Package ‘wrapperQTL’

August 31, 2025

Title MatrixEQTL Wrapper for QTL Analysis

Version 1.0.0

Maintainer Lech Kaczmarczyk <drowsygoat1982@gmail.com>

Description MatrixEQTL wrapper for QTL analysis, including functions for running the analysis, processing results, and visualizing QTL data.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports SummarizedExperiment,
ggplot2,
Matrix,
data.table,
ggbeeswarm

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Contents

---

annotate_with_rowdata_peaks

Annotate a tibble with peak coordinates from a SummarizedExperiment

---

Description

Left-joins `df` to `rowData(se)` by matching `df[[key_col]]` to `rownames(se)`. Appends four columns to `df`: `peak_id` (the matched key), `peak_chr` (from `seqnames`), `peak_start` (from `start`), and `peak_end` (from `end`). The number and order of rows in `df` are preserved.

Usage

```
annotate_with_rowdata_peaks(
  df,
  se,
  key_col = "peak_id",
  seq_field = "seqnames",
  start_field = "start",
  end_field = "end",
  warn_unmatched = TRUE
)
```

Arguments

`df`	A data.frame or tibble to annotate (target table).
`se`	A `SummarizedExperiment` whose `rowData` contains `seqnames`, `start`, and `end`; `rownames(se)` must be the peak identifiers to match.
`key_col`	Character scalar. Column in `df` that contains peak IDs matching `rownames(se)`. Default `"peak_id"`.
`seq_field, start_field, end_field`	Character scalars naming the columns in `rowData(se)` to use. Defaults: `"seqnames"`, `"start"`, `"end"`.
`warn_unmatched`	Logical; warn if some keys in `df` do not match any `rownames(se)`. Default `TRUE`.

Value

`df` with added columns: `peak_id`, `peak_chr`, `peak_start`, `peak_end`.

Examples

```
## Not run:
# df has "peak_ID" that matches rownames(se_atac_full)
df2 <- annotate_with_rowdata_peaks(
  df = df,
  se = se_atac_full,
  key_col = "peak_ID"  # becomes df2$peak_id
)

## End(Not run)
```

`checkpoint` *Debug checkpoint logger*

Description

Debug checkpoint logger

Usage

```
checkpoint(msg, verbose = TRUE)
```

`CleanIDs` *Clean and Standardize Sample IDs*

Description

Cleans sample IDs by optionally selecting parts of IDs, removing duplicate halves, and converting to uppercase or lowercase. Can operate directly on a character vector or on the header line of a tab-delimited file (saving a repaired version).

Usage

```
CleanIDs(input, Capitalize = TRUE, pattern = NULL)
```

Arguments

`input`	A character vector of IDs, or a file path to a tab-delimited text file where the first row contains IDs (first column is assumed to be a non-ID column name).
`Capitalize`	Logical; if `TRUE`, converts IDs to uppercase, otherwise to lowercase. Default is `TRUE`.
`pattern`	Optional integer vector specifying which underscore-separated parts of the ID to keep (by position). If `NULL` (default), keeps all parts, but if the ID consists of two identical halves, only keeps the first half.

Details

If a file path is provided, the function:

1. Reads the file.
2. Cleans the sample IDs in the header.
3. Writes a new file with `_names_repaired.txt` appended to the name.

When cleaning IDs:

- IDs are split on underscores.
- If `pattern` is specified, only the specified parts are kept.
- If `pattern` is `NULL` and the ID has two identical halves, only the first half is kept.
- Otherwise, all parts are concatenated.

Value

- If `input` is a vector: a cleaned character vector.
- If `input` is a file path: (invisibly) returns the path to the repaired file.

Examples

```
# Clean a vector of IDs
CleanIDs(c("S1_A", "S2_A"))

# Keep only the first two parts of IDs
CleanIDs(c("S1_part1_part2_extra", "S2_part1_part2_extra"), pattern = 1:2)

# Clean IDs in a file and save repaired file
## Not run:
CleanIDs("matrix_eqtl_input/SNP.txt")

## End(Not run)
```

`convert_chromosome_ids`

Convert chromosome names by adding or removing "chr" prefix

Description

This function converts chromosome identifiers by either adding the "chr" prefix (e.g., 1 -> `chr1`) or removing it (e.g., `chr1` -> `1`). It also handles the mitochondrial chromosome (`MT` <-> `chrM`).

Usage

```
convert_chromosome_ids(chroms, direction = c("add", "remove"))
```

Arguments

`chroms`	A character vector of chromosome names.
`direction`	A string specifying the conversion direction: `"add"` to add the "chr" prefix, or `"remove"` to remove it.

Value

A character vector of converted chromosome names.

Examples

```
convert_chromosome_ids(c("1", "2", "MT"), direction = "add")
convert_chromosome_ids(c("chr1", "chr2", "chrM"), direction = "remove")
```

`ExportFeatureLocationsForMatrixEQTL` *Export gene feature locations for Matrix eQTL analysis*

Description

This function extracts and formats genomic feature location data from a GTF file for a specified set of genes, and writes it in a format compatible with Matrix eQTL. It supports gene identifiers from Seurat objects, `SummarizedExperiments`, or character vectors.

Usage

```
ExportFeatureLocationsForMatrixEQTL(
  gtf_file,
  genes,
  output_dir = ".",
  use_gene_name = TRUE,
  add_chr_prefix = FALSE,
  filename = "feature_locations.txt"
)
```

Arguments

`gtf_file`	Path to a GTF file containing gene annotations.
`genes`	A Seurat object, a `SummarizedExperiment` object, or a character vector of gene identifiers.
`output_dir`	Directory where the output file will be saved. Defaults to the current directory.
`use_gene_name`	Logical. If `TRUE`, match genes by `gene_name` first, falling back to `gene_id`. If `FALSE`, match only by `gene_id`.
`add_chr_prefix`	Logical. If `TRUE`, adds `"chr"` prefix to chromosome names (e.g., 1 -> chr1).
`filename`	Name of the output file. Defaults to `"feature_locations.txt"`.

Details

The function filters GTF entries to keep only those of type `"gene"`, and retrieves their chromosome, start, and end coordinates. If gene identifiers are duplicated between `gene_name` and `gene_id`, matches by `gene_name` are prioritized.

Value

(Invisibly) the path to the output file containing the feature locations.

See Also

`convert_chromosome_ids` for chromosome name formatting

Examples

```
## Not run:
ExportFeatureLocationsForMatrixEQTL(
  gtf_file = "annotation.gtf",
  genes = c("GeneA", "GeneB", "GeneC"),
  output_dir = "eqtl_files",
  use_gene_name = TRUE,
  add_chr_prefix = TRUE
)

## End(Not run)
```

`extract_genotypes_from_vcf_by_pos` *Extract Genotypes from VCF by Chromosome and Position*

Description

Extracts genotype data for specific SNP positions from a VCF file and formats genotypes as unphased allele pairs (e.g., `A/G`, `C/C`).

Usage

```
extract_genotypes_from_vcf_by_pos(
  vcf_path,
  snp_df,
  chr_col = "CHR",
  pos_col = "POS",
  Capitalize = TRUE,
  pattern = NULL
)
```

Arguments

`vcf_path`	Path to the VCF file (can be gzipped or plain text).
`snp_df`	A data frame containing SNP positions to extract. Must include chromosome and position columns.
`chr_col`	Name of the column in `snp_df` that contains chromosome identifiers. Default is `"CHR"`.
`pos_col`	Name of the column in `snp_df` that contains position values. Default is `"POS"`.
`Capitalize`	Logical; whether to capitalize cleaned sample names. Default is `TRUE`.
`pattern`	Optional integer vector specifying which parts of sample names (split by "_") to retain. Used for sample name cleaning.

Details

This function:

- Extracts VCF lines corresponding to provided chromosome-position pairs.
- Parses genotype fields (`GT`) and maps 0/1 alleles to REF/ALT bases.
- Cleans sample names using the provided `pattern` or automatic heuristics.

SNPs are identified by combining chromosome, position, and ALT allele (`CHR:POS_ALT`). Lines without valid genotypes are returned as `NA`.

Value

A data frame of genotypes with SNP IDs as rows and cleaned sample names as columns. Genotypes are in unphased format (e.g., `A/T`, `G/G`).

Examples

```
## Not run:
snp_df <- data.frame(CHR = c("1", "2"), POS = c("12345", "67890"))
vcf_file <- "variants.vcf.gz"
genotypes <- extract_genotypes_from_vcf_by_pos(vcf_file, snp_df)

## End(Not run)
```

`extract_snp_gene_pairs` *Extract SNP-Gene Pairs with Chromosome and Position*

Description

Extracts unique SNP-gene pairs from a data frame and parses SNP identifiers into chromosome and position columns.

Usage

```
extract_snp_gene_pairs(df, snp_col = "snp_id", gene_col = "peak_id")
```

Arguments

`df`	A data frame containing SNP and gene identifiers.
`snp_col`	Name of the column in `df` containing SNP identifiers in the format `"CHR:POS"` or `"CHR:POS_ALT"`. Default is `"snp_id"`.
`gene_col`	Name of the column in `df` containing gene or peak identifiers. Default is `"peak_id"`.

Details

This function:

- Renames the input columns to standardized names (`SNP_ID`, `GENE_ID`).
- Ensures uniqueness of SNP-gene pairs.
- Extracts chromosome and position information from the SNP ID.

Value

A data frame with columns:

- `SNP_ID`: The SNP identifier.
- `GENE_ID`: The gene or peak identifier.
- `CHR`: Chromosome parsed from `SNP_ID`.
- `POS`: Position parsed from `SNP_ID` as an integer.

Examples

```
df <- data.frame(
  snp_id = c("1:12345_A", "2:67890_T"),
  peak_id = c("geneA", "geneB")
)
extract_snp_gene_pairs(df)
```

`get_model_constant` *Get model constant from name*

Description

Get model constant from name

Usage

```
get_model_constant(model)
```

`grep_o` *Extract First Regex Match from String*

Description

A safer wrapper around `regexpr()` + `regmatches()` to extract the first pattern match from each string.

Usage

```
grep_o(strings, pattern)
```

Arguments

`strings`	Character vector to search in.
`pattern`	Regular expression pattern to match.

Value

Character vector of matched substrings. If no match is found, returns NA.

`infer_sex_from_mosdepth` *Infer Sex (ZW system) from mosdepth summaries and optionally write outputs*

Description

Reads all `summary.txt` files under a mosdepth output directory, extracts mean coverage for chromosomes Z and W, computes the Z:W coverage ratio per sample, and calls sex under the avian ZW system (female = ZW, male = ZZ).

Usage

```
infer_sex_from_mosdepth(
  mosdepth_dir,
  out_dir = ".",
  threshold = 1.5,
  chrom_labels = c("Z", "W"),
  verbose = TRUE
)
```

Arguments

`mosdepth_dir`	Character. Path to a directory containing mosdepth outputs.
`out_dir`	Character. Directory where output files (`covariates_sex.txt` and `ZW_coverage_ratio.pdf`) will be written. Default: current working directory.
`threshold`	Numeric. Z:W ratio cutoff (default `1.5`).
`chrom_labels`	Character vector of length 2 or named list with `"Z"` and `"W"`.
`verbose`	Logical. Print messages (default `TRUE`).

Details

Infer biological sex from mosdepth Z and W chromosome coverage

Value

A list with:

- `results`: data.frame with per-sample Z, W, ratio, and sex call.
- `covariates`: one-row data.frame suitable for Matrix eQTL.
- `files`: character vector of summary files used.
- `plot_file`: path to saved PDF (if written).
- `covariate_file`: path to covariate file (if written).

`init_logging` *Initialize Logging*

Description

Initialize Logging

Usage

```
init_logging(logfile = NULL, verbose = TRUE)
```

`load_sliced_file` *Load a SlicedData object from file*

Description

Load a SlicedData object from file

Usage

```
load_sliced_file(file, sliceSize = 2000, first_col_is_rownames = TRUE)
```

`manhattan_plot_gg` *Manhattan Plot with Optional Region Highlighting*

Description

Creates a Manhattan-style plot from a data frame using **ggplot2**, with optional highlighted genomic regions. Chromosome labels are naturally ordered (via `gtools::mixedsort`), basepair positions are auto-scaled to Mb when values look like basepairs, and a horizontal significance line can be added.

Usage

```
manhattan_plot_gg(
  df,
  chr_col = "chr",
  pos_col = "location",
  p_col = "FDR_recal",
  region_start_col = "RegionStart",
  region_end_col = "RegionEnd",
  sig.level = NA,
  show_regions = TRUE,
  region_alpha = 0.1,
  col = c("gray40", "gray60"),
  point_size = 1
)
```

Arguments

`df`	A data.frame containing chromosome, position, and p-value columns.
`chr_col`	Column name (string) for chromosome identifiers. Default `"Chromosome"`.
`pos_col`	Column name (string) for genomic position (bp or Mb). Default `"PeakPosition"`.
`p_col`	Column name (string) for p-values. Default `"PeakPvalue"`.
`region_start_col`	Column name (string) for region start (bp or Mb). Default `"RegionStart"`.
`region_end_col`	Column name (string) for region end (bp or Mb). Default `"RegionEnd"`.
`sig.level`	Numeric p-value threshold for a dashed horizontal line (e.g., `5e-8`). Use `NA` (default) to omit the line.
`show_regions`	Logical; if `TRUE` (default) and both region columns are present in `df`, shaded rectangles are drawn over those regions.
`region_alpha`	Alpha for region shading (0-1). Default `0.1`.
`col`	Two-color vector used to alternate chromosome point colors. Default `c("gray40","gray60")`.
`point_size`	Point size for scatter layer. Default `1`.

Details

If region columns are not present (or `show_regions = FALSE`), the plot is drawn without any highlighted rectangles.

The function:

1. Strips a leading `"chr"` from chromosome labels and orders chromosomes using natural sort.
2. Converts positions to megabases if any positions exceed 1e6.
3. Computes cumulative genomic positions to lay chromosomes end-to-end.
4. Alternates point colors by chromosome.
5. Optionally shades regions if `show_regions = TRUE` and both region columns exist in `df`. If region columns are missing, the plot is drawn without regions.

Value

A `ggplot` object.

Examples

```
## Not run:
df <- data.frame(
  Chromosome = rep(paste0("chr", 1:3), each = 100),
  PeakPosition = rep(seq(1, 5e6, length.out = 100), 3),
  PeakPvalue = runif(300)
)
p <- manhattan_plot_gg(df, sig.level = 5e-8)
print(p)

## End(Not run)
```

`matrixEQTLwrapper` *Run MatrixEQTL*

Description

Run MatrixEQTL

Usage

```
matrixEQTLwrapper(
  feature_locations_path,
  feature_data_path,
  snpFilePath,
  covFilePath,
  snpLocPath,
  group_name,
  resultsDir = getwd(),
  cisDist = 1e+06,
  pvOutputThreshold = 1e-05,
  pvOutputThresholdCis = 1e-04,
  useModel = "linear",
  minPvByGeneSnp = TRUE,
  noFDRsaveMemory = FALSE,
  prefix = NULL,
  pvalueHist = NULL,
  verbose = TRUE,
  dry_run = TRUE,
  colNames_convention
)
```

`nearest_gene_from_gtf` *Nearest gene (by TSS) from a GTF for given chromosome/position(s) (patched to safely harmonize seqlevels)*

Description

Nearest gene (by TSS) from a GTF for given chromosome/position(s) (patched to safely harmonize seqlevels)

Usage

```
nearest_gene_from_gtf(
  gtf_file,
  chr,
  pos,
  output = c("gene", "distance", "both"),
  tss_gr = NULL
)
```

`nearest_gene_from_gtf_build_tss` *Build (and reuse) a TSS GRanges from a GTF (helper for speed)*

Description

Build (and reuse) a TSS GRanges from a GTF (helper for speed)

Usage

```
nearest_gene_from_gtf_build_tss(gtf_file)
```

Arguments

`gtf_file`	Path to GTF.

Value

A `GRanges` of width-1 TSS with `gene_name` (and possibly `gene_id`) in `mcols`. Pass this as `tss_gr=` to `nearest_gene_from_gtf()` for repeated queries.

`peak_range` *Derive Significant QTL Ranges Around Local Peaks*

Description

Iteratively identifies local peaks (smallest p-values) per chromosome and grows significant ranges around each peak using one of two strategies: *DistanceFromPeak* (bounded by absolute genomic distance from the peak) or *AdjacentPoints* (bounded by the largest allowed gap between adjacent significant positions). After collecting candidate ranges, overlapping/nearby ranges are merged (controlled by `MINGAPSIZE`).

Usage

```
peak_range(
  qtl_data,
  METHOD = "DistanceFromPeak",
  SIGTHRESHOLD = 0.01,
  SEARCHDISTANCE = 3e+06,
  LOCALPEAKTHRESHOLD = 0.01,
  MINGAPSIZE = 0
)
```

Arguments

`qtl_data` A data frame/data.table/tibble with at least the columns:

`chr_snp` Chromosome identifier (character/factor).

`location` Genomic position (integer/numeric, increasing within chromosome).

`FDR` Adjusted p-value (numeric); smaller is more significant.

Additional columns are ignored.

`METHOD` Character scalar. Range-building strategy:

- `"DistanceFromPeak"`: Define the region as the farthest significant points within `SEARCHDISTANCE` bp upstream/downstream of the peak. Significance within the region is evaluated vs. a local dynamic threshold (`PEAKVALUE / LOCALPEAKTHRESHOLD`).
- `"AdjacentPoints"`: Walk upstream and downstream from the peak, extending the region while adjacent significant points are no farther apart than `SEARCHDISTANCE` bp. Uses the same local dynamic threshold for significance.

Default `"DistanceFromPeak"`.

`SIGTHRESHOLD`	Numeric in (0,1]. Global significance cutoff for peaks. Iterations stop when the current best (minimum) p-value exceeds this threshold. Default `0.01`.
`SEARCHDISTANCE`	Numeric (bp). For `"DistanceFromPeak"`, the maximum absolute distance from the peak position. For `"AdjacentPoints"`, the maximum gap allowed between consecutive significant positions while extending upstream/downstream. Default `3e6`.
`LOCALPEAKTHRESHOLD`	Numeric > 0. The dynamic threshold factor relative to the local peak p-value; a value of `0.01` means points are considered locally significant if `p <= PEAKVALUE / 0.01` (i.e., up to two orders of magnitude less significant than the peak). Default `0.01`.
`MINGAPSIZE`	Non-negative numeric (bp). Adjacent/overlapping ranges with a gap $\leq$ `MINGAPSIZE` are merged. Default `0`.

Details

The algorithm loops per chromosome: pick the minimum p-value (the peak); construct a candidate range around it using `METHOD` and local dynamic threshold; remove covered rows; repeat until no positions with `p <= SIGTHRESHOLD` remain. Finally, overlapping/nearby ranges are merged with tolerance `MINGAPSIZE`.

Value

A `data.frame` with columns:

`Chromosome` Chromosome id.

`PeakPosition` Position of local peak.

`PeakPvalue` Local peak p-value.

`RegionStart` Start coordinate (inclusive).

`RegionEnd` End coordinate (inclusive).

`NPeaks` Number of peaks merged into the range.

`NSignificant` Count of positions with `p <= SIGTHRESHOLD` inside the range.

`RegionLength` Computed as `RegionEnd - RegionStart + 1`.

Assumptions

- Positions are on the same coordinate system and comparable within a chromosome.
- For reproducibility and performance, positions within each chromosome should be sorted ascending.
- `qtl_data$FDR` contains the p-values to be used (adjusted). If you wish to use raw p-values, pass them in the same column or rename accordingly.

Examples

```
## Not run:
out <- peak_range(qtl_data, METHOD = "DistanceFromPeak",
                  SIGTHRESHOLD = 0.01, SEARCHDISTANCE = 1e6,
                  LOCALPEAKTHRESHOLD = 0.01, MINGAPSIZE = 5e3)
head(out)

## End(Not run)
```

PlotClusterCovariateSummary

Plot Cluster-Sample Summary (Mean/Median of Non-Zero Values)

Description

Reads a serialized `SummarizedExperiment` (`.rds`), computes per **cluster x sample** summaries of a chosen assay (mean and median of non-zero entries), and saves a combined boxplot with jittered points.

Usage

```
PlotClusterCovariateSummary(
  se_path,
  save_dir,
  assay_name = NULL,
  filename = "cluster_summary_plot.pdf"
)
```

Arguments

`se_path`	Character path to an `.rds` file containing a `SummarizedExperiment`.
`save_dir`	Directory where the plot file will be written (created if it does not exist).
`assay_name`	Optional character name of the assay to use. If `NULL` (default), the first assay in `assayNames(se)` is used.
`filename`	Output filename for the saved plot (PDF/PNG depending on extension). Default: `"cluster_summary_plot.pdf"`.

Details

The input `SummarizedExperiment` must contain at least two columns in `colData(se)`:

- `cluster`: cluster identifier (coerced to character)
- `sample_id`: sample identifier (coerced to character)

For each cluster, samples present in that cluster are considered. For each `cluster x sample` subset, all *non-zero* entries of the assay are taken, and their mean and median are computed. These statistics are reshaped to long format and visualized as boxplots with quasi-random points on top.

If `assay_name` is `NULL`, the first assay is selected and a note is messaged. The plot is saved with `ggplot2::ggsave()` to `file.path(save_dir, filename)`.

Value

Invisibly returns `NULL`. The function is used for its side effects (writing a plot).

Package Requirements

This function uses **SummarizedExperiment**, **Matrix**, **ggplot2**, **ggbeeswarm**, **dplyr**, and **tidyr**. The function checks for these packages at runtime.

Examples

```
## Not run:
PlotClusterCovariateSummary(
  se_path   = "path/to/object.rds",
  save_dir  = "results/plots",
  assay_name = NULL,
  filename   = "cluster_summary_plot.pdf"
)

## End(Not run)
```

`plotPeakMatrixBoxplot` *Plot PeakMatrix Boxplots per Sample*

Description

Generates a per-sample boxplot of peak signals from a `SummarizedExperiment` object. Optionally subsets to the top N peaks per sample before plotting. Can save the plot to a PDF and optionally return the subsetted object.

Usage

```
plotPeakMatrixBoxplot(
  se,
  assay_name = "PeakMatrix",
  top_n_peaks = NULL,
  title = NULL,
  outfile = NULL,
  lim_y = 1
)
```

Arguments

`se`	A `SummarizedExperiment` containing the peak signal assay.
`assay_name`	Name of the assay to use. Default is `"PeakMatrix"`.
`top_n_peaks`	Optional integer. If set, selects the top N peaks (by signal) per sample before plotting. The function will return the subsetted `SummarizedExperiment` invisibly in this case.
`title`	Optional plot title. If `NULL`, a default title is generated.
`outfile`	Optional file path. If provided, saves the plot as a high-resolution PDF.
`lim_y`	Numeric; upper y-axis limit for plotting. Default is `1`.

Details

The function:

1. Validates that the input is a `SummarizedExperiment` with the specified assay.
2. Optionally subsets to the top N peaks per sample.
3. Converts the peak matrix to a long format (supports dense and sparse matrices).
4. Creates a combined boxplot + quasi-random scatter plot of peak signals.

Sparse matrices (`dgCMatrix` or `dgTMatrix`) are converted internally for plotting. Missing values are ignored by `ggplot2`.

Value

- If `top_n_peaks` is provided: the subsetted `SummarizedExperiment` (invisibly).
- If `top_n_peaks` is `NULL`: `NULL` (invisibly).

Examples

```
## Not run:
library(SummarizedExperiment)

# Example SE object with assay "PeakMatrix"
plotPeakMatrixBoxplot(se, top_n_peaks = 1000, outfile = "peaks.pdf")

## End(Not run)
```

prepareMatrixEQTLInputs
Prepare Matrix eQTL Input Files

Description

Prepares and filters expression, SNP, and covariate matrices for Matrix eQTL analysis. Supports SummarizedExperiment or raw matrices as input, optional GTF-based annotation, and automatic chunking for large datasets. Saves intermediate results as RDS files.

Usage

```
prepareMatrixEQTLInputs(
  verbose = TRUE,
  features,
  sampleMetadata = NULL,
  groupCol = NULL,
  sampleCol = "sample_id",
  snpPaths,
  covPaths,
  resultsDir,
  minFeatureFrac = 0.8,
  minFeatureMean = 0,
  matrixName = NULL,
  nChunks = 1,
  topNrows = NULL,
  topNSNPs = NULL,
  groupSubset = NULL,
  sliceSize = 2000,
  gtfFile = NULL,
  useGeneName = FALSE
)
```

Arguments

`verbose`	Logical; whether to print progress messages.
`features`	SummarizedExperiment object or expression matrix (genes x samples).
`sampleMetadata`	Optional data.frame with sample annotations (required if `features` is not a SE).
`groupCol`	Column in metadata indicating group assignment.
`sampleCol`	Column in metadata indicating sample identifiers (default: "sample_id").
`snpPaths`	Character vector of paths to SNP matrices or directories.
`covPaths`	Character vector of paths to covariate matrices or directories.
`resultsDir`	Directory to save all output files.
`minFeatureFrac`	Minimum fraction of samples a gene must be detected in (default: 0.8).

`minFeatureMean`	Minimum mean expression for features to be retained (default: 0).
`matrixName`	Name of assay to extract from SummarizedExperiment (optional).
`nChunks`	Number of chunks to split SNP and expression matrices into (default: 5).
`topNrows`	Optional limit on top expressed features to retain.
`topNSNPs`	Optional limit on top SNPs by variance to retain.
`groupSubset`	Optional vector of group names to include.
`sliceSize`	Slice size for SlicedData export (default: 2000).
`gtfFile`	Optional GTF file to retrieve genomic coordinates for features.
`useGeneName`	Logical; whether to prioritize gene name over gene ID (default: FALSE).

Value

Invisible TRUE; writes RDS and text files to `resultsDir`.

`read_chr_pos_from_vcf` *Read Chromosome and Position from a VCF File*

Description

Extracts chromosome and position columns from a VCF file, excluding header lines.

Usage

```
read_chr_pos_from_vcf(vcf_path)
```

Arguments

`vcf_path`	Path to the VCF file (can be gzipped or plain text).

Details

This function:

- Uses `grep -v '^##'` to skip metadata header lines.
- Reads only the first two columns from the VCF: chromosome and position.
- Renames them to `chr_snp` and `location_snp`.
- Applies `convert_chromosome_ids()` function to standardize chromosome names.

Value

A tibble with two columns:

- `chr_snp`: Chromosome identifiers, optionally converted using `convert_chromosome_ids()`.
- `location_snp`: SNP positions as character strings.

Note

This function assumes that a helper function `convert_chromosome_ids()` is available in the environment.

Examples

```
## Not run:
vcf_file <- "variants.vcf.gz"
chr_pos_tbl <- read_chr_pos_from_vcf(vcf_file)

## End(Not run)
```

`run_qtl_workflow` *Run the QTL post-processing workflow (+ optional nearest-gene annotation)*

Description

Run the QTL post-processing workflow (+ optional nearest-gene annotation)

Usage

```
run_qtl_workflow(
  results,
  vcf_file,
  apply_peak_range = TRUE,
 peak_params = list(METHOD = "DistanceFromPeak", SIGTHRESHOLD = 0.01, SEARCHDISTANCE =
    3e+06, LOCALPEAKTHRESHOLD = 0.01, MINGAPSIZE = 0),
  recalc_fdr = TRUE,
  annotate_peaks = FALSE,
  se = NULL,
  key_col = "peak_id",
  nearest_genes = FALSE,
  gtf_file = NULL,
  nearest_on = c("peak", "snp"),
  nearest_output = c("both", "gene", "distance"),
  tss_gr = NULL,
  output_dir = ".",
  prefix = "qtl",
  write_outputs = FALSE
)
```

Arguments

`results`	QTL results data.frame/tibble with columns: `p_value`, `chr_snp`, `location_snp`, `peak_id`, `QTL_type` (and optionally `FDR`).

`vcf_file`	Path to VCF used to filter SNPs (via `read_chr_pos_from_vcf`).
`apply_peak_range`	Logical; run `peak_range()` per cis/trans and join back. Default TRUE.
`peak_params`	List of params passed to `peak_range()`. Default: `list(METHOD="DistanceFromPeak", SIGTHRESHOLD=0.01, SEARCHDISTANCE=3e6, LOCALPEAKTHRESHOLD=0.01, MINGAPSIZE=0)`.
`recalc_fdr`	Logical; if TRUE, add `FDR_recal := p.adjust(p_value,"BH")`. Default TRUE.
`annotate_peaks`	Logical; if TRUE, call `annotate_with_rowdata_peaks()` on cis/trans. Default FALSE.
`se`	SummarizedExperiment for `annotate_with_rowdata_peaks()` (required if `annotate_peaks=TRUE`).
`key_col`	Peak ID column in `results` for annotation. Default "peak_id".
`nearest_genes`	Logical; if TRUE, annotate nearest gene by TSS from a GTF. Default FALSE.
`gtf_file`	Path to GTF (plain or .gz). Required if `nearest_genes=TRUE` and `tss_gr` is NULL.
`nearest_on`	Compute nearest gene relative to `"peak"` (columns `chr`, `location`) or `"snp"` (`chr_snp`, `location_snp`). Default "peak".
`nearest_output`	What to add: `"gene"`, `"distance"`, or `"both"`. Default "both".
`tss_gr`	Optional precomputed TSS GRanges (from `nearest_gene_from_gtf_build_tss()`) to speed up repeated calls.
`output_dir`	Directory for outputs when `write_outputs=TRUE`. Default ".".
`prefix`	Filename prefix for outputs. Default "qtl".
`write_outputs`	Logical; write CSVs and PDFs. Default FALSE.

Value

List:

- `filtered` – VCF-filtered results.
- `peak_counts` – counts per QTL_type.
- `cis_ranges`, `trans_ranges` – outputs from `peak_range()` (if used).
- `cis`, `trans` – final tables (with optional FDR recalculation, rowData & nearest gene annotations).
- `plot_files` – written plot paths (if any).
- `tss` – the TSS GRanges used (if nearest_genes=TRUE).

`test_matrixEQTL_run` *Run test harness for MatrixEQTL*

Description

Run test harness for MatrixEQTL

Usage

```
test_matrixEQTL_run()
```

`validate_sample_overlap`

Validate and reorder sample columns if needed, based on trimmed sample names Trims suffixes from sample names (e.g., removes '_C1_RNA') before checking

Description

Validate and reorder sample columns if needed, based on trimmed sample names Trims suffixes from sample names (e.g., removes '_C1_RNA') before checking

Usage

```
validate_sample_overlap(
  feature_data,
  snps,
  covs,
  group_name,
  colNames_convention,
  verbose = TRUE
)
```

Index