Package 'GExPipe'

September 7, 2026

Title GExPipe: Gene Expression Pipeline Shiny Application

Version 0.99.51

Description Shiny application (GExPipe) for high-throughput genomic analysis of bulk RNA-seq and microarray data (e.g. from GEO). Integrates with Bioconductor (GEOquery, Biobase, limma, DESeq2, edgeR, clusterProfiler) for a single workflow: download, QC, normalization, batch correction, differential expression, WGCNA, pathway enrichment, PPI, and machine learning. Uses common data structures (ExpressionSet, DGEList) for interoperability. For a full dependency tree (including STRINGdb + PPI helpers), use BiocManager::install(``GExPipe", dependencies = TRUE). STRING data are downloaded on first PPI use (internet required); they cannot be bundled in the package.
Microarray CEL normalization uses affy and/or oligo when supplementary CEL files are available.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.6.0)

biocViews Software, ShinyApps, GeneExpression, RNASeq, Microarray, DifferentialExpression, Normalization, Pathways, Network, NetworkEnrichment, Visualization

URL https://github.com/safarafique/GExPipe

BugReports https://github.com/safarafique/GExPipe/issues

Imports affy (>= 1.84.0), AnnotationDbi (>= 1.64.0), Biobase (>= 2.62.0), biomaRt (>= 2.58.0), caret (>= 6.0.94), circlize (>= 0.4.16), cli (>= 3.6.0), clusterProfiler (>= 4.10.0), data.table (>= 1.15.0), DESeq2 (>= 1.42.0), dplyr (>= 1.1.0), DT (>= 0.30), dynamicTreeCut (>= 1.63.1), edgeR (>= 4.0.0), enrichplot (>= 1.22.0), GEOquery (>= 2.70.0), ggplot2 (>= 3.4.0), ggpubr (>= 0.6.0), ggraph (>= 2.2.0), ggrepel (>= 0.9.5), glmnet (>= 4.1.0), glue (>= 1.6.0), gridExtra (>= 2.3), igraph (>= 2.0.0), lifecycle (>= 1.0.0), limma (>= 3.58.0), Matrix (>= 1.6.0), msigdbr (>= 7.5.1), oligo (>= 1.66.0), org.Hs.eg.db (>= 3.17.0), parallel, methods, pheatmap (>= 1.0.12), pillar (>= 1.9.0), pROC (>= 1.18.0), R.utils (>= 2.12.0), randomForest (>= 4.7.1), RColorBrewer (>= 1.1.3), Rcpp (>= 1.0.12), reshape2 (>= 1.4.4), rlang (>= 1.1.0), rms (>= 6.7.0), scales (>= 1.3.0), shiny (>= 1.8.0), shinydashboard (>= 0.7.2), shinyjs (>= 2.1.0), STRINGdb (>= 2.14.0), sva (>= 3.50.0), tibble (>= 3.2.0), tidyr (>= 1.3.0), tidygraph (>=

1.3.0), UpSetR (>= 1.4.0), vctrs (>= 0.6.0), VennDiagram (>= 1.7.0), WGCNA (>= 1.72), withr (>= 2.5.0), xgboost (>= 1.7.0)

Suggests BiocCheck, BiocManager, BiocStyle, Boruta (>= 8.0.0), bslib, car (>= 3.1.0), chromote, cicerone (>= 1.0.4), corrplot (>= 0.92), crosstalk, dcurves (>= 0.5.0), devtools, fontawesome, htmltools, htmlwidgets, kernlab (>= 0.9.32), knitr, mixOmics (>= 6.26.0), pak, pkgload, rmarkdown, remotes, SHAPforxgboost (>= 0.1.0), shinytest2, stringi, testthat

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements GNU make

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

git_url https://git.bioconductor.org/packages/GExPipe

git_branch devel

git_last_commit 5bb493c

git_last_commit_date 2026-07-31

Repository Bioconductor 3.24

Date/Publication 2026-09-06

Author Safa Rafique [aut, cre] (ORCID: <https://orcid.org/0000-0003-2646-8106>), Naeem Mahmood Ashraf [aut], Prof. Dr. Muhammad Farooq Sabar [aut]

Maintainer Safa Rafique <safa.sandhu@gmail.com>

Contents

`.gexpipe_fread_counts` *Read tabular count files (GEO supplementary files are often messy)*

Description

Read tabular count files (GEO supplementary files are often messy)

Usage

```
.gexpipe_fread_counts(path, ...)
```

`.gexpipe_geo_quiet` *Capture verbose GEOquery console output without suppressMessages()*

Description

Capture verbose GEOquery console output without suppressMessages()

Usage

```
.gexpipe_geo_quiet(expr)
```

dummy_imports *Dummy function to satisfy R CMD check that packages in Imports are used*

Description

Dummy function to satisfy R CMD check that packages in Imports are used

Usage

```
dummy_imports()
```

gexpipe_analysis_report_text *Build reproducibility report text for export*

Description

Build reproducibility report text for export

Usage

```
gexpipe_analysis_report_text(params = list(), include_session = TRUE)
```

Arguments

params	Named list of scalar analysis parameters (character or numeric).
include_session	Include `sessionInfo()` block (default TRUE).

Value

Character vector of report lines.

Examples

```
gexpipe_analysis_report_text(list(method = "limma"), include_session = FALSE)
```

gexpipe_batch_confounding_summary

Summarise Dataset x Condition confounding for batch/DE guidance

Description

Summarise Dataset x Condition confounding for batch/DE guidance

Usage

```
gexpipe_batch_confounding_summary(metadata)
```

Arguments

metadata	data.frame with Dataset and Condition columns.

Value

list with confounded (logical), table (matrix), message (character)

Examples

```
meta <- data.frame(
  Dataset = rep(c("GSE1", "GSE2"), each = 3),
  Condition = rep(c("Normal", "Disease"), times = 3)
)
gexpipe_batch_confounding_summary(meta)
```

gexpipe_batch_covariate_info

Summarise how Platform should enter batch/DE models

Description

Summarise how Platform should enter batch/DE models

Usage

```
gexpipe_batch_covariate_info(metadata)
```

Arguments

metadata	sample metadata data.frame.

Value

list with mixed_platforms, platform_dataset_confounded, include_platform_covariate

Examples

```
meta <- data.frame(
  Dataset = rep(c("GSE1", "GSE2"), each = 2),
  Platform = rep(c("Microarray", "RNAseq"), each = 2)
)
gexpipe_batch_covariate_info(meta)
```

`gexpipe_build_batch_mod` *Build ComBat / removeBatchEffect model matrix (biology to preserve)*

Description

Build ComBat / removeBatchEffect model matrix (biology to preserve)

Usage

```
gexpipe_build_batch_mod(metadata)
```

Arguments

`metadata`	sample metadata data.frame.

Value

model matrix

Examples

```
meta <- data.frame(
  Condition = rep(c("Normal", "Disease"), each = 3),
  row.names = paste0("S", 1:6),
  stringsAsFactors = FALSE
)
gexpipe_build_batch_mod(meta)
```

`gexpipe_build_de_design` *Build a model matrix for DE (limma / edgeR / voom)*

Description

Build a model matrix for DE (limma / edgeR / voom)

Usage

```
gexpipe_build_de_design(metadata)
```

Arguments

`metadata`	data.frame with Dataset, Platform, Condition columns.

Value

list with design, coef_condition, formula_desc, info

Examples

```
meta <- data.frame(
  Dataset = rep(c("GSE1", "GSE2"), each = 3),
  Condition = rep(c("Normal", "Disease"), times = 3),
  row.names = paste0("S", 1:6),
  stringsAsFactors = FALSE
)
gexpipe_build_de_design(meta)
```

---

gexpipe_deseq2_design *DESeq2 design formula for count-based DE*

Description

DESeq2 design formula for count-based DE

Usage

```
gexpipe_deseq2_design(metadata)
```

Arguments

metadata	sample metadata data.frame.

Value

list with formula (formula object) and formula_desc (character)

Examples

```
meta <- data.frame(
  Condition = rep(c("Normal", "Disease"), each = 3),
  row.names = paste0("S", 1:6),
  stringsAsFactors = FALSE
)
gexpipe_deseq2_design(meta)
```

gexpipe_de_sample_info

Summarise samples used in a DE run (transparency for mixed-platform runs)

Description

Summarise samples used in a DE run (transparency for mixed-platform runs)

Usage

```
gexpipe_de_sample_info(meta_used, total_meta = NULL, method = "limma")
```

Arguments

meta_used	Metadata rows actually used in the DE fit.
total_meta	Full unified metadata before subsetting (optional).
method	DE method name.

Value

list with human-readable fields for the Shiny UI.

Examples

```
meta <- data.frame(
  Condition = rep(c("Normal", "Disease"), each = 3),
  Platform = "RNAseq",
  row.names = paste0("S", 1:6),
  stringsAsFactors = FALSE
)
gexpipe_de_sample_info(meta)
```

gexpipe_has_mixed_platforms

Detect microarray + RNA-seq in the same analysis

Description

Detect microarray + RNA-seq in the same analysis

Usage

```
gexpipe_has_mixed_platforms(metadata)
```

Arguments

metadata	data.frame with optional Platform column.

Value

logical

Examples

```
meta <- data.frame(Platform = c("Microarray", "RNAseq"))
gexpipe_has_mixed_platforms(meta)
```

---

gexpipe_independent_filter *Independent filtering for DE (limma filterByExpr)*

Description

Removes lowly expressed genes using a design-aware filter so filtering is not tied to differential expression statistics (avoids FDR bias from variance-percentile pre-filtering).

Usage

```
gexpipe_independent_filter(expr, design = NULL, group = NULL)
```

Arguments

expr	Numeric matrix (genes x samples) or integer counts.
design	Model matrix for the DE analysis.
group	Optional factor when design is NULL (single-factor designs).

Value

list with expr (filtered), keep (logical vector), n_before, n_after, note

Examples

```
expr <- matrix(stats::rpois(5000, lambda = 50), nrow = 500, ncol = 10)
design <- stats::model.matrix(~ factor(rep(c("A", "B"), each = 5)))
gexpipe_independent_filter(expr, design = design)
```

---

gexpipe_pca_polar_df *Polar PCA coordinates for batch/platform diagnostic plots*

Description

Polar PCA coordinates for batch/platform diagnostic plots

Usage

```
gexpipe_pca_polar_df(expr, metadata, color_by = "Dataset")
```

Arguments

expr	genes x samples matrix.
metadata	sample metadata aligned to expr columns.
color_by	column name in metadata to colour points.

Value

data.frame with theta, r, and colour column.

Examples

```
expr <- matrix(rnorm(120), 20, 6, dimnames = list(paste0("G", 1:20), paste0("S", 1:6)))
meta <- data.frame(Dataset = rep(c("A", "B"), each = 3), row.names = colnames(expr))
gexpipe_pca_polar_df(expr, meta)
```

---

gexpipe_platform_dataset_confounded

Test whether Platform is not estimable alongside Dataset

Description

Returns TRUE when Platform is nested within Dataset (each GSE has one technology) or when a tentative ~ Dataset + Platform (+ Condition) design is rank-deficient.

Usage

```
gexpipe_platform_dataset_confounded(metadata)
```

Arguments

metadata	data.frame with Dataset and Platform columns.

Value

logical

Examples

```
meta <- data.frame(
  Dataset = c("GSE1", "GSE1", "GSE2", "GSE2"),
  Platform = c("Microarray", "Microarray", "RNAseq", "RNAseq")
)
gexpipe_platform_dataset_confounded(meta)
```

gexpipe_pvca_df *Simplified PVCA variance bar-chart data (PCA + per-factor R^2 on top PCs)*

Description

Aligns metadata to expression columns, runs PCA on samples, and estimates the mean fraction of variance in the top PCs explained by Dataset, Platform, and Condition. Residual is `1 - sum(factor fractions)` (approximate).

Usage

```
gexpipe_pvca_df(expr, metadata, max_samples = 100L, max_pcs = 10L)
```

Arguments

`expr`	Numeric matrix, genes x samples.
`metadata`	Sample metadata with `Dataset` and optional `Platform`, `Condition`.
`max_samples`	Subsample at most this many columns for speed (default 100).
`max_pcs`	Use at most this many principal components (default 10).

Value

A list with `ok` (logical), `data` (data.frame or `NULL`), and `message` (character, empty on success).

Examples

```
expr <- matrix(rnorm(120), 20, 6, dimnames = list(paste0("G", 1:20), paste0("S", 1:6)))
meta <- data.frame(
  Dataset = rep(c("GSE1", "GSE2"), each = 3),
  Condition = rep(c("Normal", "Disease"), times = 3),
  row.names = colnames(expr),
  stringsAsFactors = FALSE
)
gexpipe_pvca_df(expr, meta)
```

gexpipe_setup *Set up GExPipe dependencies and optionally launch the app*

Description

Detects your R version and verifies (or optionally installs) required packages. When `GExPipe` is installed from Bioconductor, dependencies are installed at package install time; `gexpipe_setup()` only checks availability unless `options(gexpipe.auto_install = TRUE)` is set.

Usage

```
gexpipe_setup(
  update = TRUE,
  optional = TRUE,
  launch = FALSE,
  port = 3838L,
  ...
)
```

Arguments

update	Logical. If TRUE (default), update already-installed packages when auto-install is enabled.
optional	Logical. Reserved for future optional feature packages. Currently all runtime dependencies are in DESCRIPTION Imports and are always installed regardless of this flag.
launch	Logical. If TRUE, launch the GExPipe Shiny app after setup completes. Default FALSE.
port	Integer. Port for the Shiny app when launch = TRUE.
...	Additional arguments passed to shiny::runApp() when launch = TRUE.

Value

Invisibly returns a named logical vector indicating which packages were successfully installed/available.

Examples

```
# Check and install missing packages only (no updates, no launch)
if (interactive()) {
  gexpipe_setup(update = FALSE, optional = FALSE, launch = FALSE)
}
```

---

gexpipe_wgcna_heatmap_cor

Drop redundant combined trait column from WGCNA module-trait correlation matrix

---

Description

For a two-group design, WGCNA may add a combined "X vs Y" contrast column that duplicates one group indicator. This helper removes that column before plotting.

Usage

```
gexpipe_wgcna_heatmap_cor(cor_mat, combined)
```

Arguments

cor_mat	Numeric matrix of module-trait correlations.
combined	Name of the combined contrast column to drop, or NULL.

Value

The correlation matrix with the combined column removed when appropriate.

Examples

```
cor_mat <- matrix(runif(6), 2, 3, dimnames = list(c("ME1", "ME2"), c("A", "B", "A vs B")))
gexpipe_wgcna_heatmap_cor(cor_mat, "A vs B")
```

`gexp_align_rnaseq_sample_names` *Align RNA-seq count-matrix column names with GEO sample metadata*

Description

GExPipe users enter **GSE accessions** in the app; this helper runs automatically during GEO download when a count file has no sample headers (e.g. `data.table::fread` assigns `V2`, `V3`, ...). Columns are renamed to GSM IDs from GEO pData so QC, normalization, and group selection see individual samples (fixes issues such as a single **V2** bar in outlier QC).

Usage

```
gexp_align_rnaseq_sample_names(count_matrix, metadata = NULL, gse_id = NULL)
```

Arguments

`count_matrix`	Numeric matrix (genes x samples).
`metadata`	Optional GEO pData with sample IDs as row names.
`gse_id`	GEO series accession (used for fallback naming).

Value

Matrix with improved column names.

Examples

```
# GSE137136-style: headerless columns after download, GSM IDs in metadata.
mat <- matrix(1:4, nrow = 2, ncol = 2,
              dimnames = list(c("GENE1", "GENE2"), c("V2", "V3")))
meta <- data.frame(title = c("sample 1", "sample 2"),
                   row.names = c("GSM111", "GSM222"))
out <- gexp_align_rnaseq_sample_names(mat, meta, "GSE137136")
colnames(out)
```

gexp_batch_correct *Variance-based gene filtering and batch correction*

Description

This helper wraps the Step 5 logic:

- filters genes by a variance percentile,
- applies one of several batch-correction methods,
- reports gene reduction statistics.

Usage

```
gexp_batch_correct(
  expr,
  metadata,
  variance_percentile = 25,
  method = c("combat_ref", "sva", "limma", "combat", "quantile_limma", "hybrid")
)
```

Arguments

expr	Combined expression matrix (genes x samples), after normalization.
metadata	Data.frame with at least columns Dataset and Condition.
variance_percentile	Numeric between 0 and 50; bottom percentile to remove.
method	Batch method: one of "limma", "combat", "combat_ref", "quantile_limma", "hybrid", "sva".

Value

A list with elements:

expr_filtered	filtered expression matrix before batch correction
batch_corrected	batch-corrected matrix
genes_before	integer, genes before filtering
genes_after	integer, genes after filtering
filter_percent	numeric, percent genes removed
log_text	character, human-readable log string

Examples

```
expr <- matrix(rnorm(200), nrow = 20)
rownames(expr) <- paste0("Gene", seq_len(nrow(expr)))
colnames(expr) <- paste0("S", seq_len(ncol(expr)))
metadata <- data.frame(
  Dataset = rep(c("D1", "D2"), each = 5),
  Condition = rep(c("Normal", "Disease"), times = 5),
  row.names = colnames(expr),
```

```
  stringsAsFactors = FALSE
)
out <- gexp_batch_correct(expr, metadata, variance_percentile = 10, method = "limma")
dim(out$batch_corrected)
```

`gexp_download_finalize_common_genes` *Finalize common genes and combined matrix after download/mapping*

Description

Finalize common genes and combined matrix after download/mapping

Usage

```
gexp_download_finalize_common_genes(
  micro_expr_list,
  rna_counts_list,
  all_genes_list
)
```

Arguments

`micro_expr_list`	Named list of microarray matrices (genes x samples).
`rna_counts_list`	Named list of RNA-seq matrices (genes x samples).
`all_genes_list`	Named list of rowname vectors per dataset.

Value

List with updated lists, common_genes, combined_expr_raw, and status.

Examples

```
m1 <- matrix(1:12, nrow = 3, dimnames = list(c("A", "B", "C"), paste0("S", 1:4)))
m2 <- matrix(1:12, nrow = 3, dimnames = list(c("B", "C", "D"), paste0("T", 1:4)))
out <- gexp_download_finalize_common_genes(
  micro_expr_list = list(D1 = m1),
  rna_counts_list = list(D2 = m2),
  all_genes_list = list(D1 = rownames(m1), D2 = rownames(m2))
)
out$common_genes
```

`gexp_download_normalize_ids_for_overlap` *Normalize dataset row IDs to gene symbols for overlap*

Description

Normalize dataset row IDs to gene symbols for overlap

Usage

```
gexp_download_normalize_ids_for_overlap(
  micro_expr_list,
  rna_counts_list,
  platform_per_gse = NULL,
  all_genes_list = NULL,
  micro_eset_list = NULL
)
```

Arguments

`micro_expr_list`	Named list of microarray matrices.
`rna_counts_list`	Named list of RNA-seq matrices.
`platform_per_gse`	Optional named list/vector of GPL IDs.
`all_genes_list`	Optional prebuilt gene lists.
`micro_eset_list`	Optional named list of `ExpressionSet` objects for `fData` fallback when probe-to-symbol mapping needs platform annotation.

Value

List with updated lists and appended log text.

Examples

```
m <- matrix(1:6, nrow = 2, dimnames = list(c("A", "B"), paste0("S", 1:3)))
r <- matrix(1:6, nrow = 2, dimnames = list(c("B", "C"), paste0("T", 1:3)))
out <- gexp_download_normalize_ids_for_overlap(
  micro_expr_list = list(M = m),
  rna_counts_list = list(R = r)
)
names(out)
```

gexp_download_one_microarray_gse

Download and parse one microarray GSE

Description

Download and parse one microarray GSE

Usage

```
gexp_download_one_microarray_gse(gse_id, micro_dir)
```

Arguments

gse_id	GEO series ID.
micro_dir	Directory for supplementary files.

Value

List with status, log text, reason, expression, metadata, eset, platform_id, and cel_paths.

Examples

```
if (interactive()) {
  out <- gexp_download_one_microarray_gse("GSE10072", tempdir())
  out$ok
}
```

gexp_download_one_rnaseq_gse

Download and parse one RNA-seq GSE

Description

Download and parse one RNA-seq GSE

Usage

```
gexp_download_one_rnaseq_gse(gse_id, rna_dir)
```

Arguments

gse_id	GEO series ID.
rna_dir	Directory containing rna_data.

Value

List with status, reason, log text, count matrix, and metadata.

Examples

```
if (interactive()) {
  out <- gexp_download_one_rnaseq_gse("GSE50760", tempdir())
  out$ok
}
```

`gexp_ensure_unique_colnames_across_datasets`
Prefix duplicate sample column names across multiple datasets

---

Description

Prefix duplicate sample column names across multiple datasets

Usage

```
gexp_ensure_unique_colnames_across_datasets(expr_lists)
```

Arguments

`expr_lists`	Named list of expression/count matrices.

Value

Updated list with unique column names where needed.

`gexp_fetch_geo_series_matrix_metadata`
Fetch sample metadata from GEO series matrix fallback

Description

Fetch sample metadata from GEO series matrix fallback

Usage

```
gexp_fetch_geo_series_matrix_metadata(gse_id)
```

Arguments

`gse_id`	GEO series ID.

Value

data.frame or NULL.

Examples

```
if (interactive()) {
  md <- gexp_fetch_geo_series_matrix_metadata("GSE10072")
  if (!is.null(md)) head(md)
}
```

gexp_is_generic_sample_names

Detect fread-style generic column names (V1, V2, X1, ...)

Description

Detect fread-style generic column names (V1, V2, X1, ...)

Usage

```
gexp_is_generic_sample_names(nms)
```

Arguments

nms	Character vector of sample/column names.

Value

Logical scalar.

gexp_normalize_and_intersect

Normalize microarray and RNA-seq datasets and compute common genes

Description

This is a non-Shiny helper version of the Step 3 logic:

- per-dataset normalization (microarray + RNA-seq),
- intersection of gene sets (common genes),
- global quantile normalization of the combined matrix,
- optional extraction of RNA-seq raw counts for count-based DE.

Usage

```
gexp_normalize_and_intersect(
  micro_expr_list,
  rna_counts_list,
  micro_norm_method = "quantile",
  rnaseq_norm_method = "TMM",
  micro_cel_paths = NULL,
  platform_per_gse = NULL,
  micro_eset_list = NULL,
  de_method = "limma",
  apply_global_quantile = TRUE
)
```

Arguments

micro_expr_list	named list of microarray expression matrices.
rna_counts_list	named list of RNA-seq count matrices.
micro_norm_method	"quantile" or "rma" (RMA requires CEL paths).
rnaseq_norm_method	"TMM" or "log2cpm_only".
micro_cel_paths	optional named list of CEL paths per GSE (for RMA).
platform_per_gse	optional named vector giving platform IDs per GSE.
micro_eset_list	optional named list of ExpressionSet objects per GSE (for RMA ID mapping).
de_method	differential expression method (used to decide whether to save raw counts for DESeq2/edgeR/limma-voom); one of "limma", "limma_voom", "deseq2", "edger".
apply_global_quantile	logical; if TRUE, apply limma quantile normalization across all samples.

Value

A list with elements:

combined_expr	globally quantile-normalized matrix (genes x samples)
combined_expr_before_global	matrix before global quantile
all_expr_norm_list	normalized per-dataset matrices (common genes only)
common_genes	character vector of common genes
normalization_stats	list of per-dataset normalization info
normalization_summary_table	data.frame summarizing gene counts
raw_counts_for_deseq2	integer matrix of RNA-seq counts (optional)
raw_counts_metadata	data.frame with sample metadata for raw counts (optional)
unified_metadata	data.frame with SampleID, Platform, Dataset, Condition=NA
log_text	character string with a human-readable log

Examples

```
out <- withr::with_seed(1, {
  m1 <- matrix(abs(rnorm(120)), nrow = 20, ncol = 6)
  m2 <- matrix(abs(rnorm(120)), nrow = 20, ncol = 6)
  rownames(m1) <- rownames(m2) <- paste0("Gene", seq_len(20))
  colnames(m1) <- paste0("D1_S", seq_len(6))
```

```
  colnames(m2) <- paste0("D2_S", seq_len(6))
  gexp_normalize_and_intersect(
    micro_expr_list = list(D1 = m1, D2 = m2),
    rna_counts_list = list(),
    de_method = "limma"
  )
})
dim(out$combined_expr)
```

`gexp_no_common_genes_diagnostic_log` *Build diagnostic log text when no common genes are found*

Description

Build diagnostic log text when no common genes are found

Usage

```
gexp_no_common_genes_diagnostic_log(all_genes_list)
```

Arguments

`all_genes_list` Named list of row IDs per dataset.

Value

Character string for log appending.

Examples

```
txt <- gexp_no_common_genes_diagnostic_log(
  list(D1 = c("1007_s_at", "1053_at"), D2 = c("ENSG000001", "ENSG000002"))
)
nchar(txt) > 0
```

`gexp_orient_count_dataframe` *Orient an RNA-seq count table to a genes x samples matrix*

Description

Some GEO supplementary files store samples as rows and genes as columns. This helper transposes when dimensions and optional metadata suggest that layout.

Usage

```
gexp_orient_count_dataframe(count_df, metadata = NULL)
```

Arguments

count_df	data.frame read from a count file.
metadata	Optional GEO pData used to hint expected sample count.

Value

List with matrix (genes x samples) and log (character).

---

gexp_parse_gse_inputs *Parse GSE IDs from Shiny text inputs*

Description

Parse GSE IDs from Shiny text inputs

Usage

```
gexp_parse_gse_inputs(
  analysis_type,
  rnaseq_gses = "",
  microarray_gses = "",
  dataset_mode = "multi"
)
```

Arguments

analysis_type	One of "rnaseq", "microarray", or "merged".
rnaseq_gses	Text field for RNA-seq IDs.
microarray_gses	Text field for microarray IDs.
dataset_mode	"single" or "multi".

Value

List with rnaseq_ids, micro_ids, dataset_mode, and messages.

Examples

```
x <- gexp_parse_gse_inputs(
  analysis_type = "merged",
  rnaseq_gses = "GSE1, GSE2",
  microarray_gses = "GSE3",
  dataset_mode = "single"
)
x$rnaseq_ids
```

gexp_prepare_download_dirs

Prepare clean download directories for current run

Description

Prepare clean download directories for current run

Usage

```
gexp_prepare_download_dirs(
  base_dir = getwd(),
  has_micro = FALSE,
  has_rna = FALSE
)
```

Arguments

base_dir	Base working directory.
has_micro	Logical; whether microarray IDs exist.
has_rna	Logical; whether RNA-seq IDs exist.

Value

Character vector log lines describing cleanup.

Examples

```
td <- tempdir()
gexp_prepare_download_dirs(td, has_micro = TRUE, has_rna = TRUE)
```

gexp_qc_build_sample_dataset_map

Map combined-expression sample IDs to source GSE datasets

Description

Used by QC outlier plots before normalization builds unified metadata.

Usage

```
gexp_qc_build_sample_dataset_map(micro_expr_list, rna_counts_list)
```

Arguments

micro_expr_list	Named list of microarray matrices.
rna_counts_list	Named list of RNA-seq matrices.

Value

Named character vector: sample ID -> GSE accession.

Examples

```
m <- matrix(1:4, nrow = 2, ncol = 2, dimnames = list(c("A", "B"), c("S1", "S2")))
r <- matrix(1:4, nrow = 2, ncol = 2, dimnames = list(c("A", "B"), c("S3", "S4")))
gexp_qc_build_sample_dataset_map(list(GSE1 = m), list(GSE2 = r))
```

---

gexp_qc_detect_outliers

Detect sample outliers from expression matrix

Description

Uses two methods on the top variable genes:

- PCA Mahalanobis distance (97.5% chi-square cutoff),
- Sample connectivity in signed network (power = 6; mean - 2*SD threshold).

Usage

```
gexp_qc_detect_outliers(expr, top_n = 5000L)
```

Arguments

expr	Numeric matrix with genes in rows and samples in columns.
top_n	Integer number of top variable genes to use (default 5000).

Value

List with outlier detection results.

Examples

```
expr <- matrix(rnorm(400), nrow = 40, ncol = 10)
rownames(expr) <- paste0("Gene", seq_len(nrow(expr)))
colnames(expr) <- paste0("S", seq_len(ncol(expr)))
out <- gexp_qc_detect_outliers(expr, top_n = 20)
length(out$all_outliers)
```

gexp_qc_exclude_samples *Exclude selected samples from download/QC state lists*

Description

Exclude selected samples from download/QC state lists

Usage

```
gexp_qc_exclude_samples(
  combined_expr_raw,
  micro_expr_list,
  rna_counts_list,
  unified_metadata = NULL,
  samples_to_exclude
)
```

Arguments

combined_expr_raw	Matrix genes x samples.
micro_expr_list	Named list of microarray matrices.
rna_counts_list	Named list of RNA-seq matrices.
unified_metadata	Optional metadata data.frame with SampleID column.
samples_to_exclude	Character vector of sample IDs.

Value

List containing updated matrices/lists/metadata.

Examples

```
expr <- matrix(rnorm(300), nrow = 30, ncol = 10)
rownames(expr) <- paste0("Gene", seq_len(nrow(expr)))
colnames(expr) <- paste0("S", seq_len(ncol(expr)))
meta <- data.frame(SampleID = colnames(expr), stringsAsFactors = FALSE)
res <- gexp_qc_exclude_samples(
  combined_expr_raw = expr,
  micro_expr_list = list(D1 = expr[, 1:5, drop = FALSE]),
  rna_counts_list = list(D2 = expr[, 6:10, drop = FALSE]),
  unified_metadata = meta,
  samples_to_exclude = c("S1", "S10")
)
ncol(res$combined_expr_raw)
```

gexp_qc_gene_overlap_summary *Build gene-overlap summary table for QC UI*

Description

Build gene-overlap summary table for QC UI

Usage

```
gexp_qc_gene_overlap_summary(all_genes_list, common_genes)
```

Arguments

all_genes_list	Named list of gene vectors by dataset.
common_genes	Character vector of common genes.

Value

Data.frame with Dataset, Total_Genes, Pct_in_Common.

Examples

```
all_genes <- list(
  D1 = c("A", "B", "C", "D"),
  D2 = c("B", "C", "E")
)
gexp_qc_gene_overlap_summary(all_genes, common_genes = c("B", "C"))
```

gexp_qc_prepare_boxplot_data *Prepare QC boxplot data from combined expression*

Description

Prepare QC boxplot data from combined expression

Usage

```
gexp_qc_prepare_boxplot_data(
  combined_expr_raw,
  micro_expr_list,
  rna_counts_list,
  max_points = 500000L
)
```

Arguments

combined_expr_raw	Matrix genes x samples.
micro_expr_list	Named list of microarray matrices.
rna_counts_list	Named list of RNA-seq matrices.
max_points	Maximum rows to keep for plotting.

Value

data.frame with Expression, Sample, Platform.

Examples

```
expr <- matrix(rnorm(120), nrow = 12, ncol = 10)
rownames(expr) <- paste0("Gene", seq_len(nrow(expr)))
colnames(expr) <- paste0("S", seq_len(ncol(expr)))
df <- gexp_qc_prepare_boxplot_data(
  combined_expr_raw = expr,
  micro_expr_list = list(D1 = expr[, 1:5, drop = FALSE]),
  rna_counts_list = list(D2 = expr[, 6:10, drop = FALSE]),
  max_points = 1000
)
head(df)
```

---

gexp_qc_prepare_density_data

Prepare density curves for QC density plot

Description

Prepare density curves for QC density plot

Usage

```
gexp_qc_prepare_density_data(combined_expr_raw, max_samples = 50L)
```

Arguments

combined_expr_raw	Matrix genes x samples.
max_samples	Maximum number of sample density lines to include.

Value

List with first density, additional densities, and colors.

Examples

```
expr <- matrix(rnorm(200), nrow = 20, ncol = 10)
d <- gexp_qc_prepare_density_data(expr, max_samples = 5)
length(d$others)
```

gexp_qc_prepare_upset_data *Prepare UpSet matrix data from per-dataset genes*

Description

Prepare UpSet matrix data from per-dataset genes

Usage

```
gexp_qc_prepare_upset_data(all_genes_list)
```

Arguments

all_genes_list	Named list of gene vectors by dataset.

Value

List with ok, upset_df, gene_lists, max_set_size, and message.

Examples

```
sets <- list(
  D1 = c("A", "B", "C"),
  D2 = c("B", "C", "D")
)
out <- gexp_qc_prepare_upset_data(sets)
names(out)
```

gexp_qc_prepare_venn_sets *Prepare cleaned gene sets for Venn plotting*

Description

Prepare cleaned gene sets for Venn plotting

Usage

```
gexp_qc_prepare_venn_sets(all_genes_list, max_sets = 5L)
```

Arguments

all_genes_list	Named list of gene vectors by dataset.
max_sets	Maximum number of sets to keep for Venn plotting.

Value

List with ok, sets, and message.

Examples

```
sets <- list(
  D1 = c("A", "B", "C"),
  D2 = c("B", "C", "D"),
  D3 = c("A", "D")
)
out <- gexp_qc_prepare_venn_sets(sets)
out$ok
```

`gexp_rebuild_all_genes_list` *Rebuild per-dataset gene lists from expression/count lists*

Description

Rebuild per-dataset gene lists from expression/count lists

Usage

```
gexp_rebuild_all_genes_list(micro_expr_list, rna_counts_list)
```

Arguments

`micro_expr_list`
: Named list of microarray matrices.

`rna_counts_list`
: Named list of RNA-seq matrices.

Value

Named list of gene vectors.

Examples

```
m <- matrix(1:6, nrow = 2, dimnames = list(c("A", "B"), paste0("S", 1:3)))
r <- matrix(1:6, nrow = 2, dimnames = list(c("B", "C"), paste0("T", 1:3)))
gexp_rebuild_all_genes_list(list(M = m), list(R = r))
```

`gexp_run_de` *Run differential expression analysis*

Description

Run differential expression analysis

Usage

```
gexp_run_de(
  expr,
  metadata,
  method = c("limma", "limma_voom", "deseq2", "edger"),
  logfc_cutoff = 0.5,
  padj_cutoff = 0.05
)
```

Arguments

expr	Matrix of (batch-corrected, normalized) expression values with genes in rows and samples in columns.
metadata	Data.frame with at least a Condition column.
method	Currently only "limma" is implemented.
logfc_cutoff	Numeric log2 fold-change cutoff.
padj_cutoff	Numeric adjusted P-value cutoff.

Value

list with de_results, sig_genes, filter_note, sample_info, formula_desc

Examples

```
expr <- matrix(rnorm(200), nrow = 20)
rownames(expr) <- paste0("G", seq_len(nrow(expr)))
colnames(expr) <- paste0("S", seq_len(ncol(expr)))
meta <- data.frame(
  Condition = rep(c("Normal", "Disease"), each = 5),
  row.names = colnames(expr),
  stringsAsFactors = FALSE
)
gexp_run_de(expr, meta)
```

---

gexp_wgcna_prepare *Prepare expression and sample data for WGCNA*

Description

This helper mirrors the "Prepare WGCNA Data" step:

- uses batch-corrected expression (or combined expression if needed),
- drops samples/genes with too many missing values,
- optionally filters to top variable genes,
- enforces a minimum fraction of non-missing samples per gene,
- returns a WGCNA-ready datExpr matrix (samples x genes) and companion sample info and gene-variance table.

Usage

```
gexp_wgcna_prepare(
  expr,
  metadata,
  gene_mode = c("top_variable", "all_common"),
  top_genes = 5000L,
  min_samples_frac = 0.5
)
```

Arguments

expr	Matrix of expression values (genes x samples), typically batch-corrected.
metadata	Data.frame with sample metadata; should contain either rownames matching `colnames(expr)` or a `SampleID` column.
gene_mode	"all_common" or "top_variable" (default).
top_genes	If `gene_mode = "top_variable"`, the number of most variable genes to select (default 5000).
min_samples_frac	Minimum fraction of samples with non-missing expression required for a gene to be kept (default 0.5).

Value

A list with elements:

datExpr	numeric matrix (samples x genes) for WGCNA
sample_info	data.frame with sample metadata aligned to datExpr
gene_variance_table	data.frame with gene variance and rank

Examples

```
expr <- matrix(rnorm(600), nrow = 60)
rownames(expr) <- paste0("Gene", seq_len(nrow(expr)))
colnames(expr) <- paste0("S", seq_len(ncol(expr)))
metadata <- data.frame(
  SampleID = colnames(expr),
  Condition = rep(c("Normal", "Disease"), each = 5),
  Dataset = rep(c("D1", "D2"), each = 5),
  stringsAsFactors = FALSE
)
prep <- gexp_wgcna_prepare(expr, metadata, gene_mode = "top_variable", top_genes = 50)
dim(prep$datExpr)
```

`runGExPipe` *Run the GExPipe Shiny application*

Description

Launches the GExPipe Shiny app for multi-omics RNA analysis (bulk RNA-seq, microarray, GEO download, QC, normalization, differential expression, WGCNA, pathway enrichment, PPI, machine learning, and more).

Usage

```
runGExPipe(
  launch.browser = TRUE,
  port = getOption("shiny.port", 3838),
  host = getOption("shiny.host", "127.0.0.1")
)
```

Arguments

`launch.browser`	If TRUE, open the app in the default browser (default). Set to FALSE in Google Colab or headless servers (no local browser).
`port`	TCP port for Shiny. Use a **positive** port (e.g. `3838`) when you will open the app manually in a browser. `0` means "pick a free port" (fine for automated tests); some setups print `http://127.0.0.1:0`, which **cannot** be pasted into a browser - then use `launch.browser = TRUE` or `port = 3838`.
`host`	The host to bind. Use "0.0.0.0" in Google Colab so port forwarding works.

Value

A Shiny app object. Users should run it with `shiny::runApp()`, which starts the server and **blocks** the R session until the app is stopped (e.g. RStudio Stop button).

Startup time

When `GExPipe` is installed from Bioconductor (or any normal R library), dependencies must be installed at package install time via `BiocManager::install("GExPipe", dependencies = TRUE)`. `runGExPipe()` then verifies imports and opens the app without downloading packages.

For the GitHub / first-run workflow, set `options(gexpipe.auto_install = TRUE)` before `runGExPipe()` to enable background dependency installation (10-30 minutes on a fresh machine).

The welcome screen loads first; the full 15-tab dashboard is built only after the user clicks **Go to Analysis**, so the initial browser response is fast even on slow networks. Prefer `GExPipe::runGExPipe()` from an **installed** package rather than opening `inst/shinyapp/` in RStudio directly (that path runs `global.R`, which duplicates the install logic).

Optional tuning before `runGExPipe()`:

- `options(gexpipe.wgcna_threads = 1L)` - fewer WGCNA threads (less parallel setup noise).
- `options(gexpipe.wgcna_threads = 0L)` or `FALSE` - skip `WGCNA::enableWGCNAThreads()` at app start.

- `options(shiny.testmode = TRUE)` with `options(gexpipe.minimal_attach_in_testmode = TRUE)` (default when unset) attaches only the Shiny stack first, then the rest after the first flush, so automated tests can connect quickly. Set `gexpipe.minimal_attach_in_testmode = FALSE` for a full attach on the first tick (e.g. deep `shinytest2` scenarios).

PPI (STRINGdb)

Install with full dependencies so STRINGdb support packages resolve: `BiocManager::install("GExPipe", dependencies = TRUE)`. STRINGdb is installed as a dependency. The first PPI run may download large STRING files (requires network). If initialization fails, try `options(gexpipe.stringdb_try_versions = c("11.5", "11", "12"))` or update STRINGdb.

Examples

```
# Non-interactive (R CMD check / example()): build an app object only.
# Dependency auto-install is skipped when interactive() is FALSE, so this
# runs quickly without touching the network.
app <- runGExPipe(launch.browser = FALSE, port = 0L)
stopifnot(inherits(app, "shiny.appobj"))

# Interactive: two-step launch. runGExPipe() builds the app object
# (installing any missing packages on first run); shiny::runApp() starts it.
if (interactive()) {
  app <- GExPipe::runGExPipe()      # Step 1: build the app object
  shiny::runApp(app, port = 3838L)  # Step 2: start the server
}
```

Index