| lower_bound | The lower boundary of the HLA region in base pairs (default is 25e6). |
| upper_bound | The upper boundary of the HLA region in base pairs (default is 34e6). |

## Value

A data frame including only rows within the specified HLA region.

## Examples

```
example_data <- data.frame(
  SNP = c("rs1", "rs2", "rs3", "rs4"),
  CHR = c(6, 6, 6, 7),
  POS = c(26000000, 33000000, 35000000, 29000000)
)
hla_data <- HLA_get(example_data, chromosome_col="CHR", position_col="POS")
print(hla_data)
```

---

# lasso_prs *Iterative Lasso PRS (iLasso) utilities*

---

## Description

Train, apply and visualize iterative Lasso-based PRS models with success gating.

## Usage

```
lasso_prs(
  a1_matrix,
  divide_ratio = 0.7,
  iterative = 100,
  nfolds = 10,
  score_type = "link",
  seed = 725
)

lasso_prs_target(a1_matrix, model, lambda, score_type = "link")

lasso_prs_rank(model, rank, auc_history)
```

## Arguments

| | |
|---|---|
| a1_matrix | data.frame; PLINK A1 matrix. |
| divide_ratio | numeric; train fraction per iteration (default 0.7). |
| iterative | integer; number of iterations (default 100) - If set to 1, then regular lasso is performed. |
| nfolds | integer; CV folds for glmnet (default 10). |
| score_type | character; "link" (linear score, default) or "response" (probability). |
| seed | integer; base random seed (default 725). |
| model | Trained glmnet model (for target/rank functions). |

| | |
|---|---|
| lambda | Best lambda from CV. |
| rank | Feature frequency table. |
| auc_history | Tibble of iteration results. |

## Value

Each function returns a list:

- lasso_prs: model, lambda, perf_train, perf_test, success_n, attempts, auc_history, feature_rank
- lasso_prs_target: pred_df, perf, target_x, target_y
- lasso_prs_rank: plots for feature frequency and diagnostics

## See Also

cv.glmnet, roc

---

# ld_ps_index *Loci_plot: Calculate the LD-matrix (LD r2) for the index SNP*

---

## Description

Loci_plot: Calculate the LD-matrix (LD r2) for the index SNP

## Usage

```
ld_ps_index(
  gwas,
  index = "rs999",
  win = 1000,
  ld_calculation = T,
  bfile = "/Users/leoarrow/project/ref/1kg.v3/EAS",
  plink_bin = plinkbinr::get_plink_exe()
)
```

## Arguments

| | |
|---|---|
| gwas | gwas summary data that needs to select loci and calculate r2 |
| index | index snp |
| win | window size to locally calculate the r2; set it larger than that you want to plot; # default calculate 1MB |
| ld_calculation | if calculate the LD locally, defaut T; use F if the index snp is a rare variant (MAF<0.01) |
| bfile | bfile |
| plink_bin | plinkbinr::get_plink_exe() |

## Value

loci data with calculated r2

leo.gwas_qc *GWAS summary QC pipeline (chip + imputed)*

## Description

Perform QC on GWAS summary stats by merging imputed and genotyped data, removing duplicates, checking consistency, matching to a reference panel, filtering by DAF/F_U cutoffs, and saving results.

## Usage

```
leo.gwas_qc(
  summary_x2_p,
  summary_x2_chip_p,
 ref_p = "/Users/leoarrow/Project/ref/1kg_maf/zuo_ref/1KG-EAS-EAF-chrposa2a1.txt.gz",
  save_dir = "~/Project/BD2025/data/qc/sex_split",
  save_name_prefix = "bd-ASA-41",
  DAF_cutoff = 0.2,
  F_U_cutoff = 0.01
)
```

## Arguments

| | |
|---|---|
| summary_x2_p | Path to imputed summary file (.assoc/.gz). |
| summary_x2_chip_p | Path to genotyped (chip) summary file. |
| ref_p | Path to reference panel or loaded df |
| save_dir | Output directory (default: "~/Project/BD2025/data/qc/sex_split"). |
| save_name_prefix | Prefix for output file names. |
| DAF_cutoff | Numeric. Max \|DAF\| allowed (default 0.2). |
| F_U_cutoff | Numeric. Min F_U required (default 0.01). |

## Details

Major steps:

- Read and standardize imputed & chip data
- Remove duplicates (keep smallest P for genotyped, drop multiple imputed)
- Merge and label GI (imputed/genotyped)
- Match with reference panel and compute DAF
- Filter by cutoffs and save full / P<1e-6 subsets and output

Logs at each step with `leo_log()` for dimension, duplication, NA, etc.

## Value

return(summary_qc); writes QC results to `save_dir`.

## Examples

```
## Not run:
leo.gwas_qc("imp.assoc.gz", "chip.assoc")

## End(Not run)
```

---

`leo_iterator` *Leo batch iterator builder*

---

## Description

Sometimes you need to handle a large number of iterations, but multi-core parallel computing can be tricky—memory usage may grow beyond what is actually required. This function helps by batching the iterations, so you only process a limited number of elements at a time, preventing excessive RAM consumption.

## Usage

```
leo_iterator(elements, batch_size)
```

## Arguments

| | |
|---|---|
| `elements` | A vector or list to be iterated. |
| `batch_size` | Number of elements per batch. |

## Details

Each call to the returned function yields the next batch. Returns `NULL` when no more elements remain.

## Value

A function (no arguments). Repeated calls produce successive batches or `NULL` if finished.

## Examples

```
## Not run:
# Suppose you have 25 elements and want to batch them in groups of 6
nums <- 1:25
it <- leo_iterator(nums, 6)
while (TRUE) {
  batch <- it()
  if (is.null(batch)) break
```

```
  # add your parallel process steps
  print(batch)
}

## End(Not run)
```

---

`leo_map_GtoCP` *Map Gene Symbols to Genomic Positions*

---

## Description

TODO: Merge all gene annotation function into one simple command. This function maps gene symbols to their genomic positions (chromosome, start, end, strand) using the specified method and genome assembly.

## Usage

```
leo_map_GtoCP(
  genes,
  gene_col = NULL,
  method = c("bioconductor", "gtf"),
  genome = c("hg19", "hg38"),
  ...
)
```

## Arguments

| | |
|---|---|
| `genes` | A character vector of gene symbols or a data frame containing gene symbols. |
| `gene_col` | The column name of gene symbols if `genes` is a data frame. |
| `method` | The method to use: `"bioconductor"` or `"gtf"`. |
| `genome` | The genome assembly to use: `"hg19"` or `"hg38"`. |
| `...` | Additional arguments to pass to the GTF method. |

## Details

The function supports two methods:

- `"bioconductor"`: uses Bioconductor packages. See `map_gene_to_chrbp_using_TxDb()`.
- `"gtf"`: uses a GTF file. See `map_gene_to_chrbp_using_gtf()`.

## Value

A data frame with columns: gene_symbol, chr, bp_start, bp_end, strand.

## Examples

```
## Not run:
# Using Bioconductor method with a character vector of gene symbols
leo_map_GtoCP(genes = c("TP53", "BRCA1", "EGFR"), method = "bioconductor", genome = "hg19")

# Using Bioconductor method with a data frame
leo_map_GtoCP(genes = data.frame(gene_name = c("TP53", "BRCA1", "EGFR"),
                                 value = c(1.2, 3.4, 5.6)),
              gene_col = "gene_name", method = "bioconductor", genome = "hg19")

# Using GTF method with a character vector of gene symbols
leo_map_GtoCP(genes = c("TP53", "BRCA1", "EGFR"), method = "gtf", genome = "hg38")

# Using GTF method with a data frame
leo_map_GtoCP(genes = data.frame(gene_name = c("TP53", "BRCA1", "EGFR"),
                                 value = c(1.2, 3.4, 5.6)),
              gene_col = "gene_name", method = "gtf", genome = "hg38",
              download_dir = "~/Project/ref/gtf")

## End(Not run)
```

---

# leo_scale_color *Apply Color Palette to ggplot*

---

## Description

This function applies a specified color palette to a ggplot object, supporting `ggsci`, `RColorBrewer`, and `viridis` palettes, as well as custom colors.

## Usage

```
leo_scale_color(plot, color_palette = "npg")
```

## Arguments

| | |
|---|---|
| `plot` | A ggplot object to which the color palette will be applied. |
| `color_palette` | A character string specifying the color palette to use. Options include `ggsci` palettes ("npg", "lancet", "jama", etc.), `RColorBrewer` palettes, `viridis` palettes, or a custom vector of colors. |

## Value

A ggplot object with the applied color palette.

leo_smr_adjust *Adjust SMR Results with FDR and Bonferroni Corrections*

## Description

It applies FDR/Bonferroni corrections for a single SMR result. The corrected results are saved in an `fdr` subdirectory within the output directory.

## Usage

```
leo_smr_adjust(
  smr_result_path,
  writePath = "",
  out_dir = "",
  QTL_type = "",
  Source = "",
  Tissue = "",
  Outcome_name = "",
  add_info_cols = T,
  drop_non_heidi = T,
  hla = F,
  write_out = T
)
```

## Arguments

| | |
|---|---|
| `smr_result_path` | Character. The path containing SMR result file. Files should have the `.smr` extension. |
| `writePath` | Character. The path to write the adjusted results. |
| `out_dir` | Character. The output directory where adjusted files will be saved. If not specified, defaults to create a dir named `fdr` within dir-name (smr_result_path). |
| `QTL_type` | Character. The type of the QTL for SMR analysis. |
| `Source` | Character. The name of the Source. |
| `Tissue` | Character. The name of the Tissue. |
| `Outcome_name` | Character. The name of the Outcome. |
| `add_info_cols` | Logical. Whether to add information columns for: QTL_type, Source, Tissue and Outcome. Default is `TRUE`. |
| `drop_non_heidi` | Logical. Whether to drop Probes without HEIDI test information. Default is `TRUE`. |
| `hla` | Logical. Whether to pre-exclude the HLA region probes. Default is `False`. |
| `write_out` | Logical. Whether to write the adjusted results to a file. Default is `TRUE`. |

## Examples

```
## Not run:
leo_smr_adjust(
 file = "./output/smr-t2d/sqtl/GTEx49/chr_combined/chr_combine_sQTL_Adipose_Subcutaneous@iri3.smr",
  out_dir = "./output/smr-t2d/sqtl/GTEx49/fdr"
)
leo_smr_adjust(
  paste0("./output/smr-t2d/sqtl/GTEx49/chr_combined/",
         "chr_combine_sQTL_Adipose_Subcutaneous@iri3.smr"),
  writePath = "./haha.fdr", out_dir = ""
)

## End(Not run)
```

---

# leo_smr_adjust_loop *Batch Adjust SMR Results with FDR and Bonferroni Corrections*

---

## Description

This function applies FDR/Bonferroni corrections to all SMR results within a specified directory.

## Usage

```
leo_smr_adjust_loop(
  dir,
  out_dir = "",
  pattern = "\\.smr$",
  QTL_type,
  Source,
  ...
)
```

## Arguments

| | |
|---|---|
| dir | Character. The directory containing SMR result files. Files should have the `.smr` extension and follow the naming convention `exposure@outcome.smr`. |
| out_dir | Character. The output directory where the adjusted SMR files will be saved. If not specified, defaults to creating a `fdr` directory within the input directory. |
| pattern | Character. A regular expression pattern to match SMR result files. Default is `"\.smr$"` (i.e., files ending with `.smr`). |
| QTL_type | Character. Type of QTL (e.g., "eQTL", "sQTL"). |
| Source | Character. Source of the data (e.g., "GTEx", "Westra"). |
| ... | Additional arguments to be passed to `leo_smr_adjust`. |

## Value

NULL.

## Examples

```
## Not run:
leo_smr_adjust_loop(dir     = "./output/smr/sqtl/GTEx49/chr_combined",
                    out_dir = "./output/smr/sqtl/GTEx49/fdr")

## End(Not run)
```

---

`leo_smr_extract_sig_res`

*Extract significant results from* `.all` *files*

---

## Description

This function scans the `combine_1outcome` folder for `.all` files, filters rows where `Pass_FDR == "Pass"` or `Pass_Bonferroni == "Pass"` (depending on `pass_type` argument), and writes the significant subset to a new file (e.g., `smr_res_YYYYMMDD_iri3.sig.all`).

## Usage

```
leo_smr_extract_sig_res(dir, pass_type = c("FDR", "Bonferroni"), out_dir = "")
```

## Arguments

| | |
|---|---|
| `dir` | Character. The main `combine_1outcome` folder from combine_smr_res_1outcome. |
| `pass_type` | Character. Which significance criterion to use: one of `c("FDR", "Bonferroni", "both")`. - "FDR": keep rows where `Pass_FDR == "Pass"` - "Bonferroni": keep rows where `Pass_Bonferroni == "Pass"` - "both": keep rows where either FDR or Bonferroni is "Pass" |
| `out_dir` | Character. Where to write significant results. Default `dir/combine_1outcome/sig`. |

## Value

NULL. Writes `.sig.all` files to disk.

## Examples

```
## Not run:
# For the directory "/Users/leoarrow/project/iridocyclitis/output/smr-t2d",
# after running combine_smr_res_1outcome, we have .all files in
# "smr-t2d/combine_1outcome".

leo_smr_extract_sig_res(
  dir       = "/Users/leoarrow/project/iridocyclitis/output/smr-t2d",
  pass_type = "FDR"
)

## End(Not run)
```

`leo_smr_merge_shared_probes` *Merge multiple SMR files and keep only shared probes for 2 outcomes*

## Description

This function finds intersected `probeID` across all provided files and merges them into a single data frame using `inner_join`.

## Usage

```
leo_smr_merge_shared_probes(
  dir = NULL,
  file_paths = NULL,
  pattern = "\\.sig\\.all$",
  out_file = ""
)
```

## Arguments

| | |
|---|---|
| `dir` | Character. A directory containing SMR files (e.g. ".sig.all" files). If provided, the function will read all matching files in this directory. |
| `file_paths` | Character vector. A set of absolute paths to SMR files. (Priority over `dir`) If this is non-NULL, `dir` is ignored. |
| `pattern` | Character. Regex pattern for searching files in `dir`. Default is "\.sig\.all$". |
| `out_file` | Character. If not empty, write the merged result to this path. Otherwise only return in R. |

## Value

A tibble containing shared probes across all specified files. If `out_file` is provided, the result is written to disk and the function returns `NULL`.

## Examples

```
## Not run:
# 1) Merge all .sig.all files in a directory:
merged_df <- leo_smr_merge_shared_probes(
  dir = "/path/to/smr-t2d/combine_1outcome/sig"
)

# 2) Merge specific files by absolute paths:
files_vec <- c(
  "/path/to/smr/combine_1outcome/smr_res_20241230_iri3.all",
  "/path/to/smr/combine_1outcome/smr_res_20241230_t1d1.all"
)
merged_df <- leo_smr_merge_shared_probes(
```

```
  file_paths = files_vec,
  out_file = "shared_probes_merged.tsv"
)

## End(Not run)
```

---

`locuszoomr_loc` *Loci_plot: prepare the locus data for locuszoomr*

---

## Description

Loci_plot: prepare the locus data for locuszoomr

## Usage

```
locuszoomr_loc(loci_data, gene, online_ld = FALSE, index_snp, flank)
```

## Arguments

| | |
|---|---|
| `loci_data` | output from ld_ps_index |
| `gene` | gene loci |
| `online_ld` | whether to use online LD; default is FALSE |
| `index_snp` | indexed snp |
| `flank` | flank size for the locus plot |

## Value

prepared data which could be pass to save_regional_plot

---

`map_ensg_to_chrbp_using_biomaRt`
*Map Ensembl Gene IDs to Genomic Positions using biomaRt*

---

## Description

This function uses biomaRt to retrieve genomic positions (chr, bp_start, bp_end, and strand) from the Ensembl database based on Ensembl gene IDs.

## Usage

```
map_ensg_to_chrbp_using_biomaRt(
  ensembl_ids,
  ensembl_col = NULL,
  genome = c("hg19", "hg38"),
  verbose = FALSE
)
```

### Arguments

| | |
|---|---|
| ensembl_ids | A character vector of Ensembl gene IDs, or a data frame containing this information. |
| ensembl_col | If ensembl_ids is a data frame, specify the column name containing the Ensembl gene IDs. |
| genome | The genome version to use: "hg19" or "hg38". |
| verbose | Logical indicating whether to print the unmapped information. |

### Value

A data frame containing genomic position information, including ensembl_gene_id, chr, bp_start, bp_end, strand.

### Examples

```
## Not run:
# Query using Ensembl gene IDs
ensembl_ids <- c("ENSG00000141510", "ENSG00000012048", "ENSG00000146648")
map_ensg_to_chrbp_using_biomaRt(ensembl_ids = ensembl_ids, genome = "hg19")

# Use a data frame as input
gene_df <- data.frame(ensembl_id = ensembl_ids, value = c(1.2, 3.4, 5.6))
map_ensg_to_chrbp_using_biomaRt(ensembl_ids = gene_df, ensembl_col = "ensembl_id", genome = "hg19")

## End(Not run)
```

---

## map_ensg_to_gene_using_biomaRt

*Map Ensembl Gene IDs to Gene Symbols using biomaRt*

---

### Description

This function uses biomaRt to retrieve gene symbols based on Ensembl gene IDs.

### Usage

```
map_ensg_to_gene_using_biomaRt(
  ensembl_ids,
  ensembl_col = NULL,
  type = c("combine", "first"),
  sep = "/",
  genome = c("hg19", "hg38"),
  verbose = F
)
```

### Arguments

| | |
|---|---|
| ensembl_ids | A character vector of Ensembl gene IDs, or a data frame containing this information. |
| ensembl_col | If ensembl_ids is a data frame, specify the column name containing the Ensembl gene IDs. |
| type | How to handle cases where one Ensembl ID maps to multiple gene symbols. Options are "combine" (default) to combine them with a separator or "first" to only use the first symbol. |
| sep | The separator to deal with one ensg mapped to multiple gene. Default with "/". |
| genome | The genome version to use: "hg19" or "hg38". |
| verbose | Logical indicating whether to print the unmapped information. |

### Value

A data frame containing Ensembl gene IDs and corresponding gene symbols.

### Examples

```
## Not run:
# Query using Ensembl gene IDs
ensembl_ids <- c("ENSG00000141510", "ENSG00000012048", "ENSG00000146648")
map_ensg_to_gene_using_biomaRt(ensembl_ids = ensembl_ids, genome = "hg19")

# Use a data frame as input
gene_df <- data.frame(ensembl_id = ensembl_ids, value = c(1.2, 3.4, 5.6))
map_ensg_to_gene_using_biomaRt(ensembl_ids = gene_df, ensembl_col = "ensembl_id", genome = "hg19")

## End(Not run)
```

---

## map_ensg_to_gene_using_org.Hs.eg.db

*Map Ensembl IDs to Gene Symbols using* org.Hs.eg.db

---

### Description

This function maps Ensembl IDs to their corresponding gene symbols using the org.Hs.eg.db package.

### Usage

```
map_ensg_to_gene_using_org.Hs.eg.db(ensembl_ids, ensembl_col = NULL)
```

### Arguments

| | |
|---|---|
| ensembl_ids | A character vector of Ensembl IDs or a data frame containing Ensembl IDs. |
| ensembl_col | The column name of Ensembl IDs if ensembl_ids is a data frame. |

## Value

A data frame with two columns: the input Ensembl IDs and the corresponding gene symbols.

## Examples

```
## Not run:
ensembl_ids <- c("ENSG00000141510.1", "ENSG00000012048",
                 "ENSG00000146648") # Example with Ensembl ID vector
map_ensg_to_gene_using_org.Hs.eg.db(ensembl_ids = ensembl_ids)

ensembl_ids_df <- data.frame(EnsemblID = ensembl_ids,
                             OtherInformation = c(1,2,3)) # Example with data frame input
map_ensg_to_gene_using_org.Hs.eg.db(ensembl_ids = ensembl_ids_df,
                                    ensembl_col = "EnsemblID")

## End(Not run)
```

---

# map_ensg_to_tss_using_biomaRt

*Map Ensembl Gene IDs to TSS Using biomaRt*

---

## Description

This function maps Ensembl gene IDs to their transcription start sites (TSS). (Note that this is inferred using the gene start and end information based on strand) It uses biomaRt for the specified genome assembly ("hg19" or "hg38").

## Usage

```
map_ensg_to_tss_using_biomaRt(
  ensembl_ids,
  ensembl_col = NULL,
  genome = c("hg19", "hg38"),
  ...
)
```

## Arguments

| | |
|---|---|
| `ensembl_ids` | A character vector of Ensembl gene IDs, or a data frame containing this information. |
| `ensembl_col` | If `ensembl_ids` is a data frame, specify the column name containing the Ensembl gene IDs. |
| `genome` | The genome version to use: "hg19" or "hg38". |
| `...` | pass to `map_ensg_to_chrbp_using_biomaRt`. See `map_ensg_to_chrbp_using_biomaRt()` |

## Value

A data frame with Ensembl gene IDs and their TSS positions.

## Examples

```
## Not run:
ensembl_ids <- c("ENSG00000141510", "ENSG00000012048", "ENSG00000146648")
map_ensg_to_tss_using_biomaRt(ensembl_ids = ensembl_ids, genome = "hg19")

ensembl_ids_df <- data.frame(EnsemblID = ensembl_ids, OtherInfo = c(1, 2, 3))
map_ensg_to_tss_using_biomaRt(ensembl_ids = ensembl_ids_df,
                              ensembl_col = "EnsemblID",
                              genome = "hg38")

## End(Not run)
```

---

# map_gene_class_using_annotables

*Map Gene Symbols to biotype & description via* **annotables**

---

## Description

Use local dataframes `annotables::grch38` and `annotables::grch37` to add *biotype* and *description* to gene symbols, providing a source column `infer_version` ("grch38" / "grch37" / "unmapped").

## Usage

```
map_gene_class_using_annotables(genes, gene_col = "Gene", quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `genes` | Character vector of gene symbols, or data frame/tibble containing gene symbols |
| `gene_col` | Column name (when `genes` is a table), default `"Gene"` |
| `quiet` | Logical; if `TRUE`, suppress progress messages |

## Details

Logic:

1. Left join with **grch38**; if matched, `infer_version` = `"grch38"`
2. If not matched, fill with **grch37**; if matched, `infer_version` = `"grch37"`
3. If still not matched, fill columns with placeholders as described above

## Value

Data frame with same structure as input, plus `biotype`, `description`, `infer_version`

## Examples

```
## Not run:
map_gene_class_using_annotables(c("TP53","BRCA1","C14orf37"))

df <- tibble::tibble(Gene = c("TP53","XIST","C14orf37"))
map_gene_class_using_annotables(df, gene_col = "Gene")

## End(Not run)
```

---

`map_gene_class_using_biomarRt`

*Map Gene Symbols to biotype & description via* **biomaRt** *(GRCh38 -> GRCh37 fallback)*

---

## Description

First connect to **Ensembl GRCh38** (www.ensembl.org) to query *gene_biotype* and *description* in batch; for unmapped genes, automatically fallback to **GRCh37** (grch37.ensembl.org). Finally append three columns for each gene:

- **biotype**
- **description**
- **infer_version** – "GRCh38" / "GRCh37" / "unmapped"

## Usage

```
map_gene_class_using_biomarRt(genes, gene_col = "Gene", quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `genes` | Character vector of gene symbols, or data frame/tibble containing gene symbols |
| `gene_col` | Column name (when `genes` is a table), default `"Gene"` |
| `quiet` | Logical; if `TRUE`, suppress progress messages |

## Value

Data frame with same structure as input, plus `biotype`, `description`, `infer_version`

## Examples

```
## Not run:
map_gene_class_using_biomarRt(c("TP53", "IGHV1-69", "TRAC", "MIR21"))

df <- tibble::tibble(Gene = c("TP53","XIST","SNORD14A"))
map_gene_class_using_biomarRt(df, gene_col = "Gene")

## End(Not run)
```

map_gene_to_chrbp_using_biomaRt *Map Gene Symbols to Genomic Positions Using biomaRt*

## Description

This function queries gene symbols for their genomic positions (chromosome, start, end, strand) using Ensembl's biomaRt for the specified genome assembly ("hg19" or "hg38").

## Usage

```
map_gene_to_chrbp_using_biomaRt(
  genes,
  gene_col = NULL,
  genome = c("hg19", "hg38")
)
```

## Arguments

| | |
|---|---|
| genes | A character vector of gene symbols to query, or a data frame containing gene symbols. |
| gene_col | The column name of gene symbols if genes is a data frame. |
| genome | The genome assembly to use: "hg19" or "hg38". |

## Details

If the input is a data frame, the function retains all existing columns and adds new columns with the mapping results.

## Value

A data frame with the original data and new columns: chr, bp_start, bp_end, strand, gene_symbol.

## Examples

```
## Not run:
# Query location of TP53, BRCA1, and EGFR genes
gene_symbols <- c("TP53", "BRCA1", "EGFR")
map_gene_to_chrbp_using_biomaRt(genes = gene_symbols, genome = "hg19")

# Using a data frame with gene symbols
gene_symbols_df <- data.frame(GeneName = gene_symbols, OtherInfo = c(1, 2, 3))
map_gene_to_chrbp_using_biomaRt(genes = gene_symbols_df, gene_col = "GeneName", genome = "hg19")

## End(Not run)
```

`map_gene_to_chrbp_using_gtf` *Map Gene Symbols Using GTF File*

## Description

Map Gene Symbols Using GTF File

## Usage

```
map_gene_to_chrbp_using_gtf(
  genes,
  gene_col = NULL,
  genome = c("hg19", "hg38"),
  gtf_file = NULL,
  download_dir = "~/Project/ref/gtf"
)
```

## Arguments

| | |
|---|---|
| `genes` | A character vector of gene symbols or a data frame containing gene symbols. |
| `gene_col` | The column name of gene symbols if `genes` is a data frame. |
| `genome` | The genome assembly to use: `"hg19"` or `"hg38"`. |
| `gtf_file` | The path to a GTF file. If `NULL`, the function will download the appropriate GTF file. |
| `download_dir` | The path where you wanna store the downloaded gtf file |

## Value

A data frame with mapping results.

## Examples

```
## Not run:
gene_symbols <- c("TP53", "BRCA1", "EGFR")
map_gene_to_chrbp_using_gtf(genes = gene_symbols, genome = "hg38")

gene_symbols_df <- data.frame(GeneName = gene_symbols, OtherInformation = c(1,2,3))
map_gene_to_chrbp_using_gtf(genes = gene_symbols_df, gene_col = "GeneName" , genome = "hg19")

## End(Not run)
```

map_gene_to_chrbp_using_TxDb *Map Gene Symbols Using Bioconductor Packages*

## Description

Map Gene Symbols Using Bioconductor Packages

## Usage

```
map_gene_to_chrbp_using_TxDb(
  genes,
  gene_col = NULL,
  genome = c("hg19", "hg38")
)
```

## Arguments

| | |
|---|---|
| genes | A character vector of gene symbols or a data frame containing gene symbols. |
| gene_col | The column name of gene symbols if genes is a data frame. |
| genome | The genome assembly to use: "hg19" or "hg38".<br>• For hg19, it needs "TxDb.Hsapiens.UCSC.hg19.knownGene" Bioconductor package<br>• For hg38, it needs "TxDb.Hsapiens.UCSC.hg38.knownGene" Bioconductor package |

## Value

A data frame with mapped results.

## Examples

```
## Not run:
gene_symbols <- c("TP53", "BRCA1", "EGFR") # Example with gene symbol vector
map_gene_to_chrbp_using_TxDb(genes = gene_symbols, genome = "hg19")

gene_symbols_df <- data.frame(GeneName = gene_symbols,
                              OtherInformation = c(1,2,3)) # Example with data frame input
map_gene_to_chrbp_using_TxDb(genes = gene_symbols_df,
                             gene_col = "GeneName",
                             genome = "hg19")

## End(Not run)
```

`map_gene_to_ensembl` *Map Gene Symbols to Ensembl IDs*

## Description

This function provides robust gene symbol to Ensembl ID mapping through:

1. Local `org.Hs.eg.db` annotations (default)
2. Ensembl BioMart web service (requires internet)

## Usage

```
map_gene_to_ensembl(
  genes,
  gene_col = NULL,
  method = c("org.Hs.eg.db", "biomart"),
  genome = c("hg19", "hg38"),
  type = c("first", "combine"),
  sep = "/",
  batch_size = 100
)
```

## Arguments

| | |
|---|---|
| `genes` | Input containing gene symbols. Can be:<br>• Character vector of gene symbols<br>• Data frame containing gene symbol column |
| `gene_col` | Column name containing gene symbols when `genes` is data frame |
| `method` | Mapping methodology:<br>• "org.Hs.eg.db": Local Bioconductor annotations (default)<br>• "biomart": Ensembl BioMart service |
| `genome` | Genome assembly version (BioMart only):<br>• "hg19": GRCh37 (default)<br>• "hg38": GRCh38 |
| `type` | Multi-mapping handling:<br>• "first": Return first valid ID (default)<br>• "combine": Concatenate multiple IDs |
| `sep` | Separator for combined IDs (default: "/") |
| `batch_size` | BioMart query batch size (default: 100) |

## Value

Data frame with original data + ensembl_id column

## Examples

```
## Not run:
# Local annotation method
genes <- c("TP53", "BRCA1", "VEGFA")
result_local <- map_gene_to_ensembl(genes)

# BioMart with custom parameters
gene_df <- data.frame(
  my_symbol = c("TP53", "BRCA1", "NONEXISTENT"),
  values = rnorm(3)
)
result_biomart <- map_gene_to_ensembl(
  gene_df,
  gene_col = "my_symbol",
  method = "biomart",
  genome = "hg19",
  batch_size = 50
)

## End(Not run)
```

---

`map_gene_to_tss_using_gtf`
*Map Genes to Their TSS Positions*

---

## Description

This function maps gene symbols to their transcription start sites (TSS) positions using hg19 or hg38 genome assembly.

## Usage

```
map_gene_to_tss_using_gtf(
  genes,
  gene_col = NULL,
  genome = c("hg19", "hg38"),
  gtf_file = NULL,
  download_dir = "~/Project/ref/gtf",
  ...
)
```

## Arguments

| | |
|---|---|
| `genes` | A character vector of gene symbols or a data frame containing gene symbols. |
| `gene_col` | The column name of gene symbols if `genes` is a data frame. |
| `genome` | The genome assembly to use: `"hg19"` or `"hg38"`. |
| `gtf_file` | The path to a GTF file. If `NULL`, the function will download the appropriate GTF file. |

| | |
|---|---|
| download_dir | The path where you want to store the downloaded gtf file. |
| ... | Pass to map_gene_to_chrbp_using_gtf. See map_gene_to_chrbp_using_gtf() |

## Value

A data frame with gene symbols and their TSS positions

## Examples

```
## Not run:
genes <- c("TP53", "BRCA1", "EGFR")
map_gene_to_tss_using_gtf(genes = genes, genome = "hg38")

genes_df <- data.frame(GeneName = genes, OtherInfo = c(1,2,3))
map_gene_to_tss_using_gtf(genes = genes_df, gene_col = "GeneName", genome = "hg19")

## End(Not run)
```

---

# mr_one_pair *One-Click Perform 2SMR*

---

## Description

perform_mr_for_one_pair perform a MR for one pair of exp and out

## Usage

```
mr_one_pair(
  dat_h,
  exp = "",
  out = "",
  save_plot = TRUE,
  res_dir = "./output/tsmr",
  fig_dir = "./figure/tsmr"
)
```

## Arguments

| | |
|---|---|
| dat_h | harmonized data |
| exp | a str indicating the exposure in dat_h |
| out | a str indicating the outcome in dat_h |
| save_plot | only save the plot if TRUE, default TRUE. |
| res_dir | dir path where the result of MR analysis stored |
| fig_dir | dir path where the figure of MR analysis stored |

## Value

list(res_pair=res_pair, res_pair_presso=res_pair_presso)

## Examples

```
## Not run:
# dat_h is harmonized TwoSampleMR data with columns:
# exposure, outcome, mr_keep, beta.exposure, beta.outcome, etc.
out <- mr_one_pair(
  dat_h = dat_h,
  exp = "BMI",
  out = "T1D",
  save_plot = FALSE,
  res_dir = "./output/tsmr",
  fig_dir = "./figure/tsmr"
)
names(out)

## End(Not run)
```

---

# mr_scatter_plot_modified

*Modified MR Scatter Plot*

---

## Description

The `mr_scatter_plot` in the `TwoSampleMR` package could be better. This is a modified version of `mr_scatter_plot` in the `TwoSampleMR` package When the slope of two method is really close, the scatter plot may not properly plot them! Change the line type here mannually herein then.

## Usage

```
mr_scatter_plot_modified(mr_results, dat)
```

## Arguments

| | |
|---|---|
| `mr_results` | same as TwoSampleMR::mr_scatter_plot |
| `dat` | same as TwoSampleMR::mr_scatter_plot |

## Examples

```
## Not run:
p1 <- mr_scatter_plot_modified(mr_results = res_pair, dat = dat_h_pair)
print(p1[[1]])

## End(Not run)
```

mrlap_one_pair *mrlap_one_pair*

## Description

`mrlap_one_pair` perform the MR-lap for one pair of exposure and outcome MR-lap

## Usage

```
mrlap_one_pair(
  exposure_data,
  outcome_data,
  exposure_name,
  outcome_name,
  ld_path,
  hm3_path,
  log_path,
  log = TRUE,
  wd = getwd()
)
```

## Arguments

| | |
|---|---|
| `exposure_data` | exposure_data |
| `outcome_data` | outcome_data |
| `exposure_name` | exposure_name |
| `outcome_name` | outcome_name |
| `ld_path` | ldsc required file |
| `hm3_path` | ldsc required file; tutorial use nomhc version list |
| `log_path` | path to where you wanna store the ldsc log |
| `log` | Logical, whether to save the log file (default: TRUE). |
| `wd` | working directory. |

## Value

A data frame containing the results of the MR-lap analysis.

## plink_clump_hla_aware *HLA-aware PLINK clumping (Yet implemented)*

### Description

Simple wrapper: "none" -> drop HLA (chr6:25–34Mb), clump non-HLA only "all" -> keep all HLA (no clump), clump non-HLA, then union "just_clump" -> clump all variants together

### Usage

```
plink_clump_hla_aware(
  bfile,
  hla_keep = c("all", "none", "just_clump"),
  output,
  plink_bin = plinkbinr::get_plink_exe()
)
```

### Arguments

| | |
|---|---|
| bfile | PLINK bfile for LD reference. |
| hla_keep | One of c("all","none","just_clump"). |
| output | Output prefix; expects {output}.clump.txt. |
| plink_bin | Path to PLINK. |

### Details

Input: {output}.clump.txt with columns "SNP","P". Defaults follow common PRS C+T: –clump-p1 1, –clump-p2 1, –clump-r2 0.1, –clump-kb 250.

### Value

Character vector of kept SNPs; also writes {output}.kept.snps.txt.

## plot_gsMap *Plot gsMap (full & highlight) with robust color mapping*

### Description

Main plotting function. Full panel uses per-annotation colors; highlight panel keeps all data but greys out non-highlighted levels.

## Usage

```
plot_gsMap(
  path,
  width1 = 6,
  height1 = 4,
  width2 = 8,
  height2 = 4,
  color = NULL,
  anno_colors = NULL,
  reverse_x = FALSE,
  reverse_y = TRUE,
  save_folder = "./figure/gsmap/tmp",
  basename = NULL,
  traitname = NULL,
  highlight_tissue = NULL,
  highlight_color = NULL,
  other_grey = "grey60"
)
```

## Arguments

| | |
|---|---|
| path | CSV file path with columns: sx, sy, annotation, logp. |
| width1, height1 | PDF size for full-annotation figure. |
| width2, height2 | PDF size for combined highlight+logp figure. |
| color | Named color vector for annotations (fallback). |
| anno_colors | Preferred named color vector for annotations (partial allowed). |
| reverse_x, reverse_y | Reverse axes or not. |
| save_folder | Output folder. |
| basename | Optional plot basename. |
| traitname | Optional trait name. |
| highlight_tissue | Levels to highlight (default: first level if missing). |
| highlight_color | Highlight color (single string or named vector). |
| other_grey | Color used for non-highlight levels in highlight panel. |

## Examples

```
## Not run:
# Simulate & write a small CSV
set.seed(1)
df <- data.frame(sx = rnorm(60), sy = rnorm(60),
                 annotation = rep(c("Heart","Liver","Lung","Brain"), each = 15),
                 logp = runif(60, 1, 12))
fp <- file.path(tempdir(), "A_B_trait_gsMap_plot.csv")
```

```
write.csv(df, fp, row.names = FALSE)

# Partial colors; highlight Brain in gold; others grey
anno_cols <- c(Heart = "#E64B35FF", Liver = "#4DBBD5FF")
plot_gsMap(path = fp, width1 = 5, height1 = 3.5, width2 = 7, height2 = 3.5,
           anno_colors = anno_cols, reverse_x = FALSE, reverse_y = TRUE,
           save_folder = tempdir(), basename = "Demo.Base", traitname = "trait",
          highlight_tissue = "Brain", highlight_color = "#FFD700", other_grey = "grey60")

## End(Not run)
```

---

`plot_gsMap_color` *Build color maps for gsMap plots (helper function)*

---

## Description

gsMap_plot Helper to build color mappings (partial fill, highlight override).

## Usage

```
plot_gsMap_color(
  annos,
  anno_colors = NULL,
  color = NULL,
  highlight_tissue = NULL,
  highlight_color = NULL
)
```

## Arguments

| | |
|---|---|
| `annos` | Character vector of annotation levels (unique, ordered). |
| `anno_colors` | Named colors for some/all levels (preferred over `color`). |
| `color` | Named colors as fallback if `anno_colors` is NULL. |
| `highlight_tissue` | Character vector of levels to highlight; fallback to first of `annos` if none valid. |
| `highlight_color` | Single color string or a named vector mapping highlight levels to colors. |

## Value

A list with:

| | |
|---|---|
| `final_map` | named colors for all `annos` after auto-fill & highlight override (for full plot) |
| `hi_levels` | chosen highlight levels |
| `hi_map` | named colors for highlight levels only (for highlight-only plot) |

## Examples

```
## Not run:
library(patchwork)
library(ggplot2)
# Simulate a small dataframe
df <- data.frame(sx = rnorm(30), sy = rnorm(30),
                 annotation = rep(c("TissueA","TissueB","TissueC"), each = 10),
                 logp = runif(30, 1, 10))

# User provides partial colors; others will be auto-filled
cm <- plot_gsMap_color(annos = unique(df$annotation),
                       anno_colors = c(TissueA = "#F2B701"),
                       highlight_tissue = "TissueB",
                       highlight_color  = "#FF0000")

# Full plot (manual scale with final_map)
gg <- ggplot2::ggplot(df, ggplot2::aes(sx, sy, color = annotation)) +
  ggplot2::geom_point() +
  ggplot2::scale_color_manual(values = cm$final_map)

# Highlight-only plot (only highlight levels mapped)
gg_hi <- ggplot2::ggplot(df, ggplot2::aes(sx, sy, color = annotation)) +
  ggplot2::geom_point() +
  ggplot2::scale_color_manual(values = cm$hi_map, name = "Highlight")
gg | gg_hi

## End(Not run)
```

---

`save_regional_plot` *Loci_plot: save_regional_plot*

---

## Description

Loci_plot: save_regional_plot

## Usage

```
save_regional_plot(
  path,
  loc,
  gene,
  save = TRUE,
  title = expression(paste(italic("CLPSL1"), " (T1D)")),
  labels = c("index"),
  filter_gene_biotype = c("protein_coding"),
  border = FALSE,
  width = 7.5,
  height = 5.5
)
```

## Arguments

| | |
|---|---|
| path | Path to the directory containing summary statistics. |
| loc | Locus string (e.g., "1:1000-2000"). |
| gene | Gene name to highlight. |
| save | Logical. Whether to save the plot (default: TRUE). |
| title | Title of the plot. |
| labels | Labels to indicate index SNP and other SNPs. |
| filter_gene_biotype | Character vector of gene biotypes to filter. |
| border | Logical. Whether to add a border for gene track. |
| width | Plot width. |
| height | Plot height. |

# Index