Package 'TransProR'

December 10, 2024

Type Package

Title Analysis and Visualization of Multi-Omics Data

Version 0.0.6

Maintainer Dongyue Yu <yudongyue@mail.nankai.edu.cn>

Description A tool for comprehensive transcriptomic data analysis, with a focus on transcript-level data preprocessing, expression profiling, differential expression analysis, and functional enrichment. It enables researchers to identify key biological processes, disease biomarkers, and gene regulatory mechanisms. 'TransProR' is aimed at researchers and bioinformaticians working with RNA-Seq data, providing an intuitive framework for in-depth analysis and visualization of transcriptomic datasets. The package includes comprehensive documentation and usage examples to guide users through the entire analysis pipeline. The differential expression analysis methods incorporated in the package include 'limma' (Ritchie et al., 2015, <doi:10.1093/nar/gkv007>; Smyth, 2005, <doi:10.1007/0-387-29362-0_23>), 'edgeR' (Robinson et al., 2010, <doi:10.1093/bioinformatics/btp616>), 'DESeq2' (Love et al., 2014, <doi:10.1186/s13059-014-0550-8>), and Wilcoxon tests (Li et al., 2022, <doi:10.1186/s13059-022-02648-4>), providing flexible and robust approaches to RNA-Seq data analysis. For more information, refer to the package vignettes and related publications.

Imports magrittr, stats, data.table, dplyr, rlang, tibble, sva, DESeq2, edgeR, limma, ggplot2, ggVennDiagram, ggdensity, ggpubr, ggtree, hrbrthemes, grid, ggraph, tidygraph, tidyr, stringr, geomtextpath, ggalt, ggnewscale, ggtreeExtra, Hmisc, circlize, graphics, spiralize, ComplexHeatmap, grDevices

Suggests prettydoc, knitr, rmarkdown

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/SSSYDYSSS/TransProRBook

BugReports https://github.com/SSSYDYSSS/TransProR/issues

LazyData true

RoxygenNote 7.3.2

Depends R (>= 4.3.0)

NeedsCompilation no

Author Dongyue Yu [aut, cre, cph] (<https://orcid.org/0000-0002-7041-2073>)

Repository CRAN

Date/Publication 2024-12-10 21:30:09 UTC

Contents

`add_boxplot` *Add a boxplot layer to a 'ggtree' plot*

Description

This function adds a boxplot layer to an existing 'ggtree' plot object using ggtreeExtra's geom_fruit for boxplots. It is primarily used to display statistical summaries of the data related to gene expressions or other metrics.

Usage

```
add_boxplot(
  p,
  data,
  fill_color = "#f28131",
  alpha = 0.6,
  offset = 0.22,
  pwidth = 0.5
)
```

Arguments

`p`	An existing ggtree plot object.
`data`	A data frame containing the data to be plotted. Expected to have columns for 'Sample' and 'value'.
`fill_color`	A character string specifying the fill color for the boxplots. Default is "#f28131".
`alpha`	Numeric value for the transparency of the boxplots. Default is 0.6.

offset	Numeric value, the position of the boxplot on the x-axis relative to its gene name. Default is 0.22.
pwidth	Numeric value, the width of the boxplot. Default is 0.5.

Value

A ‘ggtree‘ plot object with the added boxplot layer.

Examples

```
# Check and load required packages
if (requireNamespace("ggtreeExtra", quietly = TRUE) &&
 requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggtreeExtra)
  library(ggplot2)

  file_path <- system.file("extdata", "p_tree_test.rds", package = "TransProR")
  p <- readRDS(file_path)

  # Create boxplot data frame
  boxplot_data <- data.frame(
    Sample = rep(c("Species_A", "Species_B", "Species_C", "Species_D"), each = 30),
    value = c(
      rnorm(30, mean = 5, sd = 1),   # Data for Species_A
      rnorm(30, mean = 7, sd = 1.5), # Data for Species_B
      rnorm(30, mean = 6, sd = 1.2), # Data for Species_C
      rnorm(30, mean = 8, sd = 1.3)  # Data for Species_D
    )
  )

  # Call add_boxplot function to add boxplot layer
  p_with_boxplot <- add_boxplot(p, boxplot_data)
} else {
  message("Required packages 'ggtreeExtra' and 'ggplot2' are not installed.")
}
```

---

add_new_tile_layer *Add a new tile layer with dynamic scales to a ‘ggtree‘ plot*

Description

This function adds a new tile layer to an existing ‘ggtree‘ plot object, allowing for separate scales for fill and alpha transparency. This is useful when you want to add additional data layers without interfering with the existing scales in the plot. It utilizes the ’ggnewscale’ package to reset scales for new layers.

Usage

```
add_new_tile_layer(
  p,
  data,
  gene_colors,
  gene_label,
  alpha_value = c(0.3, 0.9),
  offset = 0.02,
  pwidth = 2
)
```

Arguments

p	An existing ggtree plot object.
data	A data frame containing the data to be plotted. Expected to have columns for 'Sample', 'Gene', and 'value'.
gene_colors	A named vector of colors for genes.
gene_label	A character string used as a label in the legend for the genes.
alpha_value	A numeric or named vector for setting the alpha scale based on values.
offset	Numeric value, the position of the tile on the x-axis relative to its gene name. Default is 0.02.
pwidth	Numeric value, the width of the tile. Default is 2.

Value

A 'ggtree' plot object with the added tile layer and new scales.

Examples

```
# Check and load required packages
if (requireNamespace("ggtreeExtra", quietly = TRUE) &&
 requireNamespace("ggplot2", quietly = TRUE) &&
  requireNamespace("ggnewscale", quietly = TRUE)) {
  library(ggtreeExtra)
  library(ggplot2)
  library(ggnewscale)

  file_path <- system.file("extdata", "p_tree_test.rds", package = "TransProR")
  p <- readRDS(file_path)

  # Create new expression data
  new_expression_data <- data.frame(
    Sample = rep(c("Species_A", "Species_B", "Species_C", "Species_D"), each = 3),
    Gene = rep(c("Gene6", "Gene7", "Gene8"), times = 4),
   Value = runif(12, min = 0, max = 1)  # Randomly generate expression values between 0 and 1
  )

  # Define new gene colors
```

```
  new_gene_colors <- c(
    Gene6 = "#0b5f63",
    Gene7 = "#074d41",
    Gene8 = "#1f5e27"
  )

  # Define gene label and alpha values
  gene_label <- "New Genes"
  alpha_value <- c(0.3, 0.9)

  # Add new tile layer
  p_with_new_layer <- add_new_tile_layer(
    p,
    new_expression_data,
    new_gene_colors,
    gene_label,
    alpha_value,
    offset = 0.02,
    pwidth = 2
  )
} else {
 message("Required packages 'ggtreeExtra', 'ggplot2', and 'ggnewscale' are not installed.")
}
```

`adjust_alpha_scale` *Adjust Alpha Scale for Data Visualization*

Description

This function dynamically adjusts the transparency scale for visualizations, especially useful when the range of data values varies significantly across different sources. It modifies the transparency scale based on the range of values present in the data, ensuring that the visualization accurately reflects variations within the data.

Usage

```
adjust_alpha_scale(data, name, range = c(0.2, 0.8))
```

Arguments

`data`	A data frame containing the values for which the alpha scale is to be adjusted.
`name`	Character string that will be used as the title of the legend in the plot.
`range`	Numeric vector of length 2 specifying the range of alpha values, defaults to c(0.2, 0.8).

Value

A ggplot2 alpha scale adjustment layer.

Examples

```
# Assuming 'data' is a DataFrame with a 'value' column
plot_data <- data.frame(value = c(10, 20, 30, 40, 50))
ggplot2::ggplot(plot_data, ggplot2::aes(x = 1:nrow(plot_data), y = value)) +
  ggplot2::geom_point(ggplot2::aes(alpha = value)) +
  adjust_alpha_scale(plot_data, "Transparency Scale")
```

`adjust_color_tone` *Adjust Color Tone by Modifying Saturation and Luminance*

Description

This function adjusts the saturation and luminance of a given color. It works by converting the color from RGB to Luv color space, applying the scaling factors to the saturation and luminance, and then converting it back to RGB.

Usage

```
adjust_color_tone(color, saturation_scale, luminance_scale)
```

Arguments

`color`	A color in hexadecimal format (e.g., "#FF0000") or a valid R color name.
`saturation_scale`	Numeric, the scaling factor for saturation (values < 1 decrease saturation, values > 1 increase saturation).
`luminance_scale`	Numeric, the scaling factor for luminance (values < 1 darken the color, values > 1 lighten the color).

Value

Returns a color in hexadecimal format adjusted according to the provided scales.

Examples

```
adjusted_color <- adjust_color_tone("#FF0000", saturation_scale = 0.8, luminance_scale = 1.2)
print(adjusted_color)
```

`adjust_export_pathway` *Adjust and Export Pathway Analysis Results*

Description

This function processes a dataframe containing fgsea results. It adjusts pathway names by removing underscores, converting to lowercase, then capitalizing the first letter, and joining the components with spaces. It selects and merges the top upregulated and downregulated pathways based on enrichment score (ES) and p-value.

Usage

```
adjust_export_pathway(fgseaRes, nTop = 10)
```

Arguments

`fgseaRes`	Dataframe containing fgsea results with columns 'pathway', 'ES', and 'pval'.
`nTop`	Integer, number of top pathways to select based on the p-value.

Value

A vector containing combined top upregulated and downregulated pathways.

Examples

```
# Create a synthetic fgseaRes dataframe
fgseaRes <- data.frame(
 pathway = c("KEGG_APOPTOSIS",
             "GO_CELL_CYCLE",
             "REACTOME_DNA_REPAIR",
             "KEGG_METABOLISM",
             "GO_TRANSPORT"),
 ES = c(0.45, -0.22, 0.56, -0.35, 0.33),
 pval = c(0.001, 0.02, 0.0003, 0.05, 0.01)
)

# Run the function to get top pathways
result <- adjust_export_pathway(fgseaRes = fgseaRes, nTop = 2)
```

all_degs_venn *All DEGs Venn Diagram Data*

Description

A dataset containing the differentially expressed genes (DEGs) from four different statistical analysis methods: DESeq2, edgeR, limma, and Wilcoxon test. This dataset is used for generating Venn diagrams to compare the overlap of DEGs identified by different methods.

Usage

```
data(all_degs_venn)
```

Format

A list with the following components:

DESeq2 A vector of gene IDs or gene symbols identified as DEGs by the DESeq2 method.

edgeR A vector of gene IDs or gene symbols identified as DEGs by the edgeR method.

limma A vector of gene IDs or gene symbols identified as DEGs by the limma method.

Wilcoxon_test A vector of gene IDs or gene symbols identified as DEGs by the Wilcoxon test method.

Source

The data was derived from differential expression analyses performed on a gene expression dataset using four commonly used statistical methods (DESeq2, edgeR, limma, and Wilcoxon test).

Examples

```
data(all_degs_venn)
# Example of plotting a Venn diagram using the dataset

edge_colors <- c("#1b62bb","#13822e","#332c3a","#9e2d39")
name_color <- c("#1b64bb","#13828e","#337c3a","#9e9d39")
fill_colors <- c("#e3f2fa", "#0288d1")

Contrast_degs_venn <- Contrast_Venn(all_degs_venn, edge_colors, name_color, fill_colors)
```

circos_fruits *Add multiple layers to a 'ggtree' plot for visualizing gene expression and enrichment data*

Description

This function sequentially adds multiple layers to a 'ggtree' plot, including gene expression data, boxplots for statistical summaries, and additional tile layers for pathway enrichment scores from SSGSEA and GSVA analyses. It utilizes separate functions for adding each type of layer and allows for the specification of gene colors as well as adjustments in aesthetics for each layer. The function is designed to work with specific data structures and assumes all functions for adding layers are defined and available.

Usage

```
circos_fruits(
  p,
  long_format_HeatdataDeseq,
  ssgsea_kegg_HeatdataDeseq,
  gsva_kegg_HeatdataDeseq,
  gene_colors
)
```

Arguments

p	A 'ggtree' plot object to which the data and layers will be added.
long_format_HeatdataDeseq	A data frame containing gene expression data with columns for 'Samples', 'Genes', and 'Values'.
ssgsea_kegg_HeatdataDeseq	A data frame containing SSGSEA analysis results with columns for 'Samples', 'Genes', and 'Values'.
gsva_kegg_HeatdataDeseq	A data frame containing GSVA analysis results with columns for 'Samples', 'Genes', and 'Values'.
gene_colors	A named vector of colors for genes, used for coloring tiles in different layers.

Value

A 'ggtree' plot object with multiple layers added for comprehensive visualization.

Examples

```
# Check and load required packages
if (requireNamespace("ggtreeExtra", quietly = TRUE) &&
 requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggtreeExtra)
```

```
library(ggplot2)

# Example data for gene expression, SSGSEA, and GSVA
file_path <- system.file("extdata", "p_tree_test.rds", package = "TransProR")
p <- readRDS(file_path)

# Create gene expression data frame (long_format_HeatdataDeseq)
long_format_HeatdataDeseq <- data.frame(
  Sample = rep(c("Species_A", "Species_B", "Species_C", "Species_D"), each = 5),
  Genes = rep(paste0("Gene", 1:5), times = 4),
 Value = runif(20, min = 0, max = 1)  # Randomly generate expression values between 0 and 1
)

# Create SSGSEA analysis results data frame (ssgsea_kegg_HeatdataDeseq)
ssgsea_kegg_HeatdataDeseq <- data.frame(
  Sample = rep(c("Species_A", "Species_B", "Species_C", "Species_D"), each = 3),
  Genes = rep(c("Pathway1", "Pathway2", "Pathway3"), times = 4),
 Value = runif(12, min = 0, max = 1)  # Randomly generate enrichment scores between 0 and 1
)

# Create GSVA analysis results data frame (gsva_kegg_HeatdataDeseq)
gsva_kegg_HeatdataDeseq <- data.frame(
  Sample = rep(c("Species_A", "Species_B", "Species_C", "Species_D"), each = 4),
  Genes = rep(c("PathwayA", "PathwayB", "PathwayC", "PathwayD"), times = 4),
 Value = runif(16, min = 0, max = 1)  # Randomly generate enrichment scores between 0 and 1
)

# Define gene and pathway colors (named vector), including all genes and pathways
gene_colors <- c(
  # Genes for gene expression
  Gene1 = "#491588",
  Gene2 = "#301b8d",
  Gene3 = "#1a237a",
  Gene4 = "#11479c",
  Gene5 = "#0a5797",
  # Pathways for SSGSEA
  Pathway1 = "#0b5f63",
  Pathway2 = "#074d41",
  Pathway3 = "#1f5e27",
  # Pathways for GSVA
  PathwayA = "#366928",
  PathwayB = "#827729",
  PathwayC = "#a1d99b",
  PathwayD = "#c7e9c0"
)

# Call circos_fruits function to add multiple layers
final_plot <- circos_fruits(
  p,
  long_format_HeatdataDeseq,
  ssgsea_kegg_HeatdataDeseq,
  gsva_kegg_HeatdataDeseq,
  gene_colors
```

```
  )
} else {
  message("Required packages 'ggtreeExtra' and 'ggplot2' are not installed.")
}
```

`Combat_Normal` *Process and Correct Batch Effects in TCGA's normal tissue and GTEX Data*

Description

The function first extracts histological types from the provided TCGA's normal tissue data set. After displaying these types, the user is prompted to input specific types to retain. The data is then filtered based on this input. The GTEX and TCGA's normal tissue datasets are then combined and batch corrected.

Note: This function assumes that TCGA's normal samples and GTEX samples represent different batches.

Usage

```
Combat_Normal(
  TCGA_normal_data_path,
  gtex_data_path,
  CombatNormal_output_path,
  auto_mode = FALSE,
  default_input = "11,12"
)
```

Arguments

`TCGA_normal_data_path`
: The path to the tumor data stored in an RDS file.

`gtex_data_path`
: The path to the GTEX data stored in an RDS file.

`CombatNormal_output_path`
: A character string specifying the path where the output RDS file will be saved.

`auto_mode`
: Logical. If set to TRUE, the function will not prompt the user for input and will instead use the values provided in default_input. Default is FALSE.

`default_input`
: Character string. When auto_mode is TRUE, this parameter specifies the default TGCA's normal tissue types to be retained. It should be provided as a comma-separated string (e.g., "11,12").

Details

This function takes a TCGA's normal tissue data set and a pre-saved GTEX data set, asks the user for specific TCGA's normal tissue types to retain, then merges the two datasets. The merged dataset is then corrected for batch effects using the ComBat_seq function from the 'sva' package.

The ComBat_seq function from the 'sva' package is used to correct batch effects. The function requires the 'sva' package to be installed and loaded externally.

The example code uses 'tempfile()' to generate temporary paths dynamically during execution. These paths are valid during the 'R CMD check' process, even if no actual files exist, because 'tempfile()' generates a unique file path that does not depend on the user's file system. Using 'tempfile()' ensures that the example code does not rely on specific external files and avoids errors during 'R CMD check'. CRAN review checks for documentation correctness and syntax parsing, not the existence of actual files, as long as the example code is syntactically valid.

Value

A data.frame with corrected values after the ComBat_seq adjustment. Note that this function also saves the combat_count_df data as an RDS file at the specified output path.

See Also

ComBat_seq

Examples

```
TCGA_normal_file <- system.file("extdata",
                                "SKCM_Skin_TCGA_exp_normal_test.rds",
                                package = "TransProR")
gtex_file <- system.file("extdata", "Skin_SKCM_Gtex_test.rds", package = "TransProR")
output_file <- file.path(tempdir(), "SKCM_Skin_Combat_Normal_TCGA_GTEX_count.rds")

SKCM_Skin_Combat_Normal_TCGA_GTEX_count <- Combat_Normal(
  TCGA_normal_data_path = TCGA_normal_file,
  gtex_data_path = gtex_file,
  CombatNormal_output_path = output_file,
  auto_mode = TRUE,
  default_input = "skip"
)
head(SKCM_Skin_Combat_Normal_TCGA_GTEX_count)[1:5, 1:5]
```

---

combat_tumor *Process and Correct Batch Effects in Tumor Data*

Description

The function first extracts histological types from the provided tumor data set. After displaying these types, the user is prompted to input specific types to retain. The data is then filtered based on this input.

Note: This example assumes that different tumor types represent different batches in a general sense. Users need to adjust the batch and group vectors based on real-life scenarios.

Usage

```
combat_tumor(
  tumor_data_path,
  CombatTumor_output_path,
  auto_mode = FALSE,
  default_input = "01,06"
)
```

Arguments

`tumor_data_path`	The path to the tumor data stored in an RDS file.
`CombatTumor_output_path`	A character string specifying the path where the output RDS file will be saved.
`auto_mode`	Logical. If set to TRUE, the function will not prompt the user for input and will instead use the values provided in default_input. Default is FALSE.
`default_input`	Character string. When auto_mode is TRUE, this parameter specifies the default tumor types to be retained. It should be provided as a comma-separated string (e.g., "01,06").

Details

This function takes a tumor data set, asks the user for specific tumor types to retain, and then corrects for batch effects using the ComBat_seq function from the 'sva' package.

The ComBat_seq function from the sva package is used to correct batch effects. The function requires the 'sva' package to be installed and loaded externally.

Value

A data.frame with corrected values after the ComBat_seq adjustment. Note that this function also saves the combat_count_df data as an RDS file at the specified output path.

See Also

ComBat_seq

Examples

```
tumor_file <- system.file("extdata",
                          "SKCM_Skin_TCGA_exp_tumor_test.rds",
                          package = "TransProR")
output_file <- file.path(tempdir(), "SKCM_combat_count.rds")

  SKCM_combat_count <- combat_tumor(
  tumor_data_path = tumor_file,
  CombatTumor_output_path = output_file,
  auto_mode = TRUE,
  default_input = "01,06"
)

head(SKCM_combat_count)[1:5, 1:5]
```

---

compare_merge *Compare and merge specific columns from two DEG data frames*

Description

This function takes two DEG data frames, inner joins them by a specified gene column, checks if a specified column is identical across both data frames, and merges them if they are. The resulting data frame will have a merged column named after the compared column.

Usage

```
compare_merge(df1, df2, by_gene, compare_col, suffixes, df_name)
```

Arguments

df1	First data frame.
df2	Second data frame.
by_gene	Column name by which to join the data frames, typically "Gene".
compare_col	Column to compare for identity, which will also be the name of the merged column.
suffixes	Suffixes to use for non-identical column names in the joined data frame.
df_name	Name to assign to the resulting data frame for identification.

Value

A data frame with processed columns.

Examples

```
# Create simulated DESeq2 data
DEG_deseq2 <- data.frame(
  Gene = c("Gene1", "Gene2", "Gene3", "Gene4", "Gene5"),
  change = c("up", "down", "no_change", "up", "down"),
  log2FoldChange = c(2.5, -3.2, 0.1, 1.8, -2.5),
  pvalue = c(0.01, 0.05, 0.9, 0.02, 0.03)
)

# Display the first 5 rows of the DESeq2 data
head(DEG_deseq2, 5)

# Create simulated edgeR data
DEG_edgeR <- data.frame(
  Gene = c("Gene1", "Gene2", "Gene3", "Gene4", "Gene5"),
  change = c("up", "down", "no_change", "no_change", "up"),
  log2FoldChange = c(2.3, -3.1, 0.2, 0.1, 2.7),
  pvalue = c(0.02, 0.04, 0.8, 0.6, 0.01)
)

# Display the first 5 rows of the edgeR data
head(DEG_edgeR, 5)

# Merge the DESeq2 and edgeR data
deseq2_edgeR <- compare_merge(
  df1 = DEG_deseq2,
  df2 = DEG_edgeR,
  by_gene = "Gene",
  compare_col = "change",
  suffixes = c("_1", "_2"),
  df_name = "deseq2_edgeR"
)
```

Contrast_Venn *Function to Create a Venn Diagram of DEGs with Custom Colors*

Description

This function creates a Venn Diagram using the 'ggVennDiagram' package. It allows customization of various aesthetic elements of the diagram, including colors.

Usage

```
Contrast_Venn(
  all_degs_venn,
  edge_colors,
  name_color,
  fill_colors,
```

```
  label_size = 4,
  edge_size = 3
)
```

Arguments

all_degs_venn	A list of DEG sets for Venn Diagram creation.
edge_colors	A vector of colors for the edges of the Venn Diagram sets.
name_color	A vector of colors for the names of the sets in the Venn Diagram.
fill_colors	A vector of two colors for the gradient fill of the Venn Diagram.
label_size	The size of the labels showing the number of elements in each set (default is 4).
edge_size	The size of the edges of the Venn Diagram sets (default is 3).

Value

A 'ggplot' object representing the Venn Diagram.

Examples

```
data("all_degs_venn", package = "TransProR")

edge_colors <- c("#1b62bb","#13822e","#332c3a","#9e2d39")
name_color <- c("#1b64bb","#13828e","#337c3a","#9e9d39")
fill_colors <- c("#e3f2fa", "#0288d1")

Contrast_degs_venn <- Contrast_Venn(all_degs_venn, edge_colors, name_color, fill_colors)
```

---

create_base_plot *Create a base plot with gene expression data on a phylogenetic tree*

Description

This function creates a base plot using 'ggtree' and 'ggtreeExtra' libraries, adding gene expression data as colored tiles to the plot. It allows for dynamic coloring of the genes and includes adjustments for alpha transparency based on the expression value.

Usage

```
create_base_plot(p, data, gene_colors, gene_label = "Gene")
```

Arguments

p	A ggtree plot object to which the data will be added.
data	A data frame containing gene expression data with columns for Samples, Genes, and Values.
gene_colors	A named vector of colors for genes.
gene_label	A character string used as a label in the legend for the genes. Default is "Gene".

Value

A ‘ggtree‘ plot object with the gene expression data added.

Examples

```
# Check and load required packages
if (requireNamespace("ggtreeExtra", quietly = TRUE) &&
 requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggtreeExtra)
  library(ggplot2)

  file_path <- system.file("extdata", "p_tree_test.rds", package = "TransProR")
  p <- readRDS(file_path)

  # Create gene expression data frame
  expression_data <- data.frame(
    Sample = rep(c("Species_A", "Species_B", "Species_C", "Species_D"), each = 5),
    Gene = rep(paste0("Gene", 1:5), times = 4),
   Value = runif(20, min = 0, max = 1)  # Randomly generate expression values between 0 and 1
  )

  # Define gene colors (named vector)
  gene_colors <- c(
    Gene1 = "#491588",
    Gene2 = "#301b8d",
    Gene3 = "#1a237a",
    Gene4 = "#11479c",
    Gene5 = "#0a5797"
  )

  # Call create_base_plot function to add gene expression data
  p <- create_base_plot(p, expression_data, gene_colors)
} else {
  message("Required packages 'ggtreeExtra' and 'ggplot2' are not installed.")
}
```

---

deg_filter *Function to Filter Differentially Expressed Genes (DEGs)*

Description

This function filters out genes based on their expression change status. It returns the names of genes which are not "stable".

Usage

```
deg_filter(df)
```

Arguments

df	A data frame containing gene expression data.

Value

A vector of gene names that are differentially expressed.

Examples

```
DEG_deseq2_file <- system.file("extdata", "DEG_deseq2.rds", package = "TransProR")
DEG_deseq2 <- readRDS(DEG_deseq2_file)
DEG_deseq2_test <- deg_filter(DEG_deseq2)
```

DESeq2_analyze *Differential Gene Expression Analysis using 'DESeq2'*

Description

'DESeq2': Differential gene expression analysis based on the negative binomial distribution. This function utilizes the 'DESeq2' package to conduct differential gene expression analysis. It processes tumor and normal expression data, applies DESeq2 analysis, and outputs the results along with information on gene expression changes.

Usage

```
DESeq2_analyze(
  tumor_file,
  normal_file,
  output_file,
  logFC = 2.5,
  p_value = 0.01
)
```

Arguments

tumor_file	Path to the tumor data file (RDS format).
normal_file	Path to the normal data file (RDS format).
output_file	Path to save the output DEG data (RDS format).
logFC	Threshold for log fold change.
p_value	Threshold for p-value.

Details

The DESeq2 methodology is based on modeling count data using a negative binomial distribution, which allows for handling the variability observed in gene expression data, especially in small sample sizes. This approach is well-suited for RNA-Seq data analysis.

Value

A data frame of differential expression results.

References

DESeq2:Differential gene expression analysis based on the negative binomial distribution. For more information, visit the page: https://docs.gdc.cancer.gov/Data/Bioinformatics_Pipelines/Expression_mRNA_Pipeline/

Examples

```
# Define file paths for tumor and normal data from the data folder
tumor_file <- system.file("extdata",
                          "removebatch_SKCM_Skin_TCGA_exp_tumor_test.rds",
                          package = "TransProR")
normal_file <- system.file("extdata",
                           "removebatch_SKCM_Skin_Normal_TCGA_GTEX_count_test.rds",
                           package = "TransProR")
output_file <- file.path(tempdir(), "DEG_DESeq2.rds")

DEG_DESeq2 <- DESeq2_analyze(
  tumor_file = tumor_file,
  normal_file = normal_file,
  output_file = output_file,
  2.5,
  0.01
)

# View the top 5 rows of the result
head(DEG_DESeq2, 5)
```

---

drawLegends *Draw Dual-Sided Legends on a Plot*

Description

This function creates two sets of legends, one on the left and one on the right side of a plot. It displays color-coded legends with labels corresponding to different data categories. Each legend entry consists of a colored rectangle and a text label. The left side legend has text aligned to the right of the color block, while the right side legend has text aligned to the left of the color block.

Usage

```
drawLegends(
  labels,
  colors,
  legend_width,
  x_positions,
```

```
  y_position,
  just_positions,
  text_alignments,
  font_size
)
```

Arguments

labels	Vector of labels for the legends.
colors	Vector of colors corresponding to the labels.
legend_width	The width of each legend viewport expressed in grid units.
x_positions	Numeric vector of length 2 specifying the x-positions of the left and right legends.
y_position	The y-position common for both legends, expressed as a fraction of the plot height.
just_positions	List of two vectors, each specifying the horizontal and vertical justification for the legends.
text_alignments	List of two character strings specifying text alignments for the legends ('left' or 'right').
font_size	Numeric value specifying the font size for the legend labels.

Value

Invisible. This function is called for its side effects of drawing legends on a plot.

Examples

```
labels <- c("Label1", "Label2", "Label3", "Label4", "Label5", "Label6")
colors <- c("#ff0000", "#00ff00", "#0000ff", "#ffff00", "#ff00ff", "#00ffff")

# Convert to 'unit' objects for grid
grid::grid.roundrect(
  x = grid::unit(0.5, "npc"),  # "npc" stands for normalized parent coordinates
  y = grid::unit(0.5, "npc"),
  width = grid::unit(0.1, "npc"),
  height = grid::unit(0.05, "npc"),
  gp = grid::gpar(fill = "red"),
  r = grid::unit(0.1, "npc")  # rounding radius
)

# Example of drawing legends with specific labels and colors
drawLegends(labels, colors, grid::unit(2, "cm"), c(0.225, 0.75), 0.5,
            list(c("left", "center"), c("right", "center")),
            list("right", "left"), 10)
```

edgeR_analyze *Differential Gene Expression Analysis using 'edgeR'*

Description

This function performs differential gene expression analysis using the 'edgeR' package. It reads tumor and normal expression data, merges them, filters low-expressed genes, normalizes the data, performs edgeR analysis, and outputs the results along with information on gene expression changes.

Usage

```
edgeR_analyze(
  tumor_file,
  normal_file,
  output_file,
  logFC_threshold = 2.5,
  p_value_threshold = 0.01
)
```

Arguments

tumor_file	Path to the tumor data file (RDS format).
normal_file	Path to the normal data file (RDS format).
output_file	Path to save the output DEG data (RDS format).
logFC_threshold	Threshold for log fold change for marking up/down-regulated genes.
p_value_threshold	Threshold for p-value for filtering significant genes.

Value

A data frame of differential expression results.

References

edgeR: Differential analysis of sequence read count data. For more information, visit the edgeR Bioconductor page: https://www.bioconductor.org/packages/release/bioc/vignettes/edgeR/inst/doc/edgeRUsersGuide.pdf

Examples

```
# Define file paths for tumor and normal data from the data folder
tumor_file <- system.file("extdata",
                          "removebatch_SKCM_Skin_TCGA_exp_tumor_test.rds",
                          package = "TransProR")
normal_file <- system.file("extdata",
                           "removebatch_SKCM_Skin_Normal_TCGA_GTEX_count_test.rds",
                           package = "TransProR")
```

```
output_file <- file.path(tempdir(), "DEG_edgeR.rds")

DEG_edgeR <- edgeR_analyze(
  tumor_file = tumor_file,
  normal_file = normal_file,
  output_file = output_file,
  2.5,
  0.01
)

# View the top 5 rows of the result
head(DEG_edgeR, 5)
```

`enrichment_circlize` *Draw Chord Diagram with Legends*

Description

This function creates a chord diagram from a specified dataframe and draws two sets of legends for it. It adjusts the track height of the chord diagram to optimize space and uses specified colors for the grid. Legends are drawn at specified positions with configurable text alignments and font sizes.

Usage

```
enrichment_circlize(
  all_combined_df,
  original_colors,
  labels,
  colors,
  labels2,
  colors2,
  font_size = 10
)
```

Arguments

`all_combined_df`	A dataframe containing the matrix for the chord diagram.
`original_colors`	A vector of colors for the grid columns of the chord diagram.
`labels`	A vector of labels for the first legend.
`colors`	A vector of colors corresponding to the first legend's labels.
`labels2`	A vector of labels for the second legend.
`colors2`	A vector of colors corresponding to the second legend's labels.
`font_size`	The font size used for legend texts, defaults to 10.

Value

Invisible, primarily used for its side effects of drawing on a graphics device.

Examples

```
# Sample Chord Diagram Matrix
all_combined_df <- data.frame(
  A = c(10, 20, 30),
  B = c(15, 25, 35),
  C = c(5, 10, 15)
)
rownames(all_combined_df) <- c("A", "B", "C")

# Colors for the grid of the chord diagram (corresponding to columns of the matrix)
original_colors <- c("red", "green", "blue")

# Name the colors according to the sectors (A, B, C)
names(original_colors) <- colnames(all_combined_df)

# Labels and Colors for the First Legend
labels <- c("Label 1", "Label 2", "Label 3")
colors <- c("yellow", "purple", "cyan")

# Labels and Colors for the Second Legend
labels2 <- c("Label A", "Label B", "Label C")
colors2 <- c("orange", "pink", "brown")

# Font size for the legend texts (optional, default is 10)
font_size <- 10

# Call the enrichment_circlize function with the sample data
# This is just an example; the plot will be rendered in an appropriate graphics context
# such as RStudio's plot pane or an external plotting window.
plot1 <- enrichment_circlize(all_combined_df,
                             original_colors,
                             labels,
                             colors,
                             labels2,
                             colors2,
                             font_size
                             )
```

`enrichment_spiral_plots` *Create Spiral Plots with Legends Using 'spiralize' and 'ComplexHeatmap'*

Description

This function initializes a spiral plot, adds tracks for pathways and samples, and generates legends based on the sample and pathway information in the provided data frame. It uses 'spiralize' for the spiral plot and 'ComplexHeatmap' for handling legends.

Usage

```
enrichment_spiral_plots(results)
```

Arguments

results	A data frame containing 'Pathway', 'Sample', 'Value', 'PathwayColor', and 'SampleColor' columns.

Value

No return value, called for side effects. This function generates spiral plots and adds legends based on sample and pathway information.

Examples

```
# Example: Creating enrichment spiral plots with legends

# Define the results data frame
results <- data.frame(
  Pathway = c("Pathway1", "Pathway1", "Pathway2", "Pathway2", "Pathway3"),
  Sample = c("Sample1", "Sample1", "Sample2", "Sample2", "Sample3"),
  Value = c(20, 30, 15, 35, 25),
  PathwayColor = c("red", "red", "blue", "blue", "orange"),
  SampleColor = c("green", "green", "purple", "purple", "cyan"),
  stringsAsFactors = FALSE
)

# Create the enrichment spiral plots with legends
enrichment_spiral_plots(results)
```

---

enrich_circo_bar *Combine and Visualize Data with Circular Bar Chart*

Description

This function combines multiple data frames, arranges them, and visualizes the combined data in a Circular Bar Chart using the 'ggplot2' and 'ggalluvial' packages.

Usage

```
enrich_circo_bar(data_list)
```

Arguments

data_list	A list of data frames to be combined.

Value

A 'ggplot' object representing the Circular Bar Chart.

Examples

```
# Create sample data frames for each enrichment category

# 1. Biological Process (BP)
filtered_data_BP <- data.frame(
  Description = c(
    "immune response",
    "cell proliferation",
    "signal transduction",
    "apoptotic process",
    "metabolic process"
  ),
  Count = c(120, 85, 150, 60, 95),
  color = c(
    "#1f77b4",  # blue
    "#ff7f0e",  # orange
    "#2ca02c",  # green
    "#d62728",  # red
    "#9467bd"   # purple
  ),
  stringsAsFactors = FALSE
)

# 2. Cellular Component (CC)
filtered_data_CC <- data.frame(
  Description = c(
    "nucleus",
    "cytoplasm",
    "membrane",
    "mitochondrion",
    "extracellular space"
  ),
  Count = c(90, 110, 75, 65, 80),
  color = c(
    "#1f77b4",
    "#ff7f0e",
    "#2ca02c",
    "#d62728",
    "#9467bd"
  ),
  stringsAsFactors = FALSE
)

# 3. Molecular Function (MF)
```

```
filtered_data_MF <- data.frame(
  Description = c(
    "protein binding",
    "DNA binding",
    "enzyme activity",
    "transporter activity",
    "receptor activity"
  ),
  Count = c(140, 130, 100, 70, 90),
  color = c(
    "#1f77b4",
    "#ff7f0e",
    "#2ca02c",
    "#d62728",
    "#9467bd"
  ),
  stringsAsFactors = FALSE
)

# 4. Disease Ontology (DO)
filtered_data_DO <- data.frame(
  Description = c(
    "cancer",
    "cardiovascular disease",
    "neurological disorder",
    "metabolic disease",
    "infectious disease"
  ),
  Count = c(200, 150, 120, 90, 160),
  color = c(
    "#1f77b4",
    "#ff7f0e",
    "#2ca02c",
    "#d62728",
    "#9467bd"
  ),
  stringsAsFactors = FALSE
)

# 5. Reactome Pathways
filtered_data_Reactome <- data.frame(
  Description = c(
    "Cell Cycle",
    "Apoptosis",
    "DNA Repair",
    "Signal Transduction",
    "Metabolism of Proteins"
  ),
  Count = c(110, 95, 80, 130, 85),
  color = c(
    "#1f77b4",
    "#ff7f0e",
    "#2ca02c",
```

```
    "#d62728",
    "#9467bd"
  ),
  stringsAsFactors = FALSE
)

# 6. KEGG Pathways
filtered_data_kegg <- data.frame(
  Description = c(
    "PI3K-Akt signaling pathway",
    "MAPK signaling pathway",
    "NF-kappa B signaling pathway",
    "JAK-STAT signaling pathway",
    "Toll-like receptor signaling pathway"
  ),
  Count = c(175, 160, 145, 130, 155),
  color = c(
    "#1f77b4",
    "#ff7f0e",
    "#2ca02c",
    "#d62728",
    "#9467bd"
  ),
  stringsAsFactors = FALSE
)

# Combine all filtered data frames into a list
data_list <- list(
  BP = filtered_data_BP,
  CC = filtered_data_CC,
  MF = filtered_data_MF,
  DO = filtered_data_DO,
  Reactome = filtered_data_Reactome,
  KEGG = filtered_data_kegg
)

# Create the Circular Bar Chart
combined_and_visualized_data <- enrich_circo_bar(data_list)
```

`enrich_polar_bubble` *Enrichment Polar Bubble Plot*

Description

This function creates a polar bubble plot using 'ggplot2'. It is designed to visually represent data with methods and positional metrics integrated, highlighting specific IDs if necessary.

Usage

```
enrich_polar_bubble(final_combined_df_with_id_and_position, pal, highlight_ids)
```

Arguments

final_combined_df_with_id_and_position	A data frame containing 'id', 'Count', 'method', 'Description', 'point_position', 'test_color'.
pal	A named vector of colors corresponding to the 'method' values.
highlight_ids	A vector of IDs to highlight.

Value

A 'ggplot' object representing the enriched polar bubble plot.

Examples

```
final_df <- data.frame(id = 1:10, Count = c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100),
                       method = rep("Method1", 10),
                       Description = LETTERS[1:10],
                       point_position = seq(10, 100, 10),
                       test_color = sample(c("red", "blue"), 10, replace = TRUE))
pal <- c("Method1" = "blue")
highlight_ids <- c(1, 5, 9)
enrich_polar_bubble(final_df, pal, highlight_ids)
```

---

extract_descriptions_counts *Extract and Count Descriptions with Specified Color*

Description

This function filters a data frame for specified descriptions, selects the 'Description' and 'Count' columns, and adds a new column with a specified color.

Usage

```
extract_descriptions_counts(df, descriptions, color)
```

Arguments

df	A data frame containing at least 'Description' and 'Count' columns.
descriptions	A vector of descriptions to filter in the data frame.
color	A character string specifying the color to be added as a new column.

Value

A data frame filtered by descriptions, containing 'Description', 'Count', and a new 'color' column.

Examples

```
# Generate Sample Input Data for extract_descriptions_counts Function

# Create a sample data frame with 'Description' and 'Count' columns
data <- data.frame(
  Description = c(
    "immunoglobulin production",
    "B cell mediated immunity",
    "T cell activation",
    "antigen processing and presentation",
    "cytokine signaling",
    "natural killer cell activity",
    "phagocytosis",
    "complement activation",
    "antibody-dependent cellular cytotoxicity",
    "regulatory T cell function"
  ),
  Count = c(
    150,  # immunoglobulin production
    200,  # B cell mediated immunity
    175,  # T cell activation
    125,  # antigen processing and presentation
    190,  # cytokine signaling
    160,  # natural killer cell activity
    140,  # phagocytosis
    180,  # complement activation
    130,  # antibody-dependent cellular cytotoxicity
    170   # regulatory T cell function
  ),
  stringsAsFactors = FALSE  # Ensure that strings are not converted to factors
)



descriptions_to_filter <- c("immunoglobulin production", "B cell mediated immunity")
specified_color <- "red"  # You can specify any color you desire
filtered_data_with_color <- extract_descriptions_counts(
  data, descriptions_to_filter,
  specified_color)
print(filtered_data_with_color)
```

`extract_ntop_pathways` *Extract and Store Top Pathways for Each Sample*

Description

This function processes a dataframe containing SSGSEA KEGG results. It allows specifying the number of top pathways to extract for each sample based on their scores, and stores these in a new dataframe with sample names and pathway scores.

Usage

```
extract_ntop_pathways(ssgsea_kegg, nTop = 5)
```

Arguments

ssgsea_kegg	Dataframe containing SSGSEA KEGG results with samples as columns and pathways as rows.
nTop	Integer, number of top pathways to select for each sample.

Value

A dataframe with columns 'Pathway', 'Sample', and 'Value' representing the top pathways for each sample.

Examples

```
# Example: Generating input data for the extract_ntop_pathways function

# Define example pathways
pathways <- c("Pathway_A", "Pathway_B", "Pathway_C", "Pathway_D", "Pathway_E",
              "Pathway_F", "Pathway_G", "Pathway_H", "Pathway_I", "Pathway_J")

# Define example samples
samples <- c("Sample_1", "Sample_2", "Sample_3")

# Generate random SSGSEA KEGG scores between 0 and 1
set.seed(123)  # For reproducibility
ssgsea_scores <- matrix(runif(length(pathways) * length(samples), min = 0, max = 1),
                        nrow = length(pathways), ncol = length(samples),
                        dimnames = list(pathways, samples))

# Convert to a data frame
ssgsea_kegg <- as.data.frame(ssgsea_scores)

# Extract the top 3 pathways for each sample
top_pathways <- extract_ntop_pathways(ssgsea_kegg, nTop = 3)
```

---

extract_positive_pathways

Extract Positive Pathways from SSGSEA Results and Select Random Samples

---

Description

This function processes the results of SSGSEA, specifically focusing on KEGG pathways. It extracts pathways with positive values from each sample and randomly selects a subset of them.

Usage

```
extract_positive_pathways(ssgsea_kegg, max_paths_per_sample = 5)
```

Arguments

ssgsea_kegg	A matrix or data frame with pathways as rows and samples as columns.
max_paths_per_sample	Integer, maximum number of pathways to select per sample.

Value

A data frame with selected pathways, samples, and their corresponding values.

Examples

```
# Example: Generating input data for the extract_positive_pathways function

# Define example pathways
pathways <- c("Pathway_1", "Pathway_2", "Pathway_3", "Pathway_4", "Pathway_5",
              "Pathway_6", "Pathway_7", "Pathway_8", "Pathway_9", "Pathway_10")

# Define example samples
samples <- c("Sample_A", "Sample_B", "Sample_C")

# Generate random SSGSEA KEGG scores including both positive and negative values
set.seed(456)  # For reproducibility
ssgsea_scores <- matrix(rnorm(length(pathways) * length(samples), mean = 0, sd = 1),
                        nrow = length(pathways), ncol = length(samples),
                        dimnames = list(pathways, samples))

# Convert to a data frame
ssgsea_kegg <- as.data.frame(ssgsea_scores)

# Use the extract_positive_pathways function to extract up to 3 positive pathways per sample
selected_positive_pathways <- extract_positive_pathways(ssgsea_kegg, max_paths_per_sample = 3)
```

---

facet_density_foldchange

Create faceted high-density region plots with optional points and density contours

---

Description

This function creates faceted high-density region plots using ggdensity for adding optional density rug and density contours, and scatter points. It also adds a regression line and Pearson correlation label. The plot is faceted by a grouping variable.

Usage

```
facet_density_foldchange(
  data,
  x_var,
  y_var,
  group_var,
  facet_var,
  palette,
  show_points = FALSE,
  show_density = TRUE,
  point_size = 2.5,
  point_alpha = 0.1,
  line_size = 1.6,
  cor_method = "pearson",
  cor_label_pos = c("left", 0.97),
  cor_vjust = NULL
)
```

Arguments

data	Data frame containing variables for plotting.
x_var	Name of the x-axis variable as a string.
y_var	Name of the y-axis variable as a string.
group_var	Name of the grouping variable for color mapping as a string.
facet_var	Name of the faceting variable.
palette	Color palette for the plot as a character vector.
show_points	Logical, if TRUE adds scatter points to the plot.
show_density	Logical, if TRUE adds filled density contours to the plot.
point_size	Size of the points, relevant if show_points is TRUE.
point_alpha	Transparency level of the points, relevant if show_points is TRUE.
line_size	Size of the regression line.
cor_method	Method to calculate correlation ("pearson" or "spearman").
cor_label_pos	Vector of length 2 indicating the position of the correlation label (x and y).
cor_vjust	Vertical justification for correlation label, default is NULL.

Value

A ‘ggplot‘ object representing the high-density region plot.

Examples

```
combined_df_file <- system.file("extdata", "combined_df.rds", package = "TransProR")
combined_df <- readRDS(combined_df_file)
pal2 = c("#2787e0","#1a9ae0","#1dabbf","#00897b","#43a047","#7cb342")
all_facet_density_foldchange_name1 <- facet_density_foldchange(
```

```
  data = combined_df,
  x_var = "log2FoldChange_1",
  y_var = "log2FoldChange_2",
  group_var = "name",
  facet_var = "name",
  palette = pal2,
  show_points = TRUE,
  show_density = FALSE,
  point_size = 2,
  point_alpha = 0.1,
  line_size = 1.6,
  cor_method = "pearson",
  cor_label_pos = c("left", "top"),
  cor_vjust = 1
)
```

`filter_diff_genes` *Filter Differentially Expressed Genes*

Description

This function filters a data frame to identify genes with significant differential expression based on specified thresholds for p-values and log fold change. It allows for flexible input of column names for p-values and log fold change.

Usage

```
filter_diff_genes(
  data,
  p_val_col = "adj.P.Val",
  log_fc_col = "logFC",
  p_val_threshold = 0.05,
  log_fc_threshold = 1
)
```

Arguments

`data`	A data frame containing gene expression data.
`p_val_col`	Character string indicating the column name for p-values. Default is "adj.P.Val".
`log_fc_col`	Character string indicating the column name for log fold change. Default is "logFC".
`p_val_threshold`	Numeric threshold for filtering p-values. Default is 0.05.
`log_fc_threshold`	Numeric threshold for filtering log fold change. Default is 1.0.

Value

A data frame with genes filtered by the specified criteria.

Examples

```
# Create a sample data frame with p-values and log fold changes
sample_data <- data.frame(
  adj.P.Val = c(0.03, 0.06, 0.02, 0.07),
  logFC = c(1.5, 0.8, -1.2, 1.1),
  gene = c("Gene1", "Gene2", "Gene3", "Gene4")
)

# Use the filter_diff_genes function to filter significant genes
filtered_genes <- filter_diff_genes(sample_data)
print(filtered_genes)
```

---

four_degs_venn *Function to Create a Venn Diagram of DEGs*

Description

This function creates a Venn Diagram using the ggVennDiagram package. It allows customization of various aesthetic elements of the diagram.

Usage

```
four_degs_venn(degs_list)
```

Arguments

degs_list	A list of DEG sets for Venn Diagram creation.

Value

A ggplot object representing the Venn Diagram.

Examples

```
data("all_degs_venn", package = "TransProR")
four_degs_venn <- four_degs_venn(all_degs_venn)
```

gather_graph_edge	*Gather graph edge from data frame Please note that this function is from the 'ggraph' package and has not been altered in functionality, but it has been optimized and iterated. It is not original content of 'TransProR'. However, since 'ggraph' caused frequent GitHub Action errors during the creation of 'TransProR', the author directly referenced the involved functions in 'TransProR'. This is not the author's original creation. All users please be aware!*

Description

Gather graph edge from data frame Please note that this function is from the 'ggraph' package and has not been altered in functionality, but it has been optimized and iterated. It is not original content of 'TransProR'. However, since 'ggraph' caused frequent GitHub Action errors during the creation of 'TransProR', the author directly referenced the involved functions in 'TransProR'. This is not the author's original creation. All users please be aware!

Usage

```
gather_graph_edge(df, index = NULL, root = NULL)
```

Arguments

df	A data frame
index	A vector of column names to group by
root	Root name for the root node connections, optional

Value

A tibble of graph edges

Examples

```
# Example taxonomic hierarchy data frame
OTU <- tibble::tibble(
  p = c("Firmicutes", "Firmicutes", "Bacteroidetes", "Bacteroidetes", "Proteobacteria"),
  c = c("Bacilli", "Clostridia", "Bacteroidia", "Bacteroidia", "Gammaproteobacteria"),
 o = c("Lactobacillales", "Clostridiales", "Bacteroidales", "Bacteroidales", "Enterobacterales"),
  abundance = c(100, 150, 200, 50, 300) # Abundance or some other metric
)

# Gathering graph edges by specifying hierarchical taxonomic levels
edges <- gather_graph_edge(OTU, index = c("p", "c", "o"))

# Adding a root node to the graph
edges_with_root <- gather_graph_edge(OTU, index = c("p", "c", "o"), root = "Root")
```

gather_graph_node	*Gather graph nodes from a data frame Please note that this function is from the 'ggraph' package and has not been altered in functionality, but it has been optimized and iterated. It is not original content of 'TransProR'. However, since 'ggraph' caused frequent GitHub Action errors during the creation of 'TransProR', the author directly referenced the involved functions in 'TransProR'. This is not the author's original creation. All users please be aware!*

Description

Gather graph nodes from a data frame Please note that this function is from the 'ggraph' package and has not been altered in functionality, but it has been optimized and iterated. It is not original content of 'TransProR'. However, since 'ggraph' caused frequent GitHub Action errors during the creation of 'TransProR', the author directly referenced the involved functions in 'TransProR'. This is not the author's original creation. All users please be aware!

Usage

```
gather_graph_node(
  df,
  index = NULL,
  value = utils::tail(colnames(df), 1),
  root = NULL
)
```

Arguments

df	A data frame
index	A vector of column names to group by
value	Column name used for summarizing node size, defaults to the last column
root	Root name for the root node connections, optional

Value

a tibble of graph nodes

Examples

```
# Example taxonomic hierarchy data frame (OTU table)
OTU <- tibble::tibble(
  p = c("Firmicutes", "Firmicutes", "Bacteroidetes", "Bacteroidetes", "Proteobacteria"),
  c = c("Bacilli", "Clostridia", "Bacteroidia", "Bacteroidia", "Gammaproteobacteria"),
 o = c("Lactobacillales", "Clostridiales", "Bacteroidales", "Bacteroidales", "Enterobacterales"),
  abundance = c(100, 150, 200, 50, 300) # Numeric values for node size
)
```

```
# Gathering graph nodes by specifying hierarchical taxonomic levels
nodes <- gather_graph_node(OTU, index = c("p", "c", "o"))
```

---

gene_color *Merge Genes with Color Information Based on Up/Down Regulation*

Description

This function merges selected genes with differential expression data and adds a color column based on up/down regulation.

Usage

```
gene_color(selected_genes, DEG_deseq2, up_color, down_color)
```

Arguments

selected_genes	A data frame containing selected genes with a column named "Symble".
DEG_deseq2	A data frame containing differential expression data with a column named "Symble" and a column named "change" indicating up/down regulation.
up_color	The color to assign to genes with up-regulated expression.
down_color	The color to assign to genes with down-regulated expression.

Value

A data frame containing merged genes with an additional color column.

Examples

```
selected_genes_deseq2_file <- system.file("extdata",
                                          "selected_genes_deseq2.rds",
                                          package = "TransProR")
selected_genes_deseq2 <- readRDS(selected_genes_deseq2_file)
Diff_deseq2_file <- system.file("extdata", "Diff_deseq2.rds", package = "TransProR")
Diff_deseq2 <- readRDS(Diff_deseq2_file)

result_deseq2 <- gene_color(selected_genes_deseq2, Diff_deseq2, "#0000EE", "#fc4746")
```

gene_highlights *Add gene highlights to a ggtree object*

Description

This function enhances a 'ggtree' plot by adding highlights for specific genes. It adds both a semi-transparent fan-shaped highlight and a point at the node corresponding to each gene. Colors for each gene can be customized.

Usage

```
gene_highlights(ggtree_obj, genes_to_highlight, hilight_extend = 18)
```

Arguments

ggtree_obj	A ggtree object to which the highlights will be added.
genes_to_highlight	A data frame containing genes and their corresponding colors.
hilight_extend	Integer, the extension of the highlight fan in degrees.

Value

A ggtree object with added gene highlights.

Examples

```
data("gtree", package = "TransProR")

# Define genes and their colors
genes_df <- data.frame(Symble = c("t5", "t9"),
                       color = c("#FF0000", "#0000FF"))

# Add highlights
gtree <- gene_highlights(gtree, genes_to_highlight = genes_df)
```

gene_map_pathway *Create Pathway-Gene Mapping Data Frame*

Description

This function takes multiple data frames and pathway IDs, merging them into a new data frame. Each data frame represents a type of analysis (e.g., BP, KEGG, MF, etc.).

Usage

```
gene_map_pathway(
  BP_dataframe,
  BP_ids,
  KEGG_dataframe,
  KEGG_ids,
  MF_dataframe,
  MF_ids,
  REACTOME_dataframe,
  REACTOME_ids,
  CC_dataframe,
  CC_ids,
  DO_dataframe,
  DO_ids
)
```

Arguments

BP_dataframe	Data frame for Biological Process analysis
BP_ids	Selected pathway IDs for Biological Process analysis
KEGG_dataframe	Data frame for KEGG analysis
KEGG_ids	Selected pathway IDs for KEGG analysis
MF_dataframe	Data frame for Molecular Function analysis
MF_ids	Selected pathway IDs for Molecular Function analysis
REACTOME_dataframe	Data frame for REACTOME analysis
REACTOME_ids	Selected pathway IDs for REACTOME analysis
CC_dataframe	Data frame for Cellular Component analysis
CC_ids	Selected pathway IDs for Cellular Component analysis
DO_dataframe	Data frame for Disease Ontology analysis
DO_ids	Selected pathway IDs for Disease Ontology analysis

Value

A new data frame that includes pathways, gene, type, and value columns

Examples

```
# Simulating data for different analysis types

# Simulate Biological Process (BP) data frame
BP_df <- data.frame(
  ID = c("GO:0002376", "GO:0019724"),
  geneID = c("GENE1/GENE2", "GENE3/GENE4"),
  Description = c("Immune response", "Glycosylation process")
)
```

```
# Simulate KEGG data frame
KEGG_df <- data.frame(
  ID = c("12345", "67890"),
  geneID = c("GENE5/GENE6", "GENE7/GENE8"),
  Description = c("Pathway 1", "Pathway 2")
)

# Simulate Molecular Function (MF) data frame
MF_df <- data.frame(
  ID = c("ABC123", "DEF456"),
  geneID = c("GENE9/GENE10", "GENE11/GENE12"),
  Description = c("Molecular function A", "Molecular function B")
)

# Simulate REACTOME data frame
REACTOME_df <- data.frame(
  ID = c("R-HSA-12345", "R-HSA-67890"),
  geneID = c("GENE13/GENE14", "GENE15/GENE16"),
  Description = c("Pathway in Reactome 1", "Pathway in Reactome 2")
)

# Simulate Cellular Component (CC) data frame
CC_df <- data.frame(
  ID = c("GO:0005575", "GO:0005634"),
  geneID = c("GENE17/GENE18", "GENE19/GENE20"),
  Description = c("Cellular component A", "Cellular component B")
)

# Simulate Disease Ontology (DO) data frame
DO_df <- data.frame(
  ID = c("DOID:123", "DOID:456"),
  geneID = c("GENE21/GENE22", "GENE23/GENE24"),
  Description = c("Disease A", "Disease B")
)

# Example pathway IDs for each analysis
BP_ids <- c("GO:0002376", "GO:0019724")
KEGG_ids <- c("12345", "67890")
MF_ids <- c("ABC123", "DEF456")
REACTOME_ids <- c("R-HSA-12345", "R-HSA-67890")
CC_ids <- c("GO:0005575", "GO:0005634")
DO_ids <- c("DOID:123", "DOID:456")

# Generate the pathway-gene map using the gene_map_pathway function
pathway_gene_map <- gene_map_pathway(
  BP_dataframe = BP_df, BP_ids = BP_ids,
  KEGG_dataframe = KEGG_df, KEGG_ids = KEGG_ids,
  MF_dataframe = MF_df, MF_ids = MF_ids,
  REACTOME_dataframe = REACTOME_df, REACTOME_ids = REACTOME_ids,
  CC_dataframe = CC_df, CC_ids = CC_ids,
  DO_dataframe = DO_df, DO_ids = DO_ids
)
```

```
# Display the resulting pathway-gene mapping data frame
print(pathway_gene_map)
```

`get_gtex_exp` *Get GTEx Expression Data for Specific Organ*

Description

This function retrieves gene expression data from the GTEx project that is specific to a certain organ. It performs various checks and processing steps to ensure that the data is consistent and relevant to the specified organ. The filtered and cleaned data is saved as an RDS file for further analysis.

Usage

```
get_gtex_exp(
  organ_specific,
  file_path,
  probe_map_path,
  pheno_path,
  output_path
)
```

Arguments

`organ_specific`	A character string specifying the organ to filter the gene expression data by.
`file_path`	A character string specifying the path to the GTEx gene expression data file.
`probe_map_path`	A character string specifying the path to the gtex_probeMap_gencode data file.
`pheno_path`	A character string specifying the path to the GTEx phenotype data file.
`output_path`	A character string specifying the path where the output RDS file will be saved.

Details

The function begins by checking if the gene expression and phenotype data files exist at the specified paths. It then loads these data files and processes them by setting appropriate row names, modifying column names for clarity, and filtering samples based on the specified organ. The function ensures that only samples present in both datasets are retained for consistency. It also removes any duplicate gene entries to prevent redundancy. Finally, the processed data is saved as an RDS file.

Value

A data frame containing gene expression data for the specified organ. Rows represent genes, and columns represent samples. Note that this function also saves the organ-specific GTEx data as an RDS file at the specified output path.

Note

The function will stop and throw an error if the input files do not exist, or if no samples are found for the specified organ.

CRITICAL: The 'output_path' parameter must end with '.rds' to be properly recognized by the function. It is also highly recommended that the path includes specific identifiers related to the target samples. Please structure the 'output_path' following this pattern: './your_directory/your_sample_type.gtex.rds'.

Examples

```
counts_file <- system.file("extdata", "gtex_gene_expected_count_test", package = "TransProR")
probe_map_file <- system.file("extdata",
                              "gtex_probeMap_gencode.v23.annotation.gene.probemap_test",
                              package = "TransProR")
phenotype_file <- system.file("extdata", "GTEX_phenotype_test", package = "TransProR")
ouput_file <- file.path(tempdir(), "skcm_gtex.rds")

SKCM_gtex <- get_gtex_exp(
  organ_specific = "Skin",
  file_path = counts_file,
  probe_map_path = probe_map_file,
  pheno_path = phenotype_file,
  output_path = ouput_file
)
head(SKCM_gtex[1:3, 1:3])
```

---

get_tcga_exp *TCGA Expression Data Processing*

Description

This function processes expression data and phenotype information, separates tumor and normal samples, and saves the results into different files. It's specifically designed for data obtained from TCGA.

Usage

```
get_tcga_exp(
  counts_file_path,
  gene_probes_file_path,
  phenotype_file_path,
  output_file_path
)
```

Arguments

counts_file_path
: File path to the counts data (usually in the form of a large matrix with gene expression data).

gene_probes_file_path
: File path containing the gene probes data.

phenotype_file_path
: File path to the phenotype data, which includes various sample attributes.

output_file_path
: Path where the output files, distinguished between tumor and normal, will be saved.

Value

A list containing matrices for tumor and normal expression data.

Note

IMPORTANT: This function assumes that the input files follow a specific format and structure, typically found in TCGA data releases. Users should verify their data's compatibility. Additionally, the function does not perform error checking on the data's content, which users should handle through proper preprocessing.

CRITICAL: The 'output_file_path' parameter must end with '.rds' to be properly recognized by the function. It is also highly recommended that the path includes specific identifiers related to the target samples, as the function will create further subdivisions in the specified path for tumor or normal tissues. Please structure the 'output_file_path' following this pattern: './your_directory/your_sample_type.exp.rds'.

Author(s)

Dongyue Yu

Examples

```
counts_file <- system.file("extdata", "TCGA-SKCM.htseq_counts_test.tsv", package = "TransProR")
gene_probes_file <- system.file("extdata",
                                "TCGA_gencode.v22.annotation.gene.probeMap_test",
                                package = "TransProR")
phenotype_file <- system.file("extdata", "TCGA-SKCM.GDC_phenotype_test.tsv", package = "TransProR")
ouput_file <- file.path(tempdir(), "SKCM_Skin_TCGA_exp_test.rds")

SKCM_exp <- get_tcga_exp(
  counts_file_path = counts_file,
  gene_probes_file_path = gene_probes_file,
  phenotype_file_path = phenotype_file,
  output_file_path = ouput_file
)
head(SKCM_exp[["tumor_tcga_data"]])[1:5, 1:5]
head(SKCM_exp[["normal_tcga_data"]], n = 10) # Because there is only one column.
```

gtree *Phylogenetic Tree Object*

Description

A dataset containing a phylogenetic tree object created using the ‘ggtree‘ package. This tree represents the evolutionary relationships among a set of species or genes.

Usage

```
data(gtree)
```

Format

A ‘ggtree‘ object.

Source

The phylogenetic tree was constructed based on sequence alignment data obtained from [Data Source, e.g., NCBI database, specific study, etc.].

highlight_by_node *Highlight Nodes in a Phylogenetic Tree with Custom Fill Colors and Transparency*

Description

This function adds highlights to specific nodes in a phylogenetic tree represented by a ‘ggtree‘ object. Users can specify the nodes to highlight along with custom fill colors, transparency, and extension options.

Usage

```
highlight_by_node(
  ggtree_object,
  nodes,
  fill_colors,
  alpha_values,
  extend_values
)
```

Arguments

ggtree_object	A ‘ggtree‘ object representing the phylogenetic tree.
nodes	A character vector specifying the nodes to highlight.
fill_colors	A character vector specifying the fill colors for the highlighted nodes.
alpha_values	A numeric vector specifying the transparency values for the highlighted nodes (between 0 and 1).
extend_values	A logical vector specifying whether to extend the highlight to the whole clade below each node.

Value

A modified ‘ggtree‘ object with the specified nodes highlighted.

Examples

```
plot_file <- system.file("extdata", "tree_plot.rds", package = "TransProR")
p2_plot <- readRDS(plot_file)

# Please replace the following vectors with your specific values
nodes <- c(117, 129, 125, 127, 119,
           123, 139, 166, 124, 131, 217) # x-values of the nodes you want to highlight
fill_colors <- c("#CD6600", "#CD6600", "#CD6600",
                 "#CD6600", "#009933", "#009933",
                 "#009933", "#009933", "#9B30FF",
                 "#9B30FF", "#9B30FF") # Fill colors
alpha_values <- c(0.3, 0.3, 0.3, 0.3, 0.2, 0.3,
                  0.3, 0.3, 0.3, 0.3, 0.3) # Transparency values
extend_values <- c(25, 24, 24, 25, 25, 25,
                   24, 24, 25, 24, 24) # Values for the 'extend' parameter

p2 <- highlight_by_node(
  p2_plot,
  nodes,
  fill_colors,
  alpha_values,
  extend_values
)
```

---

highlight_genes *Add Highlights for Genes on a Phylogenetic Tree*

Description

This function adds highlights for specified genes on a phylogenetic tree object.

Usage

```
highlight_genes(ggtree_obj, genes_to_highlight, hilight_extend = 18)
```

Arguments

ggtree_obj	A ggtree object representing the phylogenetic tree.
genes_to_highlight	A data frame containing gene names and corresponding colors to highlight.
hilight_extend	Numeric value indicating the extension length for highlights.

Value

A 'ggtree' object with added highlights for specified genes.

Examples

```
plot_file <- system.file("extdata", "tree_plot.rds", package = "TransProR")
p2_plot <- readRDS(plot_file)

selected_genes_deseq2_file <- system.file("extdata",
                                          "selected_genes_deseq2.rds",
                                          package = "TransProR")
selected_genes_deseq2 <- readRDS(selected_genes_deseq2_file)

Diff_deseq2_file <- system.file("extdata", "Diff_deseq2.rds", package = "TransProR")
Diff_deseq2 <- readRDS(Diff_deseq2_file)

result_deseq2 <- gene_color(selected_genes_deseq2, Diff_deseq2, "#0000EE", "#fc4746")

add_gene_highlights_p3 <- highlight_genes(p2_plot, result_deseq2, hilight_extend = 26)
```

---

limma_analyze *Differential Gene Expression Analysis using limma and voom*

Description

This function performs differential gene expression analysis using the 'limma' package with voom normalization. It reads tumor and normal expression data, merges them, filters low-expressed genes, normalizes the data, performs limma analysis, and outputs the results along with information on gene expression changes.

Usage

```
limma_analyze(
  tumor_file,
  normal_file,
  output_file,
  logFC_threshold = 2.5,
  p_value_threshold = 0.01
)
```

Arguments

tumor_file	Path to the tumor data file (RDS format).
normal_file	Path to the normal data file (RDS format).
output_file	Path to save the output DEG data (RDS format).
logFC_threshold	Threshold for log fold change for marking up/down-regulated genes.
p_value_threshold	Threshold for p-value for filtering significant genes.

Value

A data frame of differential expression results.

References

limma:Linear Models for Microarray and RNA-Seq Data User's Guide. For more information, visit the page: https://www.bioconductor.org/packages/release/bioc/vignettes/limma/inst/doc/usersguide.pdf

Examples

```
# Define file paths for tumor and normal data from the data folder
tumor_file <- system.file("extdata",
                          "removebatch_SKCM_Skin_TCGA_exp_tumor_test.rds",
                          package = "TransProR")
normal_file <- system.file("extdata",
                           "removebatch_SKCM_Skin_Normal_TCGA_GTEX_count_test.rds",
                           package = "TransProR")
output_file <- file.path(tempdir(), "DEG_limma_voom.rds")

DEG_limma_voom <- limma_analyze(
  tumor_file = tumor_file,
  normal_file = normal_file,
  output_file = output_file,
  logFC_threshold = 2.5,
  p_value_threshold = 0.01
)

# View the top 5 rows of the result
head(DEG_limma_voom, 5)
```

---

log_transform *Log transformation decision and application on data*

Description

This function evaluates the need for a log transformation based on a set of criteria and applies a log2 transformation if necessary.

Usage

```
log_transform(data)
```

Arguments

data	A numeric matrix or data frame.

Value

The original data or the data transformed with log2.

Author(s)

Dongyue Yu

Examples

```
file_path <- system.file("extdata",
                         "all_count_exp_test.csv",
                         package = "TransProR")
your_data <- read.csv(file_path,
                  row.names = 1)  # Assuming first column is row names (e.g., gene names)

TransformedData <- log_transform(data = your_data)
```

---

merge_density_foldchange

Create high-density region plot with optional points, density rugs, and contours

---

Description

This function creates a high-density region plot using hdr methods to add density rug and filled contours. It also adds a regression line and Pearson correlation label. Points can be added to the plot optionally.

Usage

```
merge_density_foldchange(
  data,
  x_var,
  y_var,
  group_var,
  palette = c("#3949ab", "#1e88e5", "#039be5", "#00897b", "#43a047", "#7cb342"),
  show_points = FALSE,
  point_size = 2.5,
  point_alpha = 0.2,
```

```
  x_lim = c(0, 20),
  y_lim = c(0, 20),
  cor_method = "pearson",
  line_size = 1.6,
  cor_label_pos = c("left", 0.97)
)
```

Arguments

data	Data frame containing variables for plotting.
x_var	Name of the x-axis variable as a string.
y_var	Name of the y-axis variable as a string.
group_var	Name of the grouping variable for color mapping as a string.
palette	Color palette for the plot as a character vector.
show_points	Logical, if TRUE adds points to the plot.
point_size	Size of the points, relevant if show_points is TRUE.
point_alpha	Transparency level of the points, relevant if show_points is TRUE.
x_lim	Numeric vector of length 2, giving the x-axis limits.
y_lim	Numeric vector of length 2, giving the y-axis limits.
cor_method	Method to calculate correlation ("pearson" or "spearman").
line_size	Size of the smoothing line.
cor_label_pos	Vector of length 2 indicating the position of the correlation label (x and y).

Value

A ggplot object representing the high-density region plot.

Examples

```
combined_df_file <- system.file("extdata", "combined_df.rds", package = "TransProR")
combined_df <- readRDS(combined_df_file)
pal1 = c("#3949ab","#1e88e5","#039be5","#00897b","#43a047","#7cb342")

all_density_foldchange_name1 <- merge_density_foldchange(
  data = combined_df,
  x_var = "log2FoldChange_1",
  y_var = "log2FoldChange_2",
  group_var = "name",
  palette = pal1,
  show_points = FALSE,
  point_size = 2.5,
  point_alpha = 0.1,
  x_lim = c(0, 20),
  y_lim = c(0, 20),
  cor_method = "pearson",
  line_size = 1.6,
  cor_label_pos = c("left", "top")
```

```
)
```

`merge_gtex_tcga` *Merge gene expression data from GTEx and TCGA datasets*

Description

This function merges gene expression data obtained from the GTEx (Genotype-Tissue Expression) and TCGA (The Cancer Genome Atlas) datasets. It is assumed that both datasets are in '.rds' format and have genes as row names. The merged dataset is saved as an RDS file at the specified output path.

Usage

```
merge_gtex_tcga(
  gtex_data_path,
  tcga_exp_path,
  output_path = "./merged_gtex_tcga_data.rds"
)
```

Arguments

`gtex_data_path`	A string that specifies the file path to the GTEx data saved in RDS format.
`tcga_exp_path`	A string that specifies the file path to the TCGA expression data saved in RDS format. This should be a data.frame with rows as genes and columns as samples.
`output_path`	A string that specifies the path where the merged dataset should be saved. The file is saved in '.rds' format. The default path is "./merged_gtex_tcga_data.rds".

Details

It is assumed that both datasets are in '.rds' format and have genes as row names.

Value

A data frame where rows represent genes and columns represent samples. The data frame contains expression values from both GTEx and TCGA datasets. It saves the merged dataset to the path specified by 'output_path'.

Note

CRITICAL: The 'output_path' parameter must end with '.rds' to be properly recognized by the function. It is also highly recommended that the path includes specific identifiers related to the target samples. Please structure the 'output_path' following this pattern: './your_directory/merged.your_sample_type.gtex.tcga.data

Examples

```
tumor_file <- system.file("extdata",
                          "removebatch_SKCM_Skin_TCGA_exp_tumor_test.rds",
                          package = "TransProR")
Normal_file <- system.file("extdata",
                           "removebatch_SKCM_Skin_Normal_TCGA_GTEX_count_test.rds",
                           package = "TransProR")
ouput_file <- file.path(tempdir(), "all_data.rds")

all_data <- merge_gtex_tcga(gtex_data_path = tumor_file,
                            tcga_exp_path = Normal_file,
                            output_path = ouput_file)
```

`merge_id_position` *Merge Data Frames by Common Row Names with Additional Columns*

Description

This function merges a list of data frames based on common row names. It adds an 'id' column to track the row order and a 'point_position' column calculated based on the maximum 'Count' value across all data frames. It filters data frames to include only common rows, sorts rows by the length of the 'Description' in descending order, and then merges them by rows.

Usage

```
merge_id_position(df_list)
```

Arguments

`df_list`	A list of data frames, each with a 'Description' and 'Count' column and set row names.

Value

A single data frame merged from the list, with additional 'id' and 'point_position' columns.

Examples

```
df1 <- data.frame(Description = c("DataA", "DataB"), Count = c(10, 20), row.names = c("R1", "R2"))
df2 <- data.frame(Description = c("DataC", "DataD"), Count = c(30, 40), row.names = c("R1", "R3"))
df_list <- list(df1, df2)
combined_df_test <- merge_id_position(df_list)
```

merge_method_color *Merge Data Frames with Specific Method and Color Columns*

Description

This function takes a list of data frames, a method name, and a list of colors. It adds a 'method' column and a 'test_color' column to each data frame, then merges all data frames by rows. It ensures that the color list length matches the list of data frames.

Usage

```
merge_method_color(df_list, method_name, color_list)
```

Arguments

df_list	A list of data frames, each containing at least 'Description' and 'Count' columns.
method_name	A string representing the method name to be added to each data frame.
color_list	A list of colors corresponding to each data frame for the 'test_color' column.

Value

A single data frame merged from the list, with each originally provided data frame now having a 'method' and a 'test_color' column.

Examples

```
df1 <- data.frame(Description = c("A", "B"), Count = c(10, 20))
df2 <- data.frame(Description = c("C", "D"), Count = c(30, 40))
df_list <- list(df1, df2)
method_name <- "Method1"
color_list <- c("Red", "Blue")
combined_df_test <- merge_method_color(df_list, method_name, color_list)
```

new_ggraph *Generate a graphical representation of pathway gene maps*

Description

This function merges multiple gene-pathway related dataframes, processes them for graph creation, and visualizes the relationships in a dendrogram layout using the provided node and edge gathering functions from the 'ggraph' package.

Usage

```
new_ggraph(
  BP_dataframe,
  BP_ids,
  KEGG_dataframe,
  KEGG_ids,
  MF_dataframe,
  MF_ids,
  REACTOME_dataframe,
  REACTOME_ids,
  CC_dataframe,
  CC_ids,
  DO_dataframe,
  DO_ids
)
```

Arguments

BP_dataframe	Dataframe for Biological Process.
BP_ids	IDs for Biological Process.
KEGG_dataframe	Dataframe for KEGG pathways.
KEGG_ids	IDs for KEGG pathways.
MF_dataframe	Dataframe for Molecular Function.
MF_ids	IDs for Molecular Function.
REACTOME_dataframe	Dataframe for REACTOME pathways.
REACTOME_ids	IDs for REACTOME pathways.
CC_dataframe	Dataframe for Cellular Component.
CC_ids	IDs for Cellular Component.
DO_dataframe	Dataframe for Disease Ontology.
DO_ids	IDs for Disease Ontology.

Value

A 'ggraph' object representing the pathway gene map visualization.

Examples

```
# Example Biological Process (BP) DataFrame
BP_dataframe <- data.frame(
  ID = c("BP1", "BP2"),
  Description = c("Biological Process 1", "Biological Process 2"),
  geneID = c("GeneA/GeneB/GeneC", "GeneD/GeneE"),
  stringsAsFactors = FALSE
)
```

```
# Example KEGG DataFrame
KEGG_dataframe <- data.frame(
  ID = c("KEGG1", "KEGG2"),
  Description = c("KEGG Pathway 1", "KEGG Pathway 2"),
  geneID = c("GeneA/GeneD", "GeneB/GeneF"),
  stringsAsFactors = FALSE
)

# Example Molecular Function (MF) DataFrame
MF_dataframe <- data.frame(
  ID = c("MF1", "MF2"),
  Description = c("Molecular Function 1", "Molecular Function 2"),
  geneID = c("GeneC/GeneE", "GeneF/GeneG"),
  stringsAsFactors = FALSE
)

# Example Reactome DataFrame
REACTOME_dataframe <- data.frame(
  ID = c("REACTOME1", "REACTOME2"),
  Description = c("Reactome Pathway 1", "Reactome Pathway 2"),
  geneID = c("GeneA/GeneF", "GeneB/GeneG"),
  stringsAsFactors = FALSE
)

# Example Cellular Component (CC) DataFrame
CC_dataframe <- data.frame(
  ID = c("CC1", "CC2"),
  Description = c("Cellular Component 1", "Cellular Component 2"),
  geneID = c("GeneC/GeneD", "GeneE/GeneH"),
  stringsAsFactors = FALSE
)

# Example Disease Ontology (DO) DataFrame
DO_dataframe <- data.frame(
  ID = c("DO1", "DO2"),
  Description = c("Disease Ontology 1", "Disease Ontology 2"),
  geneID = c("GeneF/GeneH", "GeneA/GeneI"),
  stringsAsFactors = FALSE
)

# Example IDs vectors for each category
BP_ids <- c("BP1", "BP2")
KEGG_ids <- c("KEGG1", "KEGG2")
MF_ids <- c("MF1", "MF2")
REACTOME_ids <- c("REACTOME1", "REACTOME2")
CC_ids <- c("CC1", "CC2")
DO_ids <- c("DO1", "DO2")

# Running the `new_ggraph` function to plot a graph
plot <- new_ggraph(
  BP_dataframe = BP_dataframe, BP_ids = BP_ids,
  KEGG_dataframe = KEGG_dataframe, KEGG_ids = KEGG_ids,
  MF_dataframe = MF_dataframe, MF_ids = MF_ids,
```

```
  REACTOME_dataframe = REACTOME_dataframe, REACTOME_ids = REACTOME_ids,
  CC_dataframe = CC_dataframe, CC_ids = CC_ids,
  DO_dataframe = DO_dataframe, DO_ids = DO_ids
)
```

`pathway_count` *Count Genes Present in Pathways Above a Threshold*

Description

This function filters pathways that meet a count threshold and then counts the presence of specified genes in those pathways.

Usage

```
pathway_count(GO, count_threshold, enrich_data)
```

Arguments

`GO`	A character vector of gene symbols.
`count_threshold`	An integer specifying the count threshold for selecting pathways.
`enrich_data`	A data frame containing pathway enrichment analysis results.

Value

A data frame with columns "Symble" (gene symbol), "Description" (pathway description), and "Exists" (1 if gene is present, 0 otherwise).

Examples

```
# Simulated gene list
GO <- c("Gene1", "Gene2", "Gene3", "Gene4", "Gene5")
# Simulated enrichment analysis data
enrich_data <- data.frame(
  ID = c("GO:0001", "GO:0002", "GO:0003"),
  Description = c("Pathway A", "Pathway B", "Pathway C"),
  Count = c(10, 4, 6),
  geneID = c("Gene1/Gene2/Gene3", "Gene4/Gene5", "Gene2/Gene6/Gene7")
)

# Example usage
count_threshold <- 5
result_df <- pathway_count(GO, count_threshold, enrich_data)
```

pathway_description *Describe Genes Present in Selected Pathways*

Description

This function identifies genes present in selected pathways based on gene enrichment analysis results.

Usage

```
pathway_description(GO, selected_pathways_names, enrich_data)
```

Arguments

GO	A character vector of gene symbols.
selected_pathways_names	A character vector specifying the names of selected pathways.
enrich_data	A data frame containing pathway enrichment analysis results.

Value

A data frame with columns "Symble" (gene symbol), "Description" (pathway description), and "Exists" (1 if gene is present, 0 otherwise).

Examples

```
GO <- c("Gene1", "Gene2", "Gene3", "Gene4", "Gene5")
# Simulated enrichment analysis data
enrich_data <- data.frame(
  ID = c("Pathway1", "Pathway2", "Pathway3", "Pathway4"),
  Description = c("Apoptosis", "Cell Cycle", "Signal Transduction", "Metabolism"),
  geneID = c("Gene1/Gene3", "Gene2/Gene4", "Gene1/Gene2/Gene3", "Gene5"),
  Count = c(2, 2, 3, 1),
  stringsAsFactors = FALSE
)

# Example usage
result <- pathway_description(GO,
                              selected_pathways_names="Apoptosis",
                              enrich_data)
```

prep_deseq2 *Prepare DESeq2 data for plotting*

Description

This function reads a DESeq2 DEG data frame from an RDS file, filters it, adjusts the log2FoldChange to absolute values, adds a pseudo-count to pvalues, and transforms pvalues for plotting. The final data frame is returned and optionally saved to a new RDS file.

Usage

```
prep_deseq2(input_path, output_name = NULL)
```

Arguments

input_path	Path to the RDS file containing the DESeq2 DEG data frame.
output_name	Name for the processed data frame, also used as the RDS file name.

Value

A data frame with processed DESeq2 DEG data.

Examples

```
deseq2_file <- system.file("extdata",
                           "DEG_deseq2_test.rds",
                           package = "TransProR")
deseq2 <- prep_deseq2(deseq2_file)
```

prep_edgeR *Prepare edgeR DEG data for plotting*

Description

This function reads an edgeR DEG data frame from an RDS file, filters it using deg_filter function, adjusts the logFC to absolute values, adds a pseudo-count to PValue, and transforms PValue for plotting. The final data frame is returned and optionally saved to a new RDS file.

Usage

```
prep_edgeR(input_path, output_name = NULL)
```

Arguments

input_path	Path to the RDS file containing the edgeR DEG data frame.
output_name	Name for the processed data frame, also used as the RDS file name.

Value

A data frame with processed edgeR DEG data.

Examples

```
edgeR_file <- system.file("extdata",
                          "DEG_edgeR_test.rds",
                          package = "TransProR")
edgeR <- prep_edgeR(edgeR_file)
```

`prep_limma` *Prepare limma-voom DEG data for plotting*

Description

This function reads a limma-voom DEG data frame from an RDS file, filters it using `deg_filter` function, adjusts the logFC to absolute values, adds a pseudo-count to P.Value, and transforms P.Value for plotting. The final data frame is returned and optionally saved to a new RDS file.

Usage

```
prep_limma(input_path, output_name = NULL)
```

Arguments

`input_path`	Path to the RDS file containing the limma-voom DEG data frame.
`output_name`	Name for the processed data frame, also used as the RDS file name.

Value

A data frame with processed limma-voom DEG data.

Examples

```
limma_file <- system.file("extdata",
                          "DEG_limma_voom_test.rds",
                          package = "TransProR")
limma <- prep_limma(limma_file)
```

prep_wilcoxon *Prepare Wilcoxon DEG data for plotting*

Description

This function reads a Wilcoxon DEG data frame from an RDS file, filters it using deg_filter function, adjusts the log2foldChange to absolute values, adds a pseudo-count to pValues, and transforms pValues for plotting. The final data frame is returned and optionally saved to a new RDS file.

Usage

```
prep_wilcoxon(input_path, output_name = NULL)
```

Arguments

input_path	Path to the RDS file containing the Wilcoxon DEG data frame.
output_name	Optional; name for the processed data frame, also used as the RDS file name. If not provided, the data frame will not be saved to file.

Value

A data frame with processed Wilcoxon DEG data.

Examples

```
wilcoxon_file <- system.file("extdata",
                             "Wilcoxon_rank_sum_testoutRst_test.rds",
                             package = "TransProR")
Wilcoxon <- prep_wilcoxon(wilcoxon_file)
```

process_heatdata *Process Heatmap Data with Various Selection Options*

Description

This function processes heatmap data ('heatdata') based on a given selection option. It allows customization of column names, selection of specific columns per group, or averaging columns based on a common prefix.

Usage

```
process_heatdata(
  heatdata,
  selection = 1,
  custom_names = NULL,
  num_names_per_group = NULL,
  prefix_length = 4
)
```

Arguments

heatdata	A data frame containing the heatmap data.
selection	An integer specifying the processing method: - 1: Use custom names for columns. - 2: Select a given number of columns per group based on a prefix. - 3: Calculate the average of columns per group based on a prefix.
custom_names	A character vector of custom names for columns (used when 'selection = 1'). The length of this vector must match the number of columns in 'heatdata'.
num_names_per_group	An integer specifying the number of columns to select per group (used when 'selection = 2').
prefix_length	An integer specifying the length of the prefix for grouping columns (used when 'selection = 2' or 'selection = 3'). Default is 4.

Value

A processed data frame based on the specified selection option.

Examples

```
# Example heatmap data frame
heatdata <- data.frame(
  groupA_1 = c(1, 2, 3),
  groupA_2 = c(4, 5, 6),
  groupB_1 = c(7, 8, 9),
  groupB_2 = c(10, 11, 12)
)

# Selection 1: Use custom names for columns
custom_names <- c("Sample1", "Sample2", "Sample3", "Sample4")
processed_data1 <- process_heatdata(heatdata, selection = 1, custom_names = custom_names)

# Selection 2: Select a given number of columns per group based on a prefix
processed_data2 <- process_heatdata(heatdata, selection = 2, num_names_per_group = 1)

# Selection 3: Calculate the average of columns per group based on a prefix
processed_data3 <- process_heatdata(heatdata, selection = 3, prefix_length = 6)
```

seek_gtex_organ *Load and Process GTEX Phenotype Data to Retrieve Primary Site Counts*

Description

This function reads the GTEX phenotype data from a specified path, renames its columns for better readability, and then returns a table of primary site counts.

Usage

```
seek_gtex_organ(path = "./download_data/GTEX_phenotype")
```

Arguments

path	The path to the GTEX phenotype data file. Default is "./download_data/GTEX_phenotype".

Value

A table representing the count of samples per primary site.

Examples

```
# Get the file path to the example data in the package
path <- system.file("extdata", "GTEX_phenotype_test", package = "TransProR")
# Call the `seek_gtex_organ` function with the path and print the result
SeekGtexOrgan <- seek_gtex_organ(path = path)
```

selectPathways *Randomly Select Pathways with Limited Word Count*

Description

This function randomly selects a specified number of pathways from a given list, ensuring that each selected pathway name does not exceed a specified number of words. It filters out pathways with names longer than the specified word limit before making the selection.

Usage

```
selectPathways(pathways, max_words = 10, num_select = 10)
```

Arguments

pathways	Character vector of pathways.
max_words	Integer, maximum number of words allowed in the pathway name.
num_select	Integer, number of pathways to randomly select.

Value

A character vector of selected pathways.

Examples

```
pathway_list <- c("pathway_one response to stimulus",
                  "pathway_two cell growth and death",
                  "pathway_three regulation of cellular process",
                  "pathway_four metabolic process")
selected_pathways <- selectPathways(pathway_list, max_words = 5, num_select = 2)
```

---

spiral_newrle *Render a Spiral Plot Using Run-Length Encoding*

Description

This function creates a spiral plot for visualizing sequential data in a compact and visually appealing way. It uses run-length encoding to represent the lengths and colors of sequences in the spiral.

Usage

```
spiral_newrle(x, samples, values, colors, labels = FALSE)
```

Arguments

x	A vector representing categories or segments.
samples	A vector indicating the sample each segment belongs to.
values	Numeric vector indicating the lengths of each segment.
colors	Character vector specifying the colors for each segment.
labels	Logical, whether to add labels to each segment.

Value

No return value, called for side effects. This function generates a spiral plot and optionally adds labels.

Examples

```
# Example: Creating a spiral plot using the spiral_newrle function

# Define example data
x <- c("A", "A", "B", "C")
samples <- c("Sample1", "Sample1", "Sample2", "Sample2")
values <- c(20, 30, 15, 35)
colors <- c("red", "blue", "green", "purple")
labels <- TRUE
```

```
# Initialize the spiral plot, setting the x-axis range and scaling
spiralize::spiral_initialize(xlim = c(0, sum(values)), scale_by = "curve_length",
                 vp_param = list(x = grid::unit(0, "npc"), just = "left"))

# Create a track for the spiral plot
spiralize::spiral_track(height = 0.5)

# Add segments to the spiral plot using run-length encoding
spiral_newrle(x, samples, values, colors, labels)
```

`Wilcoxon_analyze` *Differential Gene Expression Analysis Using Wilcoxon Rank-Sum Test*

Description

This function performs differential gene expression analysis using Wilcoxon rank-sum tests. It reads tumor and normal expression data, performs TMM normalization using 'edgeR', and uses Wilcoxon rank-sum tests to identify differentially expressed genes.

Usage

```
Wilcoxon_analyze(
  tumor_file,
  normal_file,
  output_file,
  logFC_threshold = 2.5,
  fdr_threshold = 0.05
)
```

Arguments

`tumor_file`	Path to the tumor data file (RDS format).
`normal_file`	Path to the normal data file (RDS format).
`output_file`	Path to save the output DEG data (RDS format).
`logFC_threshold`	Threshold for log fold change for marking up/down-regulated genes.
`fdr_threshold`	Threshold for FDR for filtering significant genes.

Value

A data frame of differential expression results.

References

Li, Y., Ge, X., Peng, F., Li, W., & Li, J. J. (2022). Exaggerated False Positives by Popular Differential Expression Methods When Analyzing Human Population Samples. Genome Biology, 23(1), 79. DOI: https://doi.org/10.1186/s13059-022-02648-4.

Examples

```
# Define file paths for tumor and normal data from the data folder
tumor_file <- system.file("extdata",
                          "removebatch_SKCM_Skin_TCGA_exp_tumor_test.rds",
                          package = "TransProR")
normal_file <- system.file("extdata",
                           "removebatch_SKCM_Skin_Normal_TCGA_GTEX_count_test.rds",
                           package = "TransProR")
output_file <- file.path(tempdir(), "Wilcoxon_rank_sum_testoutRst.rds")

# Run the Wilcoxon rank sum test
outRst <- Wilcoxon_analyze(
  tumor_file = tumor_file,
  normal_file = normal_file,
  output_file = output_file,
  logFC_threshold = 2.5,
  fdr_threshold = 0.01
)

# View the top 5 rows of the result
head(outRst, 5)
```

Index