Package 'damidBind'

January 15, 2026

Type Package

Title Differential Binding and Expression Analysis for DamID-seq Data

Version 0.99.12

Description The damidBind package provides a straightforward formal analysis pipeline to analyse and explore differential DamID binding, gene transcription or chromatin accessibility between two conditions. The package imports processed data from DamID-seq experiments, either as external raw files in the form of binding bedGraphs and GFF/BED peak calls, or as internal lists of GRanges objects. After optionally normalising data, combining peaks across replicates and determining per-replicate peak occupancy, the package links bound loci to nearby genes. For RNA Polymerase DamID data, the package calculates occupancy over genes, and optionally calcualates the FDR of significantly-enriched gene occupancy.
damidBind then uses either limma (for conventional log2 ratio DamID binding data) or NOIseq (for counts-based CATaDa chromatin accessibility data) to identify differentially-enriched regions, or differentially epxressed genes, between two conditions.
The package provides a number of visualisation tools (volcano plots, Gene Ontology enrichment plots via ClusterProfiler and proportional Venn diagrams via BioVenn for downstream data exploration and analysis. An powerful, interactive IGV genome browser interface (powered by Shiny and igvShiny) allows users to rapidly and intuitively assess significant differentially-bound regions in their genomic context.

License GPL-3

Encoding UTF-8

biocViews DifferentialExpression, GeneExpression, Transcription, Epigenetics, Visualization, Sequencing, Software, GeneRegulation

Depends R (>= 4.4.0)

Imports ggplot2, ggrepel, dplyr, tibble, stringr, tools, rlang, BiocParallel, AnnotationHub, DBI, ensembldb, GenomeInfoDb, IRanges, GenomicRanges, S4Vectors, rtracklayer, limma, NOISeq, BioVenn, clusterProfiler, enrichplot, forcats, scales, colorspace, ggnewscale, methods, stats, igvShiny, shiny, DT, dbscan, circlize, ComplexHeatmap, patchwork, splines

Suggests testthat, curl, knitr, htmltools, rmarkdown, BiocStyle, bookdown, org.Dm.eg.db

RoxygenNote 7.3.3

VignetteBuilder knitr

URL https://marshall-lab.org/damidBind

BugReports https://github.com/marshall-lab/damidBind/issues

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

git_branch devel

git_last_commit b185a6d

git_last_commit_date 2026-01-06

Repository Bioconductor 3.23

Date/Publication 2026-01-15

Author Owen Marshall [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1605-3871>)

Maintainer Owen Marshall <owen.marshall@utas.edu.au>

Contents

damidBind-package *damidBind: Differential Binding Analysis for DamID-seq Data*

Description

The damidBind package provides a streamlined workflow for determining differential protein binding, RNA polymerase occupancy, or chromatin accessibility from DamID-based sequencing experiments. It handles data loading, processing, statistical analysis, and provides a suite of visualisation tools for interpretation and exploration of the results.

Details

The package is designed for three main experimental types:

- **Transcription Factor Binding**: Analysis of conventional DamID or TaDa data to find differential binding sites for a protein of interest. Uses log-ratio data and the limma backend via `differential_binding`.
- **Gene Transcription**: Analysis of RNA Polymerase II TaDa data to infer differential gene expression, analysed over gene bodies with `load_data_genes`.
- **Chromatin Accessibility**: Analysis of CATaDa data to find differential accessibility. Uses count-based data and the NOIseq backend via `differential_accessibility`.

Core Workflow:

1. **Load Data:** Use `load_data_peaks` for TF binding/accessibility or `load_data_genes` for RNA Pol II occupancy. These functions read bedGraph and peak files, calculate occupancy scores, and annotate regions with nearby genes.
2. **Perform Differential Analysis:** Use `differential_binding` for conventional DamID log-ratio data (limma) or `differential_accessibility` for CATaDa count data (NOIseq). These return a `DamIDResults` object.
3. **Visualise and Explore:** Use the plotting functions on the 'DamIDResults' object: `plot_volcano`, `plot_venn` and `analyse_go_enrichment`; and the interactive `browse_igv_regions` to browse differentially-bound regions in an interactive IGV browser window..

For a complete walkthrough, please see the package vignette by running: 'browseVignettes("damidBind")'

Author(s)

Maintainer: Owen Marshall <owen.marshall@utas.edu.au> (ORCID)

See Also

Primary Functions:

- `load_data_peaks`: Load binding data and associated peak regions.
- `load_data_genes`: Load binding data summarised over gene bodies.
- `differential_binding`: Perform differential analysis for log-ratio data.

- differential_accessibility: Perform differential analysis for count-based data.
- DamIDResults: The main results object returned by analysis functions.

Useful links:

- The damidBind vignette: ‘browseVignettes("damidBind")‘
- Report bugs at https://github.com/marshall-lab/damidBind/issues

---

`analyse_go_enrichment` *Perform Gene Ontology (GO) enrichment analysis for differentially bound/expressed regions*

Description

This function performs Gene Ontology (GO) enrichment analysis using ‘clusterProfiler‘ for either the up-regulated or down-regulated regions/genes identified by ‘differential_binding()‘ or ‘differential_accessibility()‘. It automatically extracts the relevant gene IDs (gene_ids) and the background universe from the input ‘DamIDResults‘ object.

Usage

```
analyse_go_enrichment(
  diff_results,
  direction = "cond1",
  org_db = org.Dm.eg.db::org.Dm.eg.db,
  ontology = "BP",
  pvalue_cutoff = 0.05,
  qvalue_cutoff = 0.2,
  plot_title = NULL,
  show_category = 12,
  label_format_width = 50,
  wrap_labels = FALSE,
  fit_labels = FALSE,
  abbrev_terms = FALSE,
  abbrevs = c(regulation = "reg."),
  theme_size = 14,
  use_gse = FALSE,
  save = NULL,
  save_results_path = NULL,
  maxGSSize = 1000,
  minGSSize = 10,
  clean_gene_symbols = TRUE
)
```

Arguments

diff_results	A 'DamIDResults' object, as returned by 'differential_binding()' or 'differential_accessibility()'.
direction	Character string. Specifies which set of genes to analyse, either using condition names, "cond1" or "cond2", or "all" (for all significantly enriched genes from either direction). Default is "cond1".
org_db	An OrgDb object specifying the organism's annotation database. For Drosophila, use 'org.Dm.eg.db::org.Dm.eg.db'.
ontology	Character string. The GO ontology to use: "BP" (Biological Process), "MF" (Molecular Function), or "CC" (Cellular Component). Default is "BP".
pvalue_cutoff	Numeric. Adjusted p-value cutoff for significance. Default: 0.05.
qvalue_cutoff	Numeric. Q-value cutoff for significance. Default: 0.2.
plot_title	Character string. Title for the generated dot plot.
show_category	Integer. Number of top enriched GO categories to display in the plot. Default: 12.
label_format_width	Integer. Max character length for GO term labels on the plot before wrapping. Default: 50.
wrap_labels	Logical. Whether to wrap label text (TRUE) or truncate (FALSE) if greater than 'label_format_width' (Default: FALSE)
fit_labels	Set 'label_format_width' to the largest label width (Default: FALSE)
abbrev_terms	Logical. Whether to abbreviate common GO term words. (Default: FALSE)
abbrevs	Named vector of abbreviations to use for 'abbrev_terms'. (Default: 'c("regulation" = "reg.")')
theme_size	Integer. Base theme size to set. Default: 14.
use_gse	Logical. Whether to use GSEA via 'gseGO' rather than GO enrichment analysis (via 'enrichGO'). (Default: FALSE)
save	List or 'NULL'. Controls saving the plot to a file (dot plot). If 'NULL', 'FALSE', or '0', the plot is not saved. If a 'list', it specifies saving parameters: • filename (character): The path and base name for the output file. If not specified, the default name "damidBind_GSEA_dotplot" is used. • format (character): File format ("pdf", "svg", or "png"). Default is "pdf". • width (numeric): Width of the plot in inches. Default is 6. • height (numeric): Height of the plot in inches. Default is 6.
save_results_path	Character string or NULL. If a path is provided (e.g., "go_results.csv"), the enrichment results table will be saved to this CSV file.
maxGSSize	Integer. Maximum size of gene sets to consider. Default: 1000.
minGSSize	Integer. Minimum size of gene sets to consider. Default: 10.
clean_gene_symbols	Logical. Removes snoRNAs and tRNAs (common sources of accidental bias between different NGS methods) from the gene lists prior to enrichment analysis. Default: TRUE.

Details

This function assumes that the ‘analysis‘ slot in the ‘diff_results‘ object contains a ‘gene_id‘ column. If this column is not present, or cannot be processed, the function will return NULL.

Value

A list containing:

`enrich_go_object` ‘enrichResult‘ object from ‘clusterProfiler‘.

`results_table` Data frame of enrichment results.

`dot_plot` ‘ggplot‘ object of the dot plot.

NULL if no significant enrichment is found or if input validation fails.

Examples

```
# This example requires the 'org.Dm.eg.db' package
if (requireNamespace("org.Dm.eg.db", quietly = TRUE)) {
    # Helper function to create a sample DamIDResults object
    .generate_example_results <- function() {
        # Define a mock gene object. Note: Real, mappable FlyBase IDs are
        # used for the 'gene_id' column to ensure the example runs.
        mock_genes_gr <- GenomicRanges::GRanges(
            seqnames = S4Vectors::Rle("2L", 7),
            ranges = IRanges::IRanges(
                start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
                end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
            ),
            gene_id = c(
                "FBgn0034439", "FBgn0031267", "FBgn0051138", "FBgn0031265",
                "FBgn0004655", "FBgn0000251", "FBgn0000252"
            ),
            gene_name = c("ap", "dpr1", "side", "dpr2", "eg", "bi", "br")
        )
        data_dir <- system.file("extdata", package = "damidBind")
        loaded_data <- load_data_peaks(
            binding_profiles_path = data_dir,
            peaks_path = data_dir,
            ensdb_genes = mock_genes_gr,
            quantile_norm = TRUE
        )
        diff_results <- differential_binding(
            loaded_data,
            cond = c("L4 Neurons" = "L4",
                     "L5 Neurons" = "L5")
        )
        return(diff_results)
    }
    diff_results <- .generate_example_results()
```

```
    # Run GO Enrichment for genes enriched in the first condition ("L4")
    # Note: with tiny sample data, this may not find significant terms.
    go_results <- analyse_go_enrichment(
        diff_results,
        direction = "L4",
        org_db = org.Dm.eg.db::org.Dm.eg.db
    )

    # Print the results table if any enrichment was found
    if (!is.null(go_results)) {
        print(go_results$results_table)
    }
}
```

`analysisTable` *Access the differential binding analysis results*

Description

This function returns the full differential analysis table from the `DamIDResults` object.

Usage

```
analysisTable(object)
```

Arguments

`object`	A `DamIDResults` object.

Value

A `data.frame` with the full analysis results.

See Also

`DamIDResults-class` for an overview of the class and all its methods.

Examples

```
# Helper function to create a sample DamIDResults object for examples
.generate_example_results <- function() {
    analysis_df <- data.frame(
        logFC = c(2, -2, 0.1), P.Value = c(0.01, 0.01, 0.9), B = c(4, 3, -1),
        gene_name = c("GeneA", "GeneB", "GeneC"),
        row.names = c("chr1:1-100", "chr1:101-200", "chr1:201-300")
    )
    new("DamIDResults",
        analysis = analysis_df,
        upCond1 = analysis_df[1, , drop = FALSE],
```

```
        upCond2 = analysis_df[2, , drop = FALSE],
        cond = c("Condition 1" = "C1", "Condition 2" = "C2"),
        data = list(test_category = "bound")
    )
}
mock_results <- .generate_example_results()
analysisTable(mock_results)
```

`browse_igv_regions` *Interactive IGV visualisation (Shiny + igvShiny) of differential regions*

Description

Launches a Shiny app with an embedded IGV browser and an interactive table listing differentially-bound regions (from ‘differential_binding()‘ or ‘differential_accessibility()‘ results). Clicking on a region in the table will pan IGV to that locus. Sample coverage and region tracks are loaded as quantitative/annotation tracks. A dedicated "Save as SVG" button is provided to export the current IGV view.

Usage

```
browse_igv_regions(
  diff_results,
  samples = NULL,
  use_unique_ids = TRUE,
  colour_cond1 = "#ff6600",
  colour_cond2 = "#2288dd",
  use_genome = NULL,
  padding_width = 20000,
  trackHeight = 65,
  peakColour = "darkgreen",
  trackColour = "#6666ff",
  host = "localhost",
  port = NULL
)
```

Arguments

`diff_results`	A ‘DamIDResults‘ object, as returned by ‘differential_binding()‘ or ‘differential_accessibility()‘.
`samples`	Optional character vector of sample names to display (default: all in dataset).
`use_unique_ids`	Logical. When ‘TRUE‘ (default), simplified unique sample names will be displayed. Set as ‘FALSE‘ to use the full sample file names from loading.
`colour_cond1, colour_cond2`	Colours for differentially-enriched region tracks.
`use_genome`	IGV genome name (inferred from peak annotations if not provided).

padding_width	Width to pad browser viewbox on either side of the peak (Default: 20000)
trackHeight	Height of bedGraph tracks (Default: 65)
peakColour	Colour for significant peaks track (Default: "darkgreen")
trackColour	Colour for bedGraph tracks (Default: "#6666ff")
host	Hostname for the server location (Default: "localhost").
port	Port for connection (if NULL (default) the port is assigned by Shiny).

Value

Invisibly returns the Shiny app object created by ‘shinyApp()‘.

Examples

```
if (isTRUE(curl::has_internet()) && interactive()) {
# This example launches an interactive Shiny app in a web browser

.generate_example_results <- function() {
    mock_genes_gr <- GenomicRanges::GRanges(
        seqnames = S4Vectors::Rle("2L", 7),
        ranges = IRanges::IRanges(
            start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
            end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
        ),
      gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
       gene_name = c("geneA", "geneB", "geneC", "geneD", "geneE", "geneF", "LargeTestGene")
    )
    data_dir <- system.file("extdata", package = "damidBind")
    loaded_data <- load_data_peaks(
        binding_profiles_path = data_dir,
        peaks_path = data_dir,
        ensdb_genes = mock_genes_gr,
        quantile_norm = TRUE
    )
    diff_results <- differential_binding(
        loaded_data,
        cond = c("L4 Neurons" = "L4",
                 "L5 Neurons" = "L5")
    )
    return(diff_results)
}
diff_results <- .generate_example_results()

# Launch the interactive browser
browse_igv_regions(diff_results)
}
```

`calculate_and_add_occupancy_pvals` *Calculate and add gene occupancy FDR*

Description

This function calculates a p-value for gene occupancy scores. When adjusted for multiple-hypothesis testing, the resulting FDR may used as a proxy for determining whether a gene is actively expressed in RNA Polymerase TaDa experiments. The function applies a permutation-based null model to each sample's binding profile to determine empirical p-values, and returns these as new columns in the input occupancy dataframe.

These p-values are then aggregated and

Usage

```
calculate_and_add_occupancy_pvals(
  binding_data,
  occupancy_df,
  null_model_iterations = 1e+05,
  return_per_replicate_fdr = FALSE,
  plot_diagnostics = FALSE,
  BPPARAM = BiocParallel::bpparam(),
  seed = NULL
)
```

Arguments

`binding_data`	A 'GRanges' object of binding profiles, where metadata columns represent samples. This is typically the 'binding_profiles_data' element from the list returned by 'load_data_genes'.
`occupancy_df`	A data frame of gene occupancies, typically the 'occupancy' element from the list returned by 'load_data_genes'. It must contain sample columns and a 'nfrags' column.
`null_model_iterations`	Integer. The number of sampling iterations to build the FDR null model. Higher values are more accurate but slower. Default is 100000.
`return_per_replicate_fdr`	Logical. Returns individual FDR scores by applying BH adjustment on each individual sample. Use this option if not intending to apply downstream condition-level p-value aggregation via 'differential_binding()'. (Default: FALSE)
`plot_diagnostics`	Logical. If TRUE, will plot the Tier 2 regression fits for the p-value prediction slope, intercept and mean squared error (MSE). (Default: FALSE)
`BPPARAM`	A 'BiocParallelParam' instance specifying the parallel backend to use for computation. See 'BiocParallel::bpparam()'.

seed — An optional integer. If provided, it is used to set the random seed before the sampling process begins, ensuring that the FDR calculations are fully reproducible. If 'NULL' (default), results may vary between runs.

Details

The algorithm used here is substantially revised and adapted from the method described in Southall et al. (2013), Dev Cell, 26(1), 101-12. The broad principle is as follows:

1. For each sample, a null distribution of mean occupancy scores is simulated by repeatedly sampling random fragments from the genome-wide binding profile. This is done for various numbers of fragments.
2. A two-tiered regression model is fitted to the simulation results. This creates a statistical model that can predict the empirical p-value for any given occupancy score and fragment count.
3. This predictive model is then applied to the actual observed mean occupancy and fragment count for each gene in the 'occupancy_df'.
4. The final set of p-values is adjusted using BH correction to generate FDR scores for each gene.
5. The FDR value for each gene in each sample is appended to the 'occupancy_df' in a new column.

Key differences from the original Southall et al. algorithm include fitting natural spline models, using WLS to account for heterscedasticity of models, and sampling with replacement to generate the underlying null distribution.

This function is typically not called directly by the user, but is instead invoked via 'load_data_genes(calculate_occupancy_pva
= TRUE)'.

Value

An updated version of the input 'occupancy_df' data frame, with new columns appended. For each sample column (e.g., 'SampleA'), a corresponding p-value column (e.g., 'SampleA_pval') is added.

See Also

[load_data_genes()] which uses this function internally.

Examples

```
# Prepare sample binding data (GRanges object)
# Here, we'll load one of the sample files included with the package
# (this is a TF binding profile, which would not normally be used for
# occupancy calculations)
data_dir <- system.file("extdata", package = "damidBind")
bgraph_file <- file.path(data_dir, "Bsh_Dam_L4_r1-ext300-vs-Dam.kde-norm.gatc.2L.bedgraph.gz")

# The internal function `import_bedgraph_as_df` is used here for convenience.
# The column name 'score' will be replaced with the sample name.
binding_df <- damidBind:::import_bedgraph_as_df(bgraph_file, colname = "Sample_1")

binding_gr <- GenomicRanges::GRanges(
    seqnames = binding_df$chr,
```

```
    ranges = IRanges::IRanges(start = binding_df$start, end = binding_df$end),
    Sample_1 = binding_df$Sample_1
)

# Create a mock occupancy data frame.
# In a real analysis, this would be generated by `calculate_occupancy()`.
mock_occupancy <- data.frame(
    name = c("geneA", "geneB", "geneC"),
    nfrags = c(5, 10, 2),
    Sample_1 = c(1.5, 0.8, -0.2),
    row.names = c("geneA", "geneB", "geneC")
)

# Calculate pvals.
# We use a low null_model_iterations for speed, and a seed for reproducibility.
occupancy_with_pvals <- calculate_and_add_occupancy_pvals(
    binding_data = binding_gr,
    occupancy_df = mock_occupancy,
    null_model_iterations = 100,
    BPPARAM = BiocParallel::SerialParam(),
    seed = 42
)

# View the result, which now includes a _pval column for Sample_1.
print(occupancy_with_pvals)
```

`calculate_occupancy` *Compute occupancy for genomic regions*

Description

For each interval in the 'regions' GRanges object, this function finds all overlapping fragments in 'binding_data' and computes a weighted mean of their signal values. Any metadata columns present in the input 'regions' object are preserved in the output data.frame.

Usage

```
calculate_occupancy(
  binding_data,
  regions,
  buffer = 0,
  BPPARAM = BiocParallel::bpparam()
)
```

Arguments

`binding_data`	A data.frame as produced by 'build_dataframes()'. It must contain columns 'chr', 'start', 'end', followed by numeric sample columns.

regions	A GRanges object of genomic intervals (e.g., genes or reduced peaks) over which to calculate occupancy.
buffer	Optional integer. Number of base pairs to expand each interval in ‘regions‘ on both sides before calculating occupancy. Default is 0.
BPPARAM	A BiocParallel parameter object for parallel computation. Default is ‘BiocParallel::bpparam()‘.

Value

A data.frame with one row per region from the input ‘regions‘ object. The output includes the weighted mean occupancy for each sample, ‘nfrags‘ (number of overlapping fragments), and all original metadata columns from ‘regions‘. Rownames are generated from region coordinates to ensure uniqueness.

Examples

```
# Create a set of regions with metadata
regions_gr <- GenomicRanges::GRanges(
    "chrX", IRanges::IRanges(start = c(100, 500), width = 100),
    gene_name = c("MyGene1", "MyGene2"), score = c(10, 20)
)

# Create a mock binding data GRanges object
binding_gr <- GenomicRanges::GRanges(
    seqnames = "chrX",
    ranges = IRanges::IRanges(
        start = c(90, 150, 480, 550),
        end = c(110, 170, 520, 580)
    ),
    sampleA = c(1.2, 0.8, 2.5, 3.0),
    sampleB = c(1.0, 0.9, 2.8, 2.9)
)

# Calculate occupancy over the regions
# Use BiocParallel::SerialParam() for deterministic execution in examples
if (requireNamespace("BiocParallel", quietly = TRUE)) {
    occupancy_data <- calculate_occupancy(binding_gr, regions_gr,
        BPPARAM = BiocParallel::SerialParam()
    )
    print(occupancy_data)
}
```

---

conditionNames *Access condition name mapping*

Description

This function returns the mapping of user-friendly display names to internal condition identifiers.

Usage

```
conditionNames(object)
```

Arguments

object	A DamIDResults object.

Value

A named character vector.

See Also

DamIDResults-class

Examples

```
# Helper function to create a sample DamIDResults object for examples
.generate_example_results <- function() {
    analysis_df <- data.frame(
        logFC = c(2, -2, 0.1), P.Value = c(0.01, 0.01, 0.9), B = c(4, 3, -1),
        gene_name = c("GeneA", "GeneB", "GeneC"),
        row.names = c("chr1:1-100", "chr1:101-200", "chr1:201-300")
    )
    new("DamIDResults",
        analysis = analysis_df,
        upCond1 = analysis_df[1, , drop = FALSE],
        upCond2 = analysis_df[2, , drop = FALSE],
        cond = c("Condition 1" = "C1", "Condition 2" = "C2"),
        data = list(test_category = "bound")
    )
}
mock_results <- .generate_example_results()
conditionNames(mock_results)
```

DamIDResults-class *The DamIDResults Class*

Description

An S4 class to store the results of a differential analysis, as generated by differential_binding or differential_accessibility. It contains the full analysis table, subsets of significantly changed regions, and associated metadata.

Slots

`analysis` A 'data.frame' containing the full differential analysis table from 'limma' or 'NOISeq'.

`upCond1` A 'data.frame' of regions significantly enriched in the first condition.

`upCond2` A 'data.frame' of regions significantly enriched in the second condition.

`cond` A named 'character' vector mapping user-friendly display names (the names) to the internal condition identifiers (the values) used in the analysis.

`data` A 'list' containing the initial input data used for the analysis, including the occupancy 'data.frame' and other metadata.

Accessor Methods

The following accessor functions are available for a `DamIDResults` object.

- `analysisTable(object)`: Returns the full differential analysis table (a `data.frame`).
- `enrichedCond1(object)`: Returns a `data.frame` of regions significantly enriched in the first condition.
- `enrichedCond2(object)`: Returns a `data.frame` of regions significantly enriched in the second condition.
- `conditionNames(object)`: Returns a named `character` vector mapping display names to internal identifiers.
- `inputData(object)`: Returns a `list` containing the original input data used for the analysis.
- `expressed(object, condition, fdr = 0.05, which = "any")`: Returns a `data.frame` of genes considered expressed in 'condition', based on an FDR threshold of significantly enriched occupancy. Only available for analyses with FDR calculations, generated via `load_data_genes(calculate_oc = TRUE)`.

Generic Methods

The generic `plot()` function is also S4-enabled for this class. Calling `plot(object)` is equivalent to calling `plot_volcano(diff_results = object)`.

See Also

For more powerful and specific plotting functions, see `plot_volcano`, `plot_venn`, and `analyse_go_enrichment`.

To explore the differential analysis results in an interactive IGV browser window, see `browse_igv_regions`

Examples

```
# Helper function to create a sample DamIDResults object for examples
.generate_example_results <- function() {
    analysis_df <- data.frame(
        logFC = c(2, -2, 0.1), P.Value = c(0.01, 0.01, 0.9), B = c(4, 3, -1),
        gene_name = c("GeneA", "GeneB", "GeneC"),
        row.names = c("chr1:1-100", "chr1:101-200", "chr1:201-300")
    )
    new("DamIDResults",
        analysis = analysis_df,
```

```
        upCond1 = analysis_df[1, , drop = FALSE],
        upCond2 = analysis_df[2, , drop = FALSE],
        cond = c("Condition 1" = "C1", "Condition 2" = "C2"),
        data = list(test_category = "bound")
    )
}
mock_results <- .generate_example_results()

# Show the object summary
mock_results

# Access different parts of the object
analysisTable(mock_results)
enrichedCond1(mock_results)
conditionNames(mock_results)
```

`differential_accessibility` *Differential accessibility analysis for CATaDa (‘NOISeq‘ based)*

Description

Setup and differential analysis for CATaDa chromatin accessibility experiments using ‘NOISeq‘. Accepts output from ‘load_data_peaks‘, prepares a count matrix, performs ‘NOISeq‘ analysis, and returns differentially-accessible loci.

Usage

```
differential_accessibility(data_list, cond, regex = FALSE, norm = "n", q = 0.8)
```

Arguments

`data_list`	List. Output from load_data_peaks.
`cond`	A named or unnamed character vector of length two. The values are strings or regular expressions used to identify samples for each condition. If the vector is named, the names are used as user-friendly display names for the conditions in plots and outputs. If unnamed, the match strings are used as display names. The order determines the contrast, e.g., ‘cond[1]‘ vs ‘cond[2]‘.
`regex`	Logical. If ‘TRUE‘, the strings in ‘cond‘ are treated as regular expressions for matching sample names. If ‘FALSE‘ (the default), fixed string matching is used.
`norm`	Normalisation method passed to NOISeq. Defaults to "n" (no normalisation), but "uqua" (upper quantile) or "tmm" (trimmed mean of M) are options if needed
`q`	Numeric. Q-value threshold for NOISeq significance (default 0.8).

Value

A 'DamIDResults' object containing the results. Access slots using accessors (e.g., 'analysisTable(results)'). The object includes:

upCond1	data.frame of regions enriched in condition 1
upCond2	data.frame of regions enriched in condition 2
analysis	data.frame of full results for all tested regions
cond	A named character vector mapping display names to internal condition names
data	The original 'data_list' input

Examples

```
# NOTE: This example uses mock counts data, as the package's sample
# data is in log2-ratio format.

# Create a mock data_list with plausible count data
mock_occupancy_counts <- data.frame(
    name = c("peak1", "peak2", "peak3"),
    gene_name = c("GeneA", "GeneB", "GeneC"),
    gene_id = c("ID_A", "ID_B", "ID_C"),
    GroupA_rep1 = c(100, 20, 50), GroupA_rep2 = c(110, 25, 45),
    GroupB_rep1 = c(10, 200, 55), GroupB_rep2 = c(15, 220, 60),
    row.names = c("peak1", "peak2", "peak3")
)

mock_data_list <- list(
    occupancy = mock_occupancy_counts,
    test_category = "accessible"
)

# Run differential accessibility analysis
diff_access_results <- differential_accessibility(
    mock_data_list,
    cond = c("Group A" = "GroupA", "Group B" = "GroupB")
)

# View the results summary
diff_access_results
```

---

differential_binding *Differential binding/expression analysis ('limma')*

Description

Setup and differential analysis for occupancy/binding experiments using 'limma'. Accepts output from 'load_data_peaks' or 'load_data_genes', prepares an experiment matrix, fits linear models, and returns DE loci.

Usage

```
differential_binding(
  data_list,
  cond,
  regex = FALSE,
  fdr = 0.05,
  eBayes_trend = TRUE,
  eBayes_robust = TRUE,
  plot_diagnostics = interactive(),
  filter_occupancy = TRUE,
  filter_threshold = 0,
  filter_positive_enrichment = TRUE,
  p_combine_method = c("stouffer", "fisher")
)
```

Arguments

data_list	List. Output from load_data_peaks or load_data_genes.
cond	A named or unnamed character vector of length two. The values are strings or regular expressions used to identify samples for each condition. If the vector is named, the names are used as user-friendly display names for the conditions in plots and outputs. If unnamed, the match strings are used as display names. The order determines the contrast, e.g., 'cond[1]' vs 'cond[2]'.
regex	Logical. If 'TRUE', the strings in 'cond' are treated as regular expressions for matching sample names. If 'FALSE' (the default), fixed string matching is used.
fdr	Numeric. FDR threshold for significance (default 0.05).
eBayes_trend	Logical. If 'TRUE', the analysis will account for data heteroscedasticity, which is common in DamID-seq data. (default: TRUE)
eBayes_robust	Logical. If 'TRUE', the fitted trend should be robust to outliers. Only useful when 'eBayes_trend = TRUE'. Recommended for DamID-seq data. (default: TRUE)
plot_diagnostics	Logical. If 'TRUE' (default for interactive sessions), plots limma diagnostics to assess eBayes moderation, using 'plot_limma_diagnostics'.
filter_occupancy	NULL or integer. Filters out any locus with occupancy > 'filter_threshold' in fewer than this number of samples of any single condition when set. If set to TRUE, defaults to the minimum length of the two conditions. If FALSE or NULL, no filtering is applied. (default: TRUE)
filter_threshold	Numeric. 'filter_occupancy' uses this value for thresholding the input data. (default: 0)
filter_positive_enrichment	Logical. If 'TRUE' (default), regions are only considered significantly enriched if the mean score in the enriched condition is greater than zero. For example, for a region to be in 'upCond1', its logFC must be positive and its mean score

in condition 1 must be > 0. Set to 'FALSE' to include all statistically significant changes.

`p_combine_method`
: Method to combine p-values from replicates (default: "fisher")

Value

A 'DamIDResults' object containing the results. Access slots using accessors:

`enrichedCond1()`
: data.frame of regions enriched in condition 1

`enrichedCond2()`
: data.frame of regions enriched in condition 2

`analysisTable()`
: data.frame of full results for all tested regions

`conditionNames()`
: A named character vector mapping display names to internal condition names

`inputData()`
: The original 'data_list' input

Examples

```
# Create a mock GRanges object for gene annotations
# This object, based on the package's unit tests, avoids network access.
mock_genes_gr <- GenomicRanges::GRanges(
    seqnames = S4Vectors::Rle("2L", 7),
    ranges = IRanges::IRanges(
        start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
        end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
    ),
    strand = S4Vectors::Rle(GenomicRanges::strand(c("+", "-", "+", "+", "-", "-", "+"))),
   gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
    gene_name = c("geneA", "geneB", "geneC", "geneD", "geneE", "geneF", "LargeTestGene")
)

# Get path to sample data files included with the package
data_dir <- system.file("extdata", package = "damidBind")

# Load data
loaded_data <- load_data_peaks(
    binding_profiles_path = data_dir,
    peaks_path = data_dir,
    ensdb_genes = mock_genes_gr,
    quantile_norm = TRUE
)

# Run differential binding analysis
diff_results <- differential_binding(
    loaded_data,
    cond = c("L4 Neurons" = "L4", "L5 Neurons" = "L5")
)
```

```
# View the results summary
diff_results
```

`enrichedCond1` *Access Condition 1 enriched regions*

Description

This function returns the subset of regions significantly enriched in the first condition.

Usage

```
enrichedCond1(object)
```

Arguments

`object`	A `DamIDResults` object.

Value

A `data.frame`.

See Also

`DamIDResults-class`

Examples

```
# Helper function to create a sample DamIDResults object for examples
.generate_example_results <- function() {
    analysis_df <- data.frame(
        logFC = c(2, -2, 0.1), P.Value = c(0.01, 0.01, 0.9), B = c(4, 3, -1),
        gene_name = c("GeneA", "GeneB", "GeneC"),
        row.names = c("chr1:1-100", "chr1:101-200", "chr1:201-300")
    )
    new("DamIDResults",
        analysis = analysis_df,
        upCond1 = analysis_df[1, , drop = FALSE],
        upCond2 = analysis_df[2, , drop = FALSE],
        cond = c("Condition 1" = "C1", "Condition 2" = "C2"),
        data = list(test_category = "bound")
    )
}
mock_results <- .generate_example_results()
enrichedCond1(mock_results)
```

enrichedCond2 *Access Condition 2 enriched regions*

Description

This function returns the subset of regions significantly enriched in the second condition.

Usage

```
enrichedCond2(object)
```

Arguments

object	A DamIDResults object.

Value

A data.frame.

See Also

DamIDResults-class

Examples

```
# Helper function to create a sample DamIDResults object for examples
.generate_example_results <- function() {
    analysis_df <- data.frame(
        logFC = c(2, -2, 0.1), P.Value = c(0.01, 0.01, 0.9), B = c(4, 3, -1),
        gene_name = c("GeneA", "GeneB", "GeneC"),
        row.names = c("chr1:1-100", "chr1:101-200", "chr1:201-300")
    )
    new("DamIDResults",
        analysis = analysis_df,
        upCond1 = analysis_df[1, , drop = FALSE],
        upCond2 = analysis_df[2, , drop = FALSE],
        cond = c("Condition 1" = "C1", "Condition 2" = "C2"),
        data = list(test_category = "bound")
    )
}
mock_results <- .generate_example_results()
enrichedCond2(mock_results)
```

expressed *Get expressed genes/loci by FDR*

Description

A method to filter genes or loci that are considered 'expressed' in a specific condition, based on a False Discovery Rate (FDR) threshold. The method is a wrapper around the `filter_genes_by_fdr` function.

Usage

```
expressed(object, condition, fdr = 0.05, which = "any")
```

Arguments

`object`	A `DamIDResults` object. This object must have been generated from data loaded with `load_data_genes(calculate_occupancy_pvals = TRUE)` for the underlying FDR columns to be present.
`condition`	A character string identifying the experimental condition to filter. This can be the internal identifier or the user-friendly display name.
`fdr`	Numeric. The FDR cutoff. Defaults to 0.05.
`which`	Character string, either 'any' or 'all'. Determines whether a gene must pass the FDR threshold in any or all replicates of the condition. Defaults to 'any'.

Value

A `data.frame` containing the `gene_name` and `gene_id` of genes that pass the filter.

See Also

`filter_genes_by_fdr`, `DamIDResults-class`, `load_data_genes`

Examples

```
.generate_fdr_example_results <- function() {
    occupancy_df <- data.frame(
        gene_name = c("geneA", "geneB", "geneC"),
        gene_id = c("FBgn01", "FBgn02", "FBgn03"),
        L4_rep1 = c(1.5, 0.2, 0.8),
        L4_rep2 = c(1.7, 0.9, 0.1),
        L5_rep1 = c(0.1, 0.1, 2.0),
        L4_rep1_FDR = c(0.01, 0.10, 0.04),
        L4_rep2_FDR = c(0.03, 0.02, 0.50),
        L5_rep1_FDR = c(0.80, 0.90, 0.01),
        row.names = c("geneA", "geneB", "geneC")
    )
    diff_results_base <- list(
        occupancy = occupancy_df,
```

```
        test_category = "expressed",
        matched_samples = list("L4" = c("L4_rep1", "L4_rep2"), "L5" = "L5_rep1")
    )
    new("DamIDResults",
        analysis = data.frame(row.names = rownames(occupancy_df)),
        upCond1 = data.frame(),
        upCond2 = data.frame(),
        cond = c("L4 mock" = "L4", "L5 mock" = "L5"),
        data = diff_results_base
    )
}
mock_fdr_results <- .generate_fdr_example_results()

# Get expressed in a condition (FDR <= 0.05)
expressed(mock_fdr_results, condition = "L4 mock")

# Get genes expressed with a more stringent FDR (<= 0.01)
expressed(mock_fdr_results, condition = "L4", fdr = 0.01)
```

`extract_unique_sample_ids` *Extract unique sample names from complex labels*

Description

This function takes a vector of complex sample labels and iteratively constructs a simplified, unique name for each. It identifies all blocks of text that differ across the sample set and progressively adds them to a base name until the combination of the base name and a replicate identifier is unique for every sample.

Usage

```
extract_unique_sample_ids(
  sample_names,
  delimiter = "[-_\\.]",
  replicate_pattern = "^(n|N|r|rep|replicate|sample)\\d+"
)
```

Arguments

`sample_names`	A character vector of sample labels.
`delimiter`	A regular expression used as a delimiter to split labels into blocks. (Default: '[-_\.]')
`replicate_pattern`	A regular expression used to identify the replicate block. (Default: '^(n|N|r|rep|replicate|sample)\d+')

Value

A vector of simplified, unique names. If a unique name cannot be formed or essential information is missing for a sample, the original label for that sample is returned as a fallback.

Examples

```
labels <- c(
  "RNAPII_elav-GSE77860-n1-SRR3164378-2017-vs-Dam.scaled.kde-norm",
  "RNAPII_elav-GSE77860-n2-SRR3164379-2017-vs-Dam.scaled.kde-norm",
  "RNAPII_elav-GSE77860-n4-SRR3164380-2017-vs-Dam.scaled.kde-norm",
  "RNAPII_Wor-GSE77860-n1-SRR3164346-2017-vs-Dam.scaled.kde-norm",
  "RNAPII_Wor-GSE77860-n2-SRR3164347-2017-vs-Dam.scaled.kde-norm",
  "RNAPII_Wor-GSE77860-sample1-SRR2038537-2017-vs-Dam.scaled.kde-norm"
)
extract_unique_sample_ids(labels)
```

---

filter_genes_by_fdr *Filter genes by FDR within a specific condition*

Description

Filters a list of genes to retain only those that meet a specified False Discovery Rate (FDR) threshold. If the input is a 'DamIDResults' object and a combined condition-level FDR has been calculated, that value is used. Otherwise, the function falls back to filtering against individual replicates.

Usage

```
filter_genes_by_fdr(data, fdr = 0.05, condition, which = "any")
```

Arguments

data	A 'DamIDResults' object or the 'list' returned by 'load_data_genes()'.
fdr	A numeric value between 0 and 1 specifying the FDR cutoff. (Default: 0.05)
condition	A character string identifying the experimental condition. This string should uniquely match the relevant sample columns (e.g., "L4" will match "L4_rep1_FDR" and "L4_rep2_FDR"). If 'data' is a 'DamIDResults' object, this can be either the internal identifier or the display name for the condition.
which	A character string, either '"any"' or '"all"'. Only applicable when falling back to individual replicate scores. (Default: '"any"')

- If '"any"', a gene is kept if it meets the 'fdr' threshold in at least one replicate of the specified 'condition'.
- If '"all"', a gene is kept only if it meets the 'fdr' threshold in all replicates of the specified 'condition'.

Details

This function is primarily used in workflows involving RNA Polymerase TaDa data, where an FDR is calculated for gene occupancy to determine if a gene is actively transcribed. It allows users to identify genes in a single condition that can be considered to be expressed (i.e. RNA Pol occupancy is significantly greater than background).

Note that while this is an effective proxy for gene expression, there are edge cases (e.g. paused polymerase, short genes directly adjacent to an expressed gene TSS or TES) where a gene may have significant occupancy but not, in fact, be transcribed.

The function locates the relevant FDR columns in the ‘occupancy‘ table by searching for column names that end with ‘_FDR‘ and also contain the ‘condition‘ string.

Value

A ‘data.frame‘ containing the ‘gene_name‘, ‘gene_id‘, ‘avg_occ‘, and the most significant FDR value found.

Examples

```
# Create a mock data object with an occupancy table containing FDR values,
# similar to the output of `load_data_genes(calculate_occupancy_pvals = TRUE)`.
.generate_fdr_example_results <- function() {
    occupancy_df <- data.frame(
        gene_name = c("geneA", "geneB", "geneC"),
        gene_id = c("FBgn01", "FBgn02", "FBgn03"),
        L4_rep1 = c(1.5, 0.2, 0.8),
        L4_rep2 = c(1.7, 0.9, 0.1),
        L5_rep1 = c(0.1, 0.1, 2.0),
        L4_rep1_FDR = c(0.01, 0.10, 0.04),
        L4_rep2_FDR = c(0.03, 0.02, 0.50),
        L5_rep1_FDR = c(0.80, 0.90, 0.01),
        row.names = c("geneA", "geneB", "geneC")
    )
    diff_results_base <- list(
        occupancy = occupancy_df,
        test_category = "expressed",
        matched_samples = list("L4" = c("L4_rep1", "L4_rep2"), "L5" = "L5_rep1")
    )
    new("DamIDResults",
        analysis = data.frame(row.names = rownames(occupancy_df)),
        upCond1 = data.frame(),
        upCond2 = data.frame(),
        cond = c("L4 mock" = "L4", "L5 mock" = "L5"),
        data = diff_results_base
    )
}
mock_data <- .generate_fdr_example_results()

# Example 1: Get genes with FDR <= 0.05 in ANY L4 replicate.
# geneA (0.01, 0.03), geneB (0.02), and geneC (0.04) pass.
expressed_in_L4_any <- filter_genes_by_fdr(
    mock_data,
    fdr = 0.05,
    condition = "L4",
    which = "any"
)
print(expressed_in_L4_any)
```

```
# Example 2: Get genes with FDR <= 0.05 in ALL L4 replicates.
# Only geneA (0.01, 0.03) passes.
expressed_in_L4_all <- filter_genes_by_fdr(
    mock_data,
    fdr = 0.05,
    condition = "L4",
    which = "all"
)
print(expressed_in_L4_all)

# Example 3: Get genes with FDR <= 0.05 in any L5 replicate.
# geneC (0.01) and geneD (0.02) pass.
expressed_in_L5 <- filter_genes_by_fdr(
    mock_data,
    fdr = 0.05,
    condition = "L5",
    which = "any"
)
print(expressed_in_L5)
```

`get_ensdb_genes` *Extract gene annotation from Ensembl via AnnotationHub EnsDb*

Description

Retrieves gene information for a given organism from the most appropriate Ensembl database hosted via Bioconductor's AnnotationHub and ensembldb.

Usage

```
get_ensdb_genes(
  organism_keyword = "drosophila melanogaster",
  genome_build = NULL,
  ensembl_version = NULL,
  exclude_biotypes = c("transposable_element", "pseudogene"),
  include_gene_metadata = c("gene_id", "gene_name")
)
```

Arguments

`organism_keyword`
: Character. Unique non-case-senstive string to search for the organism (e.g., "drosophila melanogaster").

`genome_build`
: Optional character. Genome build identifier to further restrict the EnsDb selection (e.g., "BDGP6").

`ensembl_version`
: Optional integer. Specific Ensembl version to fetch. If NULL, the latest available version is used.

exclude_biotypes
: Character vector. Gene biotypes to exclude from the result (default: c("transposable_element", "pseudogene")).

include_gene_metadata
: Character vector. Metadata columns to keep for each gene (default: c("gene_id", "gene_name")).

Details

This function queries AnnotationHub for EnsDb objects matching a supplied organism keyword, with optional filtering by genome build and Ensembl version. Genes matching excluded biotypes are filtered out. Only user-selected metadata fields are retained in the genes output.

Value

List with:

genes
: A GRanges object of genes (metadata columns per argument).

ensembl_version
: Character. The Ensembl version string.

genome_build
: Character. Genome build identifier.

species
: Character. Latin binomial species name.

common_name
: Character. Species common name.

Examples

```
if (isTRUE(curl::has_internet())) {
# This example requires an internet connection and will download data.
dm_genes <- get_ensdb_genes(
    organism_keyword = "drosophila melanogaster",
    ensembl_version = 110
)

# View the fetched genes GRanges object
dm_genes$genes
}
```

---

inputData *Access original input data and metadata*

Description

This function returns the original list of input data used to generate the results.

Usage

```
inputData(object)
```

Arguments

object	A DamIDResults object.

Value

A list.

See Also

DamIDResults-class

Examples

```
# Helper function to create a sample DamIDResults object for examples
.generate_example_results <- function() {
    analysis_df <- data.frame(
        logFC = c(2, -2, 0.1), P.Value = c(0.01, 0.01, 0.9), B = c(4, 3, -1),
        gene_name = c("GeneA", "GeneB", "GeneC"),
        row.names = c("chr1:1-100", "chr1:101-200", "chr1:201-300")
    )
    new("DamIDResults",
        analysis = analysis_df,
        upCond1 = analysis_df[1, , drop = FALSE],
        upCond2 = analysis_df[2, , drop = FALSE],
        cond = c("Condition 1" = "C1", "Condition 2" = "C2"),
        data = list(test_category = "bound")
    )
}
mock_results <- .generate_example_results()
inputData(mock_results)
```

---

load_data_genes *Load genome-wide binding data for gene expression (RNA polymerase occupancy)*

Description

Reads RNA Polymerase DamID binding profiles either from bedGraph files or directly from a named list of GRanges objects. Calculates binding occupancy summarised over genes.

Usage

```
load_data_genes(
  binding_profiles_path = NULL,
  binding_profiles = NULL,
  drop_samples = NULL,
  quantile_norm = FALSE,
  organism = "drosophila melanogaster",
```

```
  calculate_occupancy_pvals = TRUE,
  return_per_replicate_fdr = FALSE,
  occupancy_plot_diagnostics = interactive(),
  null_model_iterations = 1e+05,
  ensdb_genes = NULL,
  BPPARAM = BiocParallel::bpparam(),
  plot_diagnostics = interactive()
)
```

Arguments

`binding_profiles_path`
: Character vector of directories or file globs containing log2 ratio binding tracks in bedGraph format. Wildcards ('*') supported.

`binding_profiles`
: Named list of GRanges objects representing binding profiles.

`drop_samples`
: A character vector of sample names or patterns to remove. Matching samples are removed from the analysis before normalisation and occupancy calculation. This can be useful for excluding samples that fail initial quality checks. Default: 'NULL' (no samples are dropped).

`quantile_norm`
: Logical (default: FALSE) quantile-normalise across all signal columns if TRUE.

`organism`
: Organism string (lower case) to obtain genome annotation from (if not providing a custom 'ensdb_genes' object) Defautls to "drosophila melanogaster".

`calculate_occupancy_pvals`
: Calculate occupancy p-values as a proxy for gene expression status (see details). Not used for differential expression analysis, but used when present for downstream analysis and plotting. (default: TRUE)

`return_per_replicate_fdr`
: Legacy option of returning BH-adjusted RNA Polymerase occupancy FDR values per replicate. As of v0.99.12, unadjusted p-values are returned by defualt; these are then aggregated at the condition level during 'differential_binding()' and the aggregate p-values adjusted to gain statistical power. This option exists for legacy or unsual end-user applications. Use with caution. (default: FALSE)

`occupancy_plot_diagnostics`
: Logical. If 'TRUE' (default in interactive sessions), diagnostic plots for the gene expression null model will be displayed.

`null_model_iterations`
: Number of iterations to use to determine null model for FDR (default: 100000)

`ensdb_genes`
: GRanges object: gene annotation. Automatically obtained from 'organism' if NULL.

`BPPARAM`
: BiocParallel function (defaults to BiocParallel::bpparam())

`plot_diagnostics`
: Logical. If 'TRUE' (the default in interactive sessions), diagnostic plots (PCA and correlation heatmap) will be generated and displayed for both the raw binding data and the summarised occupancy data.

Details

One of ‘binding_profiles_path‘ or ‘binding_profiles‘ must be provided.

When supplying GRanges lists, each GRanges should contain exactly one numeric metadata column representing the signal, and ‘binding_profiles‘ must be a named list, with element names used as sample names.

The algorithm for determining gene occupancy FDR (as a proxy for gene expression) is based on ‘polii.gene.call‘, which in turn was based on that described in Southall et al. (2013). Dev Cell, 26(1), 101–12. doi:10.1016/j.devcel.2013.05.020. Briefly, the algorithm establishes a null model by simulating the distribution of mean occupancy scores from random fragments. It fits a two-tiered regression to predict the False Discovery Rate (FDR), based on fragment count and score. For each gene, the true weighted mean occupancy and fragment count are calculated from the provided binding profile. Finally, the pre-computed regression models are used to assign a specific FDR to each gene based on its observed occupancy and fragment count.

Value

List with elements:

`binding_profiles_data`
: data.frame of merged binding profiles, with chr, start, end, sample columns, and _pval columns if ‘calculate_occupancy_pvals=TRUE‘

`occupancy`
: data.frame of occupancy values summarised over genes.

`test_category`
: Character scalar; will be "expressed".

Examples

```
# Create a mock GRanges object for gene annotations
# This object, based on the package's unit tests, avoids network access
# and includes a very long gene to ensure overlaps with sample data.
mock_genes_gr <- GenomicRanges::GRanges(
    seqnames = S4Vectors::Rle("2L", 7),
    ranges = IRanges::IRanges(
        start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
        end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
    ),
    strand = S4Vectors::Rle(GenomicRanges::strand(c("+", "-", "+", "+", "-", "-", "+"))),
   gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
    gene_name = c("geneA", "geneB", "geneC", "geneD", "geneE", "geneF", "LargeTestGene")
)

# Get path to sample data files included with the package
data_dir <- system.file("extdata", package = "damidBind")

# Run loading function using sample files and mock gene annotations
# This calculates occupancy over genes instead of peaks.
loaded_data_genes <- load_data_genes(
    binding_profiles_path = data_dir,
    ensdb_genes = mock_genes_gr,
    quantile_norm = FALSE,
    calculate_occupancy_pvals = FALSE
```

```
)

# View the head of the occupancy table
head(loaded_data_genes$occupancy)
```

`load_data_peaks` *Load genome-wide binding data and associated peak files or GRanges objects*

Description

Reads DamID-seq log2 ratio binding data either from bedGraph files or directly from a list of GRanges objects, and associated peak regions either from GFF/bed files or from a list of GRanges objects. This function is suitable for transcription factor binding analyses. For peak discovery, use an external peak caller (e.g. 'find_peaks').

Usage

```
load_data_peaks(
  binding_profiles_path = NULL,
  peaks_path = NULL,
  binding_profiles = NULL,
  peaks = NULL,
  drop_samples = NULL,
  maxgap_loci = 1000,
  quantile_norm = FALSE,
  organism = "drosophila melanogaster",
  ensdb_genes = NULL,
  BPPARAM = BiocParallel::bpparam(),
  plot_diagnostics = interactive()
)
```

Arguments

`binding_profiles_path`
: Character vector. Path(s) to directories or file globs containing log2 ratio binding tracks in bedGraph format. Wildcards ('*') supported.

`peaks_path`
: Character vector. Path(s) to directories or file globs containing the peak calls in GFF or BED format.

`binding_profiles`
: List of GRanges objects with binding profiles, one per sample.

`peaks`
: List of GRanges objects representing peak regions.

`drop_samples`
: A character vector of sample names or patterns to remove. Matching samples are removed from the analysis before normalisation and occupancy calculation. This can be useful for excluding samples that fail initial quality checks. Default: 'NULL' (no samples are dropped).

`maxgap_loci` Integer, the maximum bp distance between a peak boundary and a gene to associate that peak with the gene. Default: 1000.

`quantile_norm` Logical (default: FALSE). If TRUE, quantile-normalise the signal columns across all datasets.

`organism` Organism string (lower case) to obtain genome annotation from (if not providing a custom ‘ensdb_genes‘ object) Default: "drosophila melanogaster".

`ensdb_genes` GRanges object: gene annotation. Automatically obtained from ‘organism‘ if NULL.

`BPPARAM` BiocParallel function (defaults to BiocParallel::bpparam())

`plot_diagnostics`
Logical. If ‘TRUE‘ (the default in interactive sessions), diagnostic plots (PCA and correlation heatmap) will be generated and displayed for both the raw binding data and the summarised occupancy data.

Details

One of ‘binding_profiles_path‘ or ‘binding_profiles‘ must be provided. Similarly, one of ‘peaks_path‘ or ‘peaks‘ must be provided.

When supplying GRanges lists, each GRanges should contain exactly one numeric metadata column representing the binding signal, and all GRanges should be supplied as a named list, with element names used as sample names.

Value

A list with components:

`binding_profiles_data`
data.frame: Signal matrix for all regions, with columns chr, start, end, sample columns.

`peaks` list(GRanges): All loaded peak regions from input files or directly supplied.

`pr` GRanges: Reduced (union) peak regions across samples.

`occupancy` data.frame: Binding values summarised over reduced peaks, with overlap annotations.

`test_category` Character scalar; will be "bound".

Examples

```
# Create a mock GRanges object for gene annotation
# This object, based on the package's unit tests, avoids network access
# and includes a very long gene to ensure overlaps with sample data.
mock_genes_gr <- GenomicRanges::GRanges(
    seqnames = S4Vectors::Rle("2L", 7),
    ranges = IRanges::IRanges(
        start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
        end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
    ),
    strand = S4Vectors::Rle(GenomicRanges::strand(c("+", "-", "+", "+", "-", "-", "+"))),
```

```
   gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
    gene_name = c("geneA", "geneB", "geneC", "geneD", "geneE", "geneF", "LargeTestGene")
)

# Get path to sample data files included with the package
data_dir <- system.file("extdata", package = "damidBind")

# Run loading function using sample files and mock gene annotations
loaded_data <- load_data_peaks(
    binding_profiles_path = data_dir,
    peaks_path = data_dir,
    ensdb_genes = mock_genes_gr,
    quantile_norm = TRUE
)

# View the structure of the output
str(loaded_data, max.level = 1)
```

`plot_input_diagnostics` *Display diagnostic plots for input data*

Description

This function creates and displays diagnostic plots (PCA and correlation heatmap) for both occupancy and raw binding data. It is called by ‘load_data_peaks‘ and ‘load_data_genes‘.

Usage

```
plot_input_diagnostics(loaded_data, drop_samples = NULL)
```

Arguments

`loaded_data`	A list object, the output of ‘load_data_peaks‘ or ‘load_data_genes‘.
`drop_samples`	An optional character vector of sample names or patterns to remove for this diagnostic check. When used, the occupancy data is subsetted, not recalculated, providing an approximation of the effect of dropping samples. Default: ‘NULL‘.

Value

Returns the input ‘loaded_data‘ object invisibly

Examples

```
# Mock ensdb data to avoid network access
mock_genes_gr <- GenomicRanges::GRanges(
    seqnames = S4Vectors::Rle("2L", 7),
    ranges = IRanges::IRanges(
        start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
        end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
    ),
   gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
    gene_name = c("geneA", "geneB", "geneC", "geneD", "geneE", "geneF", "LargeTestGene")
)
data_dir <- system.file("extdata", package = "damidBind")

# Load the example package data
loaded_data <- load_data_peaks(
    binding_profiles_path = data_dir,
    peaks_path = data_dir,
    ensdb_genes = mock_genes_gr,
    plot_diagnostics = FALSE # don't call the function here ...
)

# Plot diagnostics
plot_input_diagnostics(loaded_data) # ... so that we can call it explicity :/
```

---

plot_limma_diagnostics

Verify Underlying Assumptions for 'limma' Analysis

Description

This diagnostic function is a wrapper around the internal '._plot_limma_diagnostics_internal()' funtion, to help assess whether the assumptions of the 'limma' empirical Bayes framework hold for a given dataset. It generates a series of plots to check for normality of residuals, homoscedasticity, and the mean-variance relationship, illustrating in particular the effect of 'trend' and 'robust' parameters to 'limma::eBayes'.

During 'limma'-based fits, the internal plot routine is called by default. This wrapper allows diagnostics to be displayed for any given log2 ratio-based 'data_list' object from 'load_data_peaks()' or 'load_data_genes()', and the effect of moderation parameters on the fit tested.

Usage

```
plot_limma_diagnostics(
  data_list,
  cond,
  drop_samples = NULL,
  filter_occupancy = TRUE,
  filter_threshold = 0,
```

```
  eBayes_trend = TRUE,
  eBayes_robust = TRUE,
  regex = FALSE
)
```

Arguments

data_list	List. The output from 'load_data_peaks' or 'load_data_genes'.
cond	A named character vector of length two defining the conditions for comparison, identical to the 'cond' argument in 'differential_binding'.
drop_samples	An optional character vector of sample names or patterns to remove for this diagnostic check. Default: 'NULL'.
filter_occupancy	NULL or integer. See 'prep_data_for_differential_analysis'. Default is 'TRUE'.
filter_threshold	Numeric. Threshold value for 'filter_occupancy'. (default: 0)
eBayes_trend	Logical. See 'limma::eBayes'. Default: 'TRUE'
eBayes_robust	Logical. See 'limma::eBayes'. Default: 'TRUE'
regex	Logical. Whether to use regular expressions for matching condition names. Default is 'FALSE'.

Details

The function first prepares the data and fits a linear model using the 'limma' package. It then calls an internal plotting routine to generate the following checks:

1. **Homoscedasticity (Residuals vs. Fitted):** A scatter plot of model residuals against fitted values. A random cloud around y=0 supports the assumption of constant variance.
2. **Effect of eBayes moderation:** Histograms of t-statistics before and after empirical Bayes moderation.
3. **Mean-variance trend ('plotSA'):** The primary diagnostic for the 'eBayes' step, showing the relationship between average log2 occupancy and variance. Points should be evenly distributed around the blue trendline; any outliers are highlighted in red.

The function uses the internal 'prep_data_for_differential_analysis' function to ensure that the data being tested is identical to that used in the main differential analysis.

Value

Invisibly returns 'NULL'. This function is called to generate diagnostic plots in the active graphics device.

Examples

```
mock_genes_gr <- GenomicRanges::GRanges(
    seqnames = S4Vectors::Rle("2L", 7),
    ranges = IRanges::IRanges(
        start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
```

```
        end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
    ),
   gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
    gene_name = c("geneA", "geneB", "geneC", "geneD", "geneE", "geneF", "LargeTestGene")
)

data_dir <- system.file("extdata", package = "damidBind")

loaded_data <- load_data_peaks(
    binding_profiles_path = data_dir,
    peaks_path = data_dir,
    ensdb_genes = mock_genes_gr,
    quantile_norm = TRUE,
    plot_diagnostics = FALSE
)

conditions <- c("L4 Neurons" = "L4", "L5 Neurons" = "L5")

plot_limma_diagnostics(
    data_list = loaded_data,
    cond = conditions
)
```

---

plot_venn *Draw proportional Venn diagrams for differential binding analysis*

Description

Generates a two-set proportional Venn diagram summarising the results of the differential binding analysis. The set union represents significant binding peaks that fail to show significant differences in occupancy; the exclusive regions of each set represent regions with enriched differential binding in that condition. Note that regions can be bound in both conditions, and still show differential occupancy. For gene expression analysis, the set of analysed genes can optionally be filtered by FDR such that the universe is restricted to only genes deemed expressed, as is typically expected for DEG representations.

Usage

```
plot_venn(
  diff_results,
  title = NULL,
  subtitle = "",
  set_labels = NULL,
  filename = NULL,
  font = "sans",
  format = c("pdf", "svg"),
  region_colours = c("#FFA500", "#2288DD", "#CCCCCC"),
  fdr_filter_threshold = NULL
)
```

Arguments

diff_results	A 'DamIDResults' object, as returned by 'differential_binding()' or 'differential_accessibility()'.
title	Plot title to use (default: generated from test condition context)
subtitle	Subtitle to use (default is empty).
set_labels	Character vector of length 2. Names for the two sets/circles (defaults to the analysis condition names).
filename	Character. Path at which to save the diagram, if not NULL.
font	Font name to use (default is "sans")
format	Character. Output plot format, "pdf" or "svg" (default "pdf").
region_colours	Character vector of length 2 or 3. Fill colours for each set region (default: c("#FFA500", "#2288DD", "#CCCCCC")).
fdr_filter_threshold	Numeric or NULL. If a value (e.g., 0.05) is provided, the universe of loci considered for the Venn diagram will be restricted to those that pass this FDR threshold in at least one sample. Used for illustrating DEGs with RNA Pol TaDa. If NULL (default), all tested loci are used.

Value

The function is called to generating a plot. It invisibly returns 'NULL'.

Examples

```
# Helper function to create a sample DamIDResults object
.generate_example_results <- function() {
    mock_genes_gr <- GenomicRanges::GRanges(
        seqnames = S4Vectors::Rle("2L", 7),
        ranges = IRanges::IRanges(
            start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
            end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
        ),
      gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
       gene_name = c("geneA", "geneB", "geneC", "geneD", "geneE", "geneF", "LargeTestGene")
    )
    data_dir <- system.file("extdata", package = "damidBind")
    loaded_data <- load_data_peaks(
        binding_profiles_path = data_dir,
        peaks_path = data_dir,
        ensdb_genes = mock_genes_gr,
        quantile_norm = TRUE
    )
    diff_results <- differential_binding(
        loaded_data,
        cond = c("L4 Neurons" = "L4",
                 "L5 Neurons" = "L5")
    )
    return(diff_results)
```

```
}
diff_results <- .generate_example_results()

# Generate the Venn diagram
plot_venn(diff_results)
```

`plot_volcano` *Volcano plot of differentially bound/expressed loci*

Description

Creates a volcano plot from the results of a differential analysis. The plot shows the log-fold change against a measure of statistical significance. The function offers extensive customisation for point appearance, gene labelling, and highlighting specific groups of loci.

Usage

```
plot_volcano(
  diff_results,
  fdr_filter_threshold = NULL,
  plot_config = list(),
  label_config = list(),
  highlight = NULL,
  highlight_config = list(),
  label_display = list(),
  save = NULL
)
```

Arguments

`diff_results` A 'DamIDResults' object, as returned by 'differential_binding()' or 'differential_accessibility()'.

`fdr_filter_threshold`
Numeric or NULL. If a value (e.g., 0.05) is provided, the volcano plot will only include loci that have an FDR value less than or equal to this threshold in at least one replicate of the two conditions being plotted. This requires that the data was loaded using 'load_data_genes' with 'calculate_occupancy_pvals = TRUE', which generates the necessary '_FDR' columns. If 'NULL' (default), no FDR-based filtering is performed.

`plot_config` List. Names to override plot details (title, axes, size, colours, etc); see details.

- `title`, `xlab`, `ylab` (character)
- `ystat` (character): The column name from 'analysisTable(diff_results)' to use for the y-axis (e.g., "minuslogp" or "B"). Default is "B".
- `base_size` (integer): ggplot theme base font size.
- `sig_colour`, `nonsig_colour` (colours)

- sig_alpha, sig_size: alpha and size for significant points.
- nonsig_alpha, nonsig_size: alpha and size for non-significant points.

label_config List. Fine-grained label controls; if missing or 'NULL', no labels are added (see details).

- genes: character vector to restrict labels to a subset (default: label all significant).
- label_size: label size (numeric).
- clean_names: logical; if 'TRUE', applies regex filtering to labels.
- names_clean, names_clean_extra: regex to exclude from labels when clean_names is 'TRUE'.
- max_overlaps: integer; maximum ggrepel overlaps. (default: 10)

highlight List. A simple list where each element is a character vector of genes/loci to highlight. Each element of this list will correspond to a separate highlight group. If 'NULL', no highlight overlays are drawn.

highlight_config

List. Additional highlight configuration options, applied consistently across all highlight groups. If missing or 'NULL', defaults are used.

- alpha: Numeric; transparency for highlight points (default: 1).
- size: Numeric; size for highlight points (default: 2).
- label: Logical; if 'TRUE', labels are added for all highlight groups (default: 'FALSE').
- colour: A list of colours, where each element corresponds to a highlight group in the 'highlight' list. If not specified or not enough colours are provided, a default hue palette is used.
- label_size: Numeric; label size (default: 4).
- max_overlaps: Integer; maximum ggrepel overlaps for highlight labels (default: 10).
- sig_labels_only: Logical; whether to only label significant loci in the set
- legend: Logical; whether to draw a plot legend for the highlight groups (default: TRUE).
- legend_inside: Logical; whether to draw the plot legend for the highlight groups inside the plot (default: TRUE).
- legend_inside_pos: String, either 'r' (right) or 'l' (left). Presets for internal legend position in the bottom right or left corners of the plot. (default: 'r')
- legend_position_override: Numeric. Manual override for internal legend positioning when not set to the default, NULL.
- legend_justification_override: Numeric. Manual override for internal legend justification when not set to the default, NULL.
- label_fill: logical; if 'TRUE', uses 'geom_label_repel', else 'geom_text_repel' (default: FALSE)
- text_col: logical; if 'TRUE', text is coloured as per points, else black (default: FALSE)

- text_luminosity: Numeric (0-100). When using 'text_col', setting a non-zero value will darken the luminosity of the highlight colour on text labels for increased contrast. 0 = no change; 100 = black. (default: 0)

label_display List. Additional label display options for sampling dense labels in all groups. Uses KNN-based sampling to optimise display when not NULL.

- scale: Logical; if TRUE, labelled coordinate data are centred and scaled (using scale(center = TRUE, scale = TRUE)) before sampling. Note: this does not affect plotted values. (default: TRUE).
- r: Numeric or "auto"; the sampling exclusion radius. If "auto", r is set to the median distance to the k_for_r-th nearest neighbour across all points. A smaller r keeps more points. (Default: 0.2).
- k_search: Integer; maximum number of neighbours to find in the initial KNN search. Must be greater than or equal to both k and k_for_r (default: 30).
- k_priority: Integer; number of neighbours used to infer the isolation priority score. Must be less than or equal to k_search (default: 30).
- k_for_r: Integer; which neighbour to use for the "auto" r calculation (default: 30).

save List or 'NULL'. Controls saving the plot to a file. If 'NULL', 'FALSE', or '0', the plot is not saved. If a 'list', it specifies saving parameters:

- filename (character): The path and base name for the output file (e.g., "my_volcano_plot"). If not specified, a default is used.
- format (character): File format ("pdf", "svg", or "png"). Default is "pdf".
- width (numeric): Width of the plot in inches. Default is 5.
- height (numeric): Height of the plot in inches. Default is 4.

Value

A 'ggplot' object

Examples

```
# Helper function to create a sample DamIDResults object
.generate_example_results <- function() {
    mock_genes_gr <- GenomicRanges::GRanges(
        seqnames = S4Vectors::Rle("2L", 7),
        ranges = IRanges::IRanges(
            start = c(1000, 2000, 3000, 5000, 6000, 7000, 8000),
            end = c(1500, 2500, 3500, 5500, 6500, 7500, 20000000)
        ),
     gene_id = c("FBgn001", "FBgn002", "FBgn003", "FBgn004", "FBgn005", "FBgn006", "FBgn007"),
        gene_name = c("ap", "dpr1", "side", "mav", "geneE", "geneF", "LargeTestGene")
    )
    data_dir <- system.file("extdata", package = "damidBind")
    loaded_data <- load_data_peaks(
        binding_profiles_path = data_dir,
        peaks_path = data_dir,
        ensdb_genes = mock_genes_gr,
```

```
        quantile_norm = TRUE
    )
    diff_results <- differential_binding(
        loaded_data,
        cond = c("L4 Neurons" = "L4",
                 "L5 Neurons" = "L5")
    )
    return(diff_results)
}
diff_results <- .generate_example_results()

# Generate a default volcano plot
plot_volcano(diff_results)
```

`quantile_normalisation` *Quantile normalisation (native R code version)*

Description

Performs quantile normalisation of a numeric matrix in native R, matching the algorithm used by 'preprocessCore' (including its tie-handling rule).

Usage

```
quantile_normalisation(x)
```

Arguments

`x`	A numeric matrix; rows are features (e.g., genes), columns are samples/arrays.

Details

This function is a native R implementation of the standard quantile normalisation algorithm. It is designed to be a drop-in replacement for, and produce identical results to, the function of the same name in the 'preprocessCore' package.

This native R version is provided within 'damidBind' to avoid known issues where the 'preprocessCore' package can lead to errors or cause R to crash on some Linux systems due to conflicts with OpenMP and/or BLAS/LAPACK library configurations. By providing this native R implementation, 'damidBind' ensures it works reliably for all users without requiring them to recompile dependencies or manage system environment variables.

This implementation exactly mirrors the behaviour of the 'preprocessCore' library's classic quantile normalisation, including its specific handling of ties: average ranks are computed for ties, and if the fractional part of a rank is greater than 0.4, the output value is the average of the two adjacent quantile means; otherwise, only the lower (floored) quantile mean is used.

The function stops if any NA, Inf, or NaN values are present in `x`.

Value

A numeric matrix of the same dimensions as x, quantile normalised.

Examples

```
set.seed(1)
x <- matrix(rnorm(9), nrow = 3)
quantile_normalisation(x)
```

---

reduce_regions *Reduce a list of GRanges to unique, non-overlapping regions*

Description

Takes a list of GRanges objects (e.g., peak sets from multiple samples), combines them, and merges any overlapping or adjacent regions into a single, minimal set of genomic intervals.

Usage

```
reduce_regions(peaks)
```

Arguments

peaks	A list of GRanges objects.

Value

A GRanges object containing the reduced (union) regions, with a ‘name‘ metadata column in the format "chr:start-end".

Examples

```
# Create a list of GRanges objects with overlapping regions
gr1 <- GenomicRanges::GRanges("chr1", IRanges::IRanges(c(100, 200), width = 50))
gr2 <- GenomicRanges::GRanges("chr1", IRanges::IRanges(c(120, 300), width = 50))
gr_list <- list(gr1, gr2)

# Reduce the list to a single set of non-overlapping regions
reduced <- reduce_regions(gr_list)
print(reduced)
# The result combines overlapping regions [100-149] and [120-169] into [100-169].
```

`sample_labels_by_isolation` *Sample data points based on local isolation*

Description

An issue with labelling points on dense plots (e.g., volcano plots) is that high point density prevents clear labelling, even with tools like 'ggrepel'. This function addresses this by retaining isolated points while sampling from points in higher-density regions. It takes a dataframe with Cartesian coordinates and returns a logical vector identifying which points to select for labelling. The result is a less cluttered plot where labels are present even in crowded areas, providing a better representation of the underlying data.

Usage

```
sample_labels_by_isolation(
  df,
  x_col,
  y_col,
  r,
  k_priority = 30,
  scale = TRUE,
  k_search = 30,
  k_for_r = 5
)
```

Arguments

`df`	A dataframe containing the point coordinates.
`x_col`	A character string with the name of the column containing x-coordinates.
`y_col`	A character string with the name of the column containing y-coordinates.
`r`	The exclusion radius. This can be a positive numeric value or the string "auto". If "auto", the radius is calculated as the median distance to the 'k_for_r'-th nearest neighbour across all points. A smaller 'r' will result in more points being kept. Note: The interpretation of 'r' depends on whether 'scale' is 'TRUE'.
`k_priority`	An integer for calculating the isolation priority score. Must be less than or equal to 'k_search'. Default: 30.
`scale`	A logical value. If 'TRUE', the coordinate data is centred and scaled (using 'scale(center=TRUE, scale=TRUE)') before distance calculations. Defaults: 'TRUE'.
`k_search`	The maximum number of neighbours to find in the initial KNN search. This value must be greater than or equal to both 'k_priority' and 'k_for_r'. Default: 30.
`k_for_r`	An integer specifying which neighbour to use for the 'auto' 'r' calculation. Default: 5.

Details

The algorithm in detail: 1. If ‘scale = TRUE‘, the coordinate data is centred and scaled. 2. An exact k-nearest neighbour (KNN) search for all points is conducted using the ‘dbscan::kNN‘ function. 3. A priority score is calculated for each point, defined as the median distance to its ‘k_priority‘ nearest neighbours, and the list of points sorted by this score. 4. The function iterates through the sorted list of points in descending order: a. If a point has not yet been processed, it is marked as ’processed’ and ’kept’. b. A radius search is performed around this point using ‘dbscan::frNN‘. All neighbours within the specified exclusion radius ‘r‘ are then marked as ’processed’ and will be ignored in subsequent iterations. 5. A logical vector is returned, where ‘TRUE‘ corresponds to a point that should be kept for labelling.

Value

A logical vector of length ‘nrow(df)‘. ‘TRUE‘ indicates the point at that index should be kept for labelling.

Examples

```
library(ggplot2)
library(ggrepel)

# Generate sample data with a dense cluster
set.seed(42)
n_points <- 1000
cluster_data <- data.frame(
  x = rnorm(n_points, mean = 5, sd = 1),
  y = rnorm(n_points, mean = 5, sd = 1),
  label = paste("Point", 1:n_points)
)

# Use the function to get a logical vector for filtering
kept_labels <- sample_labels_by_isolation(
  df = cluster_data,
  x_col = "x",
  y_col = "y",
  scale = FALSE,
  r = "auto",
  k_priority = 30,
  k_search = 30,
  k_for_r = 5
)

# Create the label dataframe for ggplot
label_df <- cluster_data[kept_labels, ]

# Plot the results
ggplot(cluster_data, aes(x = x, y = y)) +
  geom_point(colour = "grey70", alpha = 0.7) +
  geom_point(data = label_df, colour = "firebrick") +
  geom_text_repel(
    data = label_df,
```

```
  aes(label = label),
  min.segment.length = 0,
  box.padding = 0.25,
  max.overlaps = Inf
) +
coord_fixed() +
labs(
  title = "Sampled Labels",
  subtitle = paste(sum(kept_labels), "of", nrow(cluster_data), "points labelled"),
  caption = "Red points are selected for labelling."
) +
theme_bw()
```

Index