Package 'BioUtils'

May 22, 2026

Title Biological Data Analysis and Visualization

Version 0.1.3

Description Provides tools for the analysis and visualization of gene expression data from the NCBI Gene Expression Omnibus (GEO). Implements a complete workflow including data import, quality control, differential expression analysis, co-expression network analysis, pathway enrichment, and multi-gene biomarker discovery. Differential expression uses the empirical Bayes moderated t-statistic of Smyth (2004) <doi:10.2202/1544-6115.1027>. Gene set enrichment analysis follows Subramanian et al. (2005) <doi:10.1073/pnas.0506580102>. Multi-gene biomarker selection uses the LASSO method of Tibshirani (1996) <doi:10.1111/j.2517-6161.1996.tb02080.x>. Effect sizes are computed as Cohen's d following Cohen (1988).

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.1.2)

Suggests knitr, rmarkdown, msigdbr, plotly

VignetteBuilder knitr

Imports tidyr, GEOquery, Biobase, limma, glmnet, ggplot2, pheatmap, grDevices, fgsea

URL https://spencertreadway.github.io/BioUtils/, https://github.com/spencertreadway/BioUtils

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Spencer Treadway [aut, cre]

Maintainer Spencer Treadway <spencer.treadway08@gmail.com>

Repository CRAN

Date/Publication 2026-05-22 20:24:09 UTC

Contents

`adaptive.t.test` *Perform Adaptive T-Test*

Description

Automatically selects between Welch's and Student's t-test based on the result of a variance equality test, then returns a unified result structure regardless of which branch was taken.

Usage

```
adaptive.t.test(df, alpha = 0.05)
```

Arguments

`df`	A data frame containing at least two columns: **expression** Numeric vector of gene expression values. **group** Character or factor vector with exactly two group labels.
`alpha`	Numeric. Significance level for the variance equality test (`var.test()`). Default is `0.05`.

Details

If `var.test()` returns a p-value below `alpha`, variances are considered unequal and Welch's t-test (`var.equal = FALSE`) is used. Otherwise, Student's t-test (`var.equal = TRUE`) is applied alongside a one-way ANOVA as a confirmatory check. The returned list is consumed internally by `analyze.gene()`.

Value

A named list with three elements:

variance.test The result object from `var.test()` or `oneway.test()`, depending on the branch taken.

t.test The result object from `t.test()`.

p.value Numeric. The p-value from the t-test.

Examples

```
# Build a minimal example data frame
analysis.df <- data.frame(
  expression = c(1.2, 2.3, 1.8, 2.1, 3.4, 2.9, 3.1, 2.7),
  group = rep(c("normal", "RCC"), each = 4)
)
result <- adaptive.t.test(analysis.df, alpha = 0.05)
result$p.value
```

`adjust.pvalues` *Adjust P-Values for Multiple Comparisons*

Description

Applies a multiple testing correction to a vector of raw p-values, reducing the false discovery rate when many genes are tested simultaneously.

Usage

```
adjust.pvalues(p.values, method = "BH")
```

Arguments

`p.values`	Numeric vector of raw p-values, one per gene or probe.
`method`	Character. Correction method passed to `p.adjust()`. Common options are `"BH"` (Benjamini-Hochberg), `"bonferroni"`, and `"holm"`. Default is `"BH"`.

Details

When `analyze.gene()` is applied across many probes in a loop, each test is performed independently and p-values are not corrected for multiplicity. This function should be applied to the resulting p-value vector before interpreting significance. BH correction is recommended for exploratory genomic analyses; Bonferroni is more conservative and suited to confirmatory settings. For genome-wide analysis, prefer `run.limma.de()` which handles correction internally.

Value

A numeric vector of adjusted p-values, the same length and order as the input. For the BH method values represent the estimated false discovery rate (FDR); for Bonferroni and Holm they represent the family-wise error rate.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS507", log.transform = TRUE))
probe.ids <- find.probe.by.gene(geo$gene, c("GENE1", "GENE2", "GENE3"))

raw.pvals <- sapply(probe.ids, function(id) {
  expr <- get.gene.expression(geo$expression, id)
  df <- build.analysis.df(expr, geo$phenotype, geo$gene)
  analyze.gene(df)$p.value
})
adj.pvals <- adjust.pvalues(raw.pvals, method = "BH")
```

`analyze.gene` *Analyze Gene Expression Between Groups*

Description

Performs a comprehensive statistical analysis comparing gene expression between two groups. Integrates parametric and nonparametric testing, effect size estimation, bootstrapped confidence intervals, and a heuristic biological relevance assessment.

Usage

```
analyze.gene(df, alpha = 0.05, n.boot = 1000)
```

Arguments

`df`	A data frame as produced by `build.analysis.df()`, containing at least two columns: **expression** Numeric vector of gene expression values. **group** Character or factor vector with exactly two group labels.

When the data frame contains multiple genes (i.e., a gene column with more than one unique value), expression values from all genes are pooled together. Subset the data frame to a single gene before calling this function for per-gene results.

`alpha` Numeric. Significance threshold. Default is `0.05`.

`n.boot` Integer. Number of bootstrap resamples for the confidence interval. Default is `1000`.

Details

This function integrates multiple statistical perspectives to provide a more complete picture of group differences than a p-value alone. The result is designed to feed directly into `plot.gene.analysis()` for visualization.

Value

A named list with nine elements:

p.value Numeric. P-value from the adaptive t-test.

effect.size Numeric. Cohen's d.

effect.size.class Character. Qualitative magnitude label from `classify.effect.size()`.

confidence.interval Named list with `lower` and `upper` bounds for the bootstrapped CI around Cohen's d.

nonparametric.p Numeric. P-value from the Wilcoxon rank-sum test.

robustness Character. Agreement assessment between parametric and nonparametric tests.

biological.relevance Character. Heuristic relevance label from `flag.biological.relevance()`.

interpretation Character. Human-readable summary combining all of the above.

raw List. Raw output from the parametric and nonparametric tests for further inspection.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS3268", log.transform = TRUE))
probe <- find.probe.by.gene(geo$gene, "mucin 20, cell surface associated")
expr <- get.gene.expression(geo$expression, probe)
analysis.df <- build.analysis.df(expr, geo$phenotype, geo$gene)
result <- analyze.gene(analysis.df, n.boot=100)
cat(result$interpretation)
```

`build.analysis.df` *Build Analysis Data Frame*

Description

Combines a multi-gene expression matrix with sample metadata into a long-format, analysis-ready data frame. Each row represents one gene-sample observation, with human-readable gene names resolved from the annotation data frame.

Usage

```
build.analysis.df(expr.matrix, phenotype, genes, group.col = "disease.state")
```

Arguments

`expr.matrix`	Numeric matrix of gene expression values as returned by `get.gene.expression()`. Rows are probe IDs, columns are sample IDs.
`phenotype`	Data frame of sample metadata as returned by `extract.expression()$phenotype`. Row names must correspond to the column names of `expr.matrix`.
`genes`	Data frame of gene annotations as returned by `extract.expression()$gene`. Row indices must correspond to the probe IDs (row names) of `expr.matrix`. Used to resolve probe IDs to the human-readable gene names stored in the `"Gene symbol"` column.
`group.col`	Character. Name of the column in `phenotype` to use as the grouping variable. Default is `"disease.state"`.

Details

The function pivots `expr.matrix` from wide format (probes x samples) to long format, merges sample metadata by sample ID, resolves probe IDs to gene names using the annotation data frame, and selects the three columns needed for analysis. The output is the standard input for `analyze.gene()`, `gene.analysis.plot()`, and `fit.lasso()`.

Requires `tidyr` for the pivot step. Ensure `tidyr` is listed under `Imports` in the package DESCRIPTION.

Value

A long-format data frame with one row per gene-sample pair and three columns:

gene Character. Human-readable gene name resolved from the gene annotation data frame.

expression Numeric. Expression value for that gene-sample pair.

group Character or factor. Group label for each sample, renamed from the `group.col` column in `phenotype` for consistency with downstream functions.

Rows with `NaN` or `NA` expression values are removed.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS3268", log.transform = TRUE))
probe <- find.probe.by.gene(geo$gene, c("MUC20", "ADH1A"))
expr <- get.gene.expression(geo$expression, probe)
df <- build.analysis.df(expr, geo$phenotype, geo$gene)
head(df)
```

---

classify.effect.size *Classify Effect Size*

Description

Categorizes an absolute Cohen's d value into a qualitative magnitude label using conventional thresholds.

Usage

```
classify.effect.size(d)
```

Arguments

d	Numeric. Cohen's d effect size value (signed or unsigned).

Value

Character string. One of "negligible", "small", "moderate", or "large" based on the absolute value of d, using the thresholds: negligible < 0.2 <= small < 0.5 <= moderate < 0.8 <= large.

Examples

```
classify.effect.size(0.3)   # "small"
classify.effect.size(-0.9)  # "large"
```

`compute.ci` *Compute Confidence Interval for Effect Size*

Description

Estimates a confidence interval for Cohen's d using bootstrap resampling. This provides a measure of uncertainty around the effect size estimate, which is especially useful when sample sizes are small or distributions deviate from normality.

Usage

```
compute.ci(df, alpha = 0.05, n.boot = 1000)
```

Arguments

`df`	A data frame containing at least two columns: **expression** Numeric vector of gene expression values. **group** Character or factor vector indicating group membership.
`alpha`	Numeric. Significance level used to compute the confidence interval bounds. Default is `0.05`, yielding a 95 percent CI.
`n.boot`	Integer. Number of bootstrap resamples to perform. Default is `1000`.

Details

The function repeatedly resamples the input data with replacement and recomputes Cohen's d for each resample. The confidence interval is derived from the empirical distribution of bootstrapped effect sizes using the percentile method.

Value

A named list with two elements:

lower Lower bound of the bootstrapped confidence interval.

upper Upper bound of the bootstrapped confidence interval.

Examples

```
# Build a minimal example data frame
analysis.df <- data.frame(
  expression = c(1.2, 2.3, 1.8, 2.1, 3.4, 2.9, 3.1, 2.7),
  group = rep(c("normal", "RCC"), each = 4)
)
ci <- compute.ci(analysis.df, alpha = 0.05, n.boot = 100)
```

compute.effect.size *Compute Effect Size (Cohen's d)*

Description

Calculates the standardized mean difference between two groups.

Usage

```
compute.effect.size(df)
```

Arguments

df	A data frame containing at least two columns: **expression** Numeric vector of gene expression values. **group** Character or factor vector with exactly two group labels, as produced by `build.analysis.df()`.

Value

Numeric. Cohen's d value. Positive values indicate the first group (alphabetically) has higher mean expression; negative values indicate the second group is higher.

correlation.heatmap.plot *Plot Gene Co-expression Correlation Heatmap*

Description

Visualizes a gene-by-gene correlation matrix as a clustered heatmap, revealing groups of co-expressed genes within a candidate gene set.

Usage

```
correlation.heatmap.plot(cor.matrix, gene.names = NULL)
```

Arguments

cor.matrix	Numeric matrix as returned by `gene.correlation.matrix()`. Must be square and symmetric with row/column names corresponding to probe IDs.
gene.names	Character vector of gene names to use as axis labels in place of probe IDs. Must be the same length and order as `cor.matrix` rows. Default is `NULL`, which uses probe IDs.

Details

Hierarchical clustering of the correlation matrix groups genes with similar co-expression patterns into visible blocks on the heatmap. These blocks often correspond to genes in the same pathway or under shared regulatory control. This function is a targeted companion to `gene.correlation.matrix()` for a pre-selected gene set, and complements the genome-wide network view produced by WGCNA.

Value

A `pheatmap` object displaying the correlation matrix with hierarchical clustering applied to both rows and columns. Color scale runs from blue (strong negative correlation) through white (no correlation) to red (strong positive correlation).

Examples

```
mat <- matrix(
  c(1.00, 0.85, 0.62, 0.91,
    0.85, 1.00, 0.74, 0.88,
    0.62, 0.74, 1.00, 0.69,
    0.91, 0.88, 0.69, 1.00),
  nrow = 4,
  dimnames = list(
    c("BRCA1", "TP53", "MYC", "EGFR"),
    c("BRCA1", "TP53", "MYC", "EGFR")
  )
)
correlation.heatmap.plot(mat, gene.names = c("BRCA1", "TP53", "MYC", "EGFR"))
```

`extract.expression` *Extract Expression and Metadata from GEO Dataset*

Description

Decomposes an `ExpressionSet` object into its core components, returning a named list that is used as the primary data structure throughout BioUtils.

Usage

```
extract.expression(eset)
```

Arguments

`eset`	An `ExpressionSet` object as returned by `load.geo.soft()`.

Details

The three returned components form the basis of all downstream BioUtils workflows. The `gene` data frame is passed to `find.probe.by.gene()` for probe lookup, `expression` is passed to `get.gene.expression()`, and `phenotype` is passed to `build.analysis.df()` for group labeling.

Value

A named list with three elements:

expression Numeric matrix of gene expression values. Rows are probe IDs, columns are sample IDs.

phenotype Data frame of sample metadata (e.g., disease state, age, tissue). Row names are sample IDs matching the column names of `expression`.

gene Data frame of probe-level gene annotations (e.g., gene symbol, gene title, chromosomal location). Row names are probe IDs matching the row names of `expression`.

Examples

```
eset <- load.geo.soft(accession = "GDS507", log.transform = TRUE)
geo <- extract.expression(eset)
head(geo$phenotype)
dim(geo$expression)
```

`find.probe.by.gene` *Find Probe IDs by Gene Name*

Description

Searches the gene annotation data frame to find the probe IDs corresponding to one or more gene names or descriptions.

Usage

```
find.probe.by.gene(genes, gene.names)
```

Arguments

`genes`	Data frame of gene annotations as returned by `extract.expression()$gene`. The second column (index 2) is expected to contain gene titles or descriptions.
`gene.names`	Character vector of one or more gene names or descriptions to search for. Matching is exact and case-sensitive.

Details

Probe IDs are the integer row names of the `genes` annotation data frame. These IDs are used directly to index rows of the expression matrix in `get.gene.expression()`. If multiple gene names are supplied, all matching probe IDs are returned as a vector, which is then passed as-is to `get.gene.expression()` to retrieve a multi-row expression matrix.

Value

Integer vector of probe IDs corresponding to the matched genes. The length of the vector equals the number of matches found. Returns an empty integer vector if no matches are found.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS3268", log.transform = TRUE))

# Single gene
probe <- find.probe.by.gene(geo$gene, "mucin 20, cell surface associated")

# Multiple genes
probes <- find.probe.by.gene(geo$gene, c(
  "mucin 20, cell surface associated",
  "alcohol dehydrogenase 1A (class I), alpha polypeptide"
))
```

---

fit.lasso *Fit LASSO Regression for Multi-Gene Biomarker Discovery*

Description

Applies cross-validated LASSO logistic regression to identify a sparse set of genes that jointly predict a binary phenotype. Unlike univariate tests, LASSO accounts for the joint contribution of all genes simultaneously, automatically penalizing redundant predictors.

Usage

```
fit.lasso(expression.matrix, phenotype.vector, alpha = 1, nfolds = 10)
```

Arguments

`expression.matrix`	Numeric matrix of gene expression values as returned by `extract.expression()$expression`. Rows are probes, columns are samples. The matrix is transposed internally so samples become rows as required by `glmnet`.
`phenotype.vector`	Factor or integer vector of binary phenotype labels, one per sample (e.g., `0` for control, `1` for disease). Must be the same length as the number of columns in `expression.matrix`.
`alpha`	Numeric. Elastic net mixing parameter. `1` (default) gives pure LASSO; `0` gives Ridge; values between 0 and 1 give elastic net.
`nfolds`	Integer. Number of cross-validation folds. Default is `10`.

Details

LASSO is suited to high-dimensional genomic data where the number of genes far exceeds samples. It shrinks most coefficients to exactly zero, retaining only genes with independent predictive value. This complements `run.limma.de()` - while limma ranks genes by individual differential expression, LASSO identifies the minimal subset with non-redundant joint predictive signal.

Value

A `cv.glmnet` object. Key elements:

lambda.min Lambda with minimum cross-validated error.

lambda.1se Largest lambda within 1 SE of the minimum (sparser model).

glmnet.fit The full sequence of fitted models across lambda values.

Extract selected genes with `coef(fit, s = "lambda.1se")`; probes with non-zero coefficients are the LASSO-selected biomarkers.

Examples

```
# Synthetic example — small expression matrix with binary outcome
set.seed(42)
expr.mat <- matrix(rnorm(200), nrow = 20, ncol = 10)
rownames(expr.mat) <- paste0("probe", 1:20)
colnames(expr.mat) <- paste0("sample", 1:10)
phenotype.binary <- c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1)
lasso.fit <- fit.lasso(expr.mat, phenotype.binary)
coef(lasso.fit, s = "lambda.1se")
```

`flag.biological.relevance` *Assess Biological Relevance of Gene Expression Differences*

Description

Provides a heuristic interpretation of whether a statistically significant difference in gene expression is likely to be biologically meaningful, based on effect size magnitude and p-value.

Usage

```
flag.biological.relevance(effect.size, p.value, alpha = 0.05)
```

Arguments

`effect.size`	Numeric. Cohen's d or other standardized effect size.
`p.value`	Numeric. P-value from a statistical test.
`alpha`	Numeric. Significance threshold used for the p-value comparison. Default is `0.05`.

Details

These rules are heuristic and intended as guidance rather than definitive biological conclusions. The thresholds for effect size (0.5) and significance are conventional starting points; domain knowledge should inform interpretation. This function is called internally by `analyze.gene()` using the same `alpha` passed to that function.

Value

Character string. One of three heuristic labels:

"Potentially biologically meaningful" p < alpha and |d| > 0.5

"Statistically significant but small effect" p < alpha and |d| <= 0.5

"No strong evidence of biological relevance" p >= alpha

Examples

```
flag.biological.relevance(effect.size = 0.6, p.value = 0.01)
```

`gene.analysis.plot` *Plot Gene Expression with Automatic Statistical Analysis*

Description

Generates an annotated visualization of gene expression between two groups. For a single-gene data frame, the plot is annotated with statistical results from `analyze.gene()`. For a multi-gene data frame, the plot is faceted by gene and annotations are omitted, use `analyze.gene()` on each gene subset for per-gene statistics.

Usage

```
gene.analysis.plot(df, alpha = 0.05, n.boot = 1000, show.points = TRUE)
```

Arguments

`df`	A data frame as produced by `build.analysis.df()`, containing: **expression** Numeric vector of gene expression values. **group** Character or factor vector with exactly two group labels. **gene** Character vector of gene names. Optional; when present and containing more than one unique value, enables faceted multi-gene plotting.
`alpha`	Numeric. Significance level passed to `analyze.gene()`. Default is `0.05`. Only used in single-gene mode.
`n.boot`	Integer. Number of bootstrap resamples for the confidence interval, passed to `analyze.gene()`. Default is `1000`. Only used in single-gene mode.
`show.points`	Logical. Whether to overlay jittered individual sample points on the violin/boxplot. Default is `TRUE`.

Details

Single-gene mode is triggered when the gene column is absent or contains exactly one unique value. Multi-gene mode is triggered when gene contains more than one unique value, producing a facet_wrap(~ gene) layout. The plot theme is consistent across both modes and matches the style conventions of the BioUtils package.

Value

A ggplot object. In single-gene mode the plot includes a text annotation with p-value, Cohen's d, and bootstrapped CI, and a subtitle with the full interpretation string. In multi-gene mode the plot is faceted by gene with no statistical annotation.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS3268", log.transform = TRUE))

# Single-gene plot with statistical annotation
probe <- find.probe.by.gene(geo$gene, "mucin 20, cell surface associated")
expr <- get.gene.expression(geo$expression, probe)
df <- build.analysis.df(expr, geo$phenotype, geo$gene)
gene.analysis.plot(df)

# Multi-gene faceted plot
probes <- find.probe.by.gene(geo$gene, c(
  "mucin 20, cell surface associated",
  "alcohol dehydrogenase 1A (class I), alpha polypeptide"
))
expr.multi <- get.gene.expression(geo$expression, probes)
df.multi <- build.analysis.df(expr.multi, geo$phenotype, geo$gene)
gene.analysis.plot(df.multi)
```

---

gene.correlation.matrix

Compute Pairwise Gene Co-expression Correlation Matrix

Description

Calculates pairwise correlations between a set of genes across all samples, producing a symmetric correlation matrix that quantifies co-expression relationships.

Usage

```
gene.correlation.matrix(expression.matrix, probe.ids, method = "pearson")
```

Arguments

expression.matrix	Numeric matrix of gene expression values as returned by `extract.expression()$expression`. Rows are probes, columns are samples.
probe.ids	Integer vector of probe IDs to include, as returned by `find.probe.by.gene()`.
method	Character. Correlation method: `"pearson"` (default), `"spearman"`, or `"kendall"`. Spearman is recommended when distributions are skewed or outliers are a concern.

Details

Co-expression correlations capture whether two genes tend to be simultaneously up- or down-regulated across samples, which can suggest shared regulatory control or pathway membership. The resulting matrix is the direct input for `plot.correlation.heatmap()`.

Value

A symmetric numeric matrix of dimensions `length(probe.ids)` x `length(probe.ids)`, where each cell contains the pairwise correlation coefficient across all samples. Diagonal values are 1. Row and column names correspond to probe IDs.

Examples

```
set.seed(42)
expr.mat <- matrix(rnorm(400), nrow = 4, ncol = 100)
rownames(expr.mat) <- c(101, 102, 103, 104)
probe.ids <- c(101, 102, 103, 104)
cor.mat <- gene.correlation.matrix(expr.mat, probe.ids)
correlation.heatmap.plot(cor.mat, gene.names = c("BRCA1", "TP53", "MYC", "EGFR"))
```

`get.gene.expression` *Extract Gene Expression by Probe ID*

Description

Retrieves expression values for one or more genes from the expression matrix using their integer probe IDs.

Usage

```
get.gene.expression(expression, probe.id)
```

Arguments

expression	Numeric matrix of gene expression values as returned by `extract.expression()$expression`. Rows are probe IDs, columns are sample IDs.
probe.id	Integer vector of one or more probe IDs as returned by `find.probe.by.gene()`.

Details

The returned matrix is the direct input for `build.analysis.df()`. Using `as.matrix()` ensures the single-probe case returns a one-row matrix rather than a named vector, which keeps the `pivot_longer` step in `build.analysis.df()` consistent regardless of how many genes are queried.

Value

A numeric matrix with one row per probe and one column per sample. Row names are the probe IDs; column names are the sample IDs. When a single probe ID is supplied the result is still a matrix (not a vector), ensuring that `build.analysis.df()` receives a consistent input type.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS3268", log.transform = TRUE))
probes <- find.probe.by.gene(geo$gene, c(
  "mucin 20, cell surface associated",
  "alcohol dehydrogenase 1A (class I), alpha polypeptide"
))
expr <- get.gene.expression(geo$expression, probes)
dim(expr)  # probes x samples
```

`get.gene.name` *Resolve Probe IDs to Gene Names*

Description

Converts one or more integer probe IDs back to their human-readable gene title strings or short gene symbols using the gene annotation data frame. This is the inverse operation of `find.probe.by.gene()`.

Usage

```
get.gene.name(genes, probe.id, use.symbols = FALSE)
```

Arguments

`genes`	Data frame of gene annotations as returned by `extract.expression()$gene`. Must contain columns `"ID"`, `"Gene title"`, and `"Gene symbol"`.
`probe.id`	Integer or character vector of one or more probe IDs to resolve, as returned by `find.probe.by.gene()`.
`use.symbols`	Logical. If `FALSE` (default), returns full descriptive gene titles from the `"Gene title"` column, suitable for interpretation and reporting. If `TRUE`, returns short gene symbols from the `"Gene symbol"` column, suitable for plot axis labels and any downstream tool that expects standard gene symbols such as `run.gsea()`.

Value

Character vector of gene names corresponding to the supplied probe IDs. Returns empty strings `""` for probes with no annotation, which should be filtered with `which(result != "")`.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS507",
                                        log.transform = TRUE))
de.results <- run.limma.de(geo)
top.probes <- rownames(head(de.results, 10))

# Full titles for reporting
top.titles  <- get.gene.name(geo$gene, top.probes)

# Short symbols for plot labels and GSEA
top.symbols <- get.gene.name(geo$gene, top.probes, use.symbols = TRUE)
```

`load.geo.soft` *Load GEO Dataset from SOFT File*

Description

Imports a GEO dataset and converts it into a structured ExpressionSet object. If a local SOFT file is found at `file.path` it is loaded directly. Otherwise the dataset is downloaded from NCBI GEO using `accession` and cached in `tempdir()`.

Usage

```
load.geo.soft(file.path = NULL, accession = NULL, log.transform = FALSE)
```

Arguments

`file.path`	Character. Path to a local GEO SOFT file. If the file does not exist and `accession` is provided, the dataset is downloaded automatically.
`accession`	Character. GEO accession ID (e.g., `"GDS507"`). Only used when `file.path` does not exist. Default is `NULL`.
`log.transform`	Logical. Whether to apply a log2 transformation to the expression matrix during import. Default is `FALSE`.

Value

An `ExpressionSet` object for use with `extract.expression()`.

Examples

```
# Download automatically if not found locally
eset <- load.geo.soft("GDS507.soft", accession = "GDS507", log.transform = TRUE)

# Download without a local file at all
eset <- load.geo.soft(NULL, accession = "GDS507", log.transform = TRUE)
```

---

nonparametric.test *Perform Nonparametric Group Comparison*

Description

Conducts a Wilcoxon rank-sum test (Mann-Whitney U test) to compare expression values between two groups without assuming normality.

Usage

```
nonparametric.test(df)
```

Arguments

df	A data frame containing at least two columns:
	expression Numeric vector of gene expression values.
	group Character or factor vector with exactly two group labels.

Details

This test is useful as a robustness check against parametric methods such as the t-test, especially when data are skewed or contain outliers. The result is used by `analyze.gene()` to populate the `robustness` field of its return value.

Value

A named list with two elements:

p.value Numeric. P-value from the Wilcoxon rank-sum test.

test Character. Name of the test used (`"Wilcoxon rank-sum"`).

Examples

```
analysis.df <- data.frame(
  expression = c(1.2, 2.3, 1.8, 2.1, 3.4, 2.9, 3.1, 2.7,
                 1.5, 2.0, 1.9, 2.4),
  group      = rep(c("normal", "RCC"), each = 6)
)
res <- nonparametric.test(analysis.df)
```

```
res$p.value
```

`pca.plot` *Plot PCA of Sample Expression Profiles*

Description

Reduces the dimensionality of the full expression matrix using Principal Component Analysis and plots samples in the first two principal component axes, colored by a phenotype variable. Useful for quality control, batch effect detection, and exploratory assessment of sample clustering.

Usage

```
pca.plot(
  expression.matrix,
  phenotype,
  color.by = "disease.state",
  scale = TRUE
)
```

Arguments

`expression.matrix`	Matrix of gene expression values. Rows are probes, columns are samples.
`phenotype`	Data frame of sample metadata, as returned by `extract.expression()$phenotype`.
`color.by`	Character. Column name in `phenotype` to use for point coloring. Default is `"disease.state"`.
`scale`	Logical. Whether to scale probes to unit variance before PCA. Default is `TRUE`.

Details

PCA is typically applied at the whole-dataset level before any gene filtering, making it a useful first diagnostic after `load.geo.soft()` and `extract.expression()`. Tight clustering of samples by disease state in PC1/PC2 space suggests a strong and detectable expression signal. Unexpected clustering by batch, sex, or other covariates can indicate the need for covariate adjustment in downstream `run.limma.de()` models.

Value

A ggplot object showing samples as points in PC1 vs PC2 space, colored by the specified phenotype variable. The percentage of variance explained is shown on each axis label.

Examples

```
# Create a synthetic expression matrix (50 probes x 12 samples)
set.seed(42)
expr.mat <- matrix(rnorm(600), nrow = 50, ncol = 12)
rownames(expr.mat) <- paste0("probe", 1:50)
colnames(expr.mat) <- paste0("sample", 1:12)

# Create a matching phenotype data frame
phenotype <- data.frame(
  disease.state = rep(c("normal", "RCC"), each = 6),
  row.names = paste0("sample", 1:12)
)

pca.plot(expr.mat, phenotype, color.by = "disease.state")
```

`run.gsea` *Run Gene Set Enrichment Analysis*

Description

Tests whether predefined gene sets (e.g., pathways or GO terms) are systematically enriched among genes ranked by differential expression, using the fgseaMultilevel algorithm from `fgsea`.

Usage

```
run.gsea(de.results, genes, pathways, min.size = 15, max.size = 500)
```

Arguments

`de.results`	Data frame as returned by `run.limma.de()`. Must contain a `logFC` column. Row names are probe IDs.
`genes`	Data frame of gene annotations as returned by `extract.expression()$gene`. Must contain columns `"ID"` for probe identifiers and `"Gene symbol"` for gene symbols matching the format used by pathway databases such as MSigDB.
`pathways`	Named list of character vectors, where each element is a gene set and names are pathway labels. MSigDB or GO gene sets in this format can be loaded via `msigdbr`.
`min.size`	Integer. Minimum number of genes required in a gene set for it to be tested. Default is `15`.
`max.size`	Integer. Maximum gene set size. Default is `500`.

Details

Uses the `fgseaMultilevel` algorithm, which estimates p-values using an adaptive multilevel splitting Monte Carlo approach. This is more accurate than the original permutation-based `fgseaSimple` and does not require a fixed permutation count.

GSEA operates on the full ranked gene list from `run.limma.de()` rather than a filtered subset, preserving information from all genes. Probe IDs from the TopTable row names are resolved to gene symbols via the `genes` annotation data frame before matching against pathway gene sets. Probes with no gene annotation are silently dropped from the ranked list, as they cannot be matched to any pathway. The `leadingEdge` genes are strong candidates for follow-up with `analyze.gene()` or `gene.analysis.plot()`.

Value

A data frame with one row per tested pathway, containing:

pathway Gene set name.

pval Nominal p-value.

padj BH-adjusted p-value across all tested pathways.

NES Normalized enrichment score. Positive values indicate enrichment among upregulated genes; negative values indicate enrichment among downregulated genes.

size Number of genes from the pathway present in the ranked list.

leadingEdge List-column of genes driving the enrichment score.

Examples

```
library(msigdbr)
geo <- extract.expression(load.geo.soft(accession = "GDS507",
                                        log.transform = TRUE))
hallmark.df <- msigdbr(species = "Homo sapiens", category = "H")
pathways <- split(hallmark.df$gene_symbol, hallmark.df$gs_name)
de.results <- run.limma.de(geo)
gsea.results <- run.gsea(de.results, geo$gene, pathways)
head(gsea.results[order(gsea.results$padj), ])
```

---

run.limma.de *Run Differential Expression Analysis Using limma*

Description

Performs genome-wide differential expression analysis across all probes using the limma linear modeling framework. This is the recommended approach for microarray data from GEO, as it borrows variance information across genes to produce more stable estimates than gene-by-gene t-tests.

Usage

```
run.limma.de(geo, condition.col = "disease.state", adjust.method = "BH")
```

Arguments

geo	Named list as returned by extract.expression(), containing at minimum $expression (probe x sample matrix) and $phenotype (sample metadata data frame).
condition.col	Character. Name of the column in geo$phenotype to use as the grouping variable. Default is "disease.state".
adjust.method	Character. P-value correction method passed to limma::topTable(). Default is "BH" (Benjamini-Hochberg FDR).

Details

Unlike running analyze.gene() on each probe individually, limma fits a linear model to all probes simultaneously and applies empirical Bayes shrinkage to the variance estimates. This stabilizes results for genes with few observations and is the standard method for microarray DE analysis.

The returned TopTable is the primary input for downstream functions such as volcano.plot() and run.gsea(), and can be filtered by adj.P.Val and logFC thresholds to define a gene signature.

Value

A data frame (TopTable) with one row per probe, sorted by evidence of differential expression, containing:

logFC Log2 fold-change between conditions.

AveExpr Average log2 expression across all samples.

t Moderated t-statistic.

P.Value Raw p-value.

adj.P.Val FDR-adjusted p-value (BH by default).

B Log-odds that the gene is differentially expressed.

Examples

```
geo <- extract.expression(load.geo.soft(accession = "GDS3268", log.transform = TRUE))
de.results <- run.limma.de(geo, condition.col = "disease.state")
head(de.results)
```

`volcano.plot` *Plot Volcano Plot from Differential Expression Results*

Description

Generates a volcano plot visualizing the relationship between statistical significance and fold-change magnitude for all probes in a differential expression analysis. Points are colored by whether they exceed user-defined significance and fold-change thresholds.

Usage

```
volcano.plot(de.results, fc.threshold = 1, fdr.threshold = 0.05)
```

Arguments

`de.results`	Data frame as returned by `run.limma.de()`, containing at minimum the columns `logFC` and `adj.P.Val`.
`fc.threshold`	Numeric. Absolute log2 fold-change cutoff for labeling genes as differentially expressed. Default is 1 (i.e., 2-fold).
`fdr.threshold`	Numeric. Adjusted p-value cutoff. Default is `0.05`.

Details

The volcano plot is the standard summary visualization for a TopTable from `run.limma.de()` and serves as the genome-wide complement to `gene.analysis.plot()`, which visualizes a single gene. Probes that appear in the upper corners (high fold-change, low FDR) are the strongest candidates for follow-up with `analyze.gene()` or inclusion in a gene signature for `run.gsea()`.

Value

A ggplot object showing each probe as a point with `logFC` on the x-axis and `-log10(adj.P.Val)` on the y-axis. Points are colored as upregulated, downregulated, or not significant. Dashed threshold lines are drawn at the specified cutoffs.

Index