Package 'DEVis'

September 27, 2018

Type Package

Title A Differential Expression Analysis Toolkit for Visual Analytics and Data Aggregation

Version 0.99.0

Author Adam Price

Maintainer Adam Price <ap3637@cumc.columbia.edu>

Description Differential expression analysis tools for data aggregation, visualization, exploratory analysis, and project organization.

License LGPL

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

Suggests knitr, rmarkdown, kableExtra

VignetteBuilder knitr

Depends DESeq2

Imports ggplot2, pheatmap, RColorBrewer, ggthemes, MASS, plyr, ggsci, SummarizedExperiment, methods, ggdendro, PoiClaClu, reshape2, grDevices, stats, utils, gridExtra

RemoteType local

RemoteUrl /Users/adam/GitHub/DEvis/DEVis

RemoteSha 0.99.0

R topics documented:

`create_dir_struct` *Initialize the directory structure for automatically storing and structuring results.*

Description

This function initializes and, if necessary, creates the directory structure for output of analysis based on a provided base directory. A /results/ folder will be created in the base directory containing a complete directory structure for all possible results that can be generated by this package. All visualizations and data aggregation tools rely on this directory structure for data output. Note that this function will NOT overwrite existing directories. Initialization of data directories should be performed seperately using init_data_paths().

Usage

```
create_dir_struct(base_dir)
```

Arguments

`base_dir`	Base directory in which to build directory structure for result output.

Value

This function does not return a value.

See Also

init_data_paths, set_output_mode

Examples

```
## Not run:

#Initialize a DEVis directory structure at a base directory.
create_dir_struct(base_dir="/Users/adam/projects/ebola/")

This command will create the following folders:
/Users/adam/projects/ebola/
  /results/
    /DE/
      /boxplot/
      /counts/
      /data/
      /density_plots/
      /divergence/
      /heatmaps/
      /profile_plots/
      /series_plots/
      /volcano/
   /dendrograms/
   /geneplots/
   /group_stats/
   /MDS/
     /standard/
     /hulls/
   /sample_distance/
     /euclidian/
     /poisson/


## End(Not run)
```

`create_master_res` *Create a data set consisting of aggregated data for multiple contrasts.*

Description

This function creates a master result set for the provided DE result sets. This function finds the union of DE gene names and extracts those rows fram all DE result sets, then merges the sets into a single master DE file containing both log2foldchange and padj values. The results are written to the DE output directory and returned by the function. Relies on init_cutoffs() significant thresholds.

Usage

```
create_master_res(res_list, filename = "master_DE.txt",
  method = "union", lfc_filter = FALSE)
```

Arguments

res_list	A list of DESeq result sets created with DESeq2::results(). I.E: list(res1, res2, ..., resN).
filename	Destination output filename.
method	Method for aggregating data. "union" will gather expression data for all samples in results list when a gene is differentially expressed in at least one sample. "intersection" will gather expression data only for genes that are DE in all samples of the result list.
lfc_filter	Impose a filter on the DE set by a minimum absolute log2foldChange as determined by init_cutoffs(). Default=FALSE.

Value

This function returns a data frame containing union-based aggregation of all provided result sets. Results of this function are also written to file in the /DE/data/ directory in tab-delimited format.

See Also

init_cutoffs, create_dir_struct

Examples

```
## Not run:

#Prepare a result list for aggregation.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

#Generate the master DE data frame using union-based aggregation
master_df <- create_master_res(res_list=myResList, filename="master_DE.txt",
                               method="union")

#Generate the master DE data frame using intersection-based aggregation.
#Filter genes below minimum log fold-change.
master_df <- create_master_res(res_list=myResList, filename="master_DE.txt",
                               method="union", lfc_filter=TRUE)


## End(Not run)
```

create_relabel_field *Create a new metadata field by renaming existing levels of an existing field.*

Description

This function, given a target metadata field, will create a new column based on that field that has different labels. This is particularly useful when cleaning figures for publication quality, as often labels will contain abbreviations or delimiting characters such as "_". This function allows for a new column to be generated with more human-friendly labels that can be fed into visualizations instead of the defaults. Requires that target data has been prepared with prep_targets().

Usage

```
create_relabel_field(target_column, new_column_name)
```

Arguments

target_column Column name in targets data to be included in the new column. String.

new_column_name Name for the new column that will be created. String.

Value

This function returns target data containing the newly created label field.

See Also

prep_targets

Examples

```
## Not run:

myCounts <- prep_counts(count_input="master_count_data.txt", delim="t")
myTargets <- prep_targets(target_input="master_count_data.txt", delim="t")

#Create a  field based on the "treatment_time" fields with new labels.
myTargets <- create_relabel_field(target_column="treatment_time",
                                  new_column_name="treatment_time_relabel")


## End(Not run)
```

DESeqResMeta-class *An S4 class extending the DESeqResults object containing additional fields for differential expression data.*

Description

This class is used to separate and identify relevant pieces of data by various visualizations in this package.

Slots

numDE The total number of differentially expressed genes in the result set.

numUp The number of upregulated differentially expressed genes in the result set.

numDown The number of downregulated differentially expressed genes in the result set.

allNames The names of all differentially expressed genes in the result set.

upNames The names of all upregulated differentially expressed genes in the result set.

downNames The names of all downregulated differentially expressed genes in the result set.

case The "case" condition that was used to make the contrast for this result set.

control The "control" condition that was used to make the contrast for this result set.

contrast The contrast that was made for this result set. I.E. case_vs_control

design_field The field in the provided target data that was used for experimental design in differential expression. I.E. Condition_Time.

de_boxplot *Visualize differentially expressed genes as a function of experimental design.*

Description

This function plots groupwise expression for all differentially expressed genes with regard to the experimental design. For example, for a differential expression design of ~Condition_Time, boxplots for differentially expressed genes for each condition_time case in the targets metadata will be produced.

Usage

```
de_boxplot(res_list, filename = "de_boxplot.pdf", theme = 1,
  returnData = FALSE)
```

Arguments

res_list	A list of DESeq result sets. Results can be calculated individually using DESeq's results() function. Lists of results can be created by creating a list(result1, result2 ... result_N).
filename	Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /DE/boxplot/ directory.
theme	Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.
returnData	Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return the long-form table of expression containing catagorical grouping, and sample IDs, and log2 fold-change.

Examples

```
## Not run:

#Prepare a result list for aggregation.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

#Plot differentially expressed genes for the aggregate result sets.
de_boxplot(res_list=myResList, filename="DE_condition_time_boxplot.pdf",
           theme=2, returnData=FALSE)
```

```
## End(Not run)
```

de_counts *Visualize differentially expressed gene counts as a stacked barplot.*

Description

This function generates a stacked barplot representing differentially expressed gene counts for a result sets, distinguishing up and down regulated genes across one or more result sets. This function will produce a stacked barplot and an output file containing count information. Count data for the provided result sets will also be returned by this function.

Usage

```
de_counts(res_list, filename = "de_count_barplot.pdf",
  customLabels = FALSE, theme = 1, returnData = FALSE)
```

Arguments

res_list	A list of DESeq result sets created with DESeq2::results(). I.E: list(res1, res2, ..., resN).
filename	Output file destination. String. Valid extensions are .pdf and .png. The data file accompanying this plot file will have the same name, but will be output as a tab-delimited text file. Output will be written to the /DE/counts/ directory. Alternatively, file generation can be turned off using set_output_mode("screen").
customLabels	If customLabels is set to TRUE, the user will be prompted to provide a custom label for each label on the x-axis.
theme	The color and design scheme for the output plot. Valid selections are integers between 1-6.
returnData	Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

A data frame containing count information for both up and down regulated gene counts for each result set provided in res_list. Output files will be written to /DE/counts/.

See Also

create_dir_struct, set_output_mode

Examples

```
## Not run:

#Prepare a result list for aggregation.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
```

```
myResList <- list(res.day1, res.day2, res.day3)

#Visualize count data for the result set and save the results.
de_counts(res_set=myResList, filename="DE_counts.png", theme=2)


## End(Not run)
```

de_density_plot *Vizualize density plots of fold-change or significance values for aggregated data sets.*

Description

This function plots log2 fold-change or adjusted p-values for all differentially expressed genes for each contrast in a result set. Data aggregation across a series of result sets entails that not every gene in an aggregated data set will necessarily be differentially expressed, depending on the aggregation method. For instance, a gene that is differentially expressed in a day1 contrast that is not significant at day2 will be included in a union based aggregation of genes for day1 and day2. Visualizing the density of fold-change or p-values can reveal to what extent aggregation was consistent across conditions and can inform decision making in data aggregation.

Usage

```
de_density_plot(res_list, filename = "de_density_plot.pdf",
  type = "pval", method = "union", returnData = FALSE)
```

Arguments

`res_list` A list of DESeq result sets. Results can be calculated individually using DESeq's results() function. Lists of results can be created by creating a list(result1, result2 ... result_N).

`filename` Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /DE/density_plots/ directory.

`type` The type of data to be displayed in this plot. Valid selections are "lfc" (log2foldChange) and "pval".

`method` The method for computing overlaying results. Valid selections are "union" or "intersection". Union merges data for all result sets for genes that are differentially expressed in at least 1 result set. Intersection merges data for genes that only are differentially expressed in all result sets provided.

`returnData` Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return a data frame for aggregated differentially expressed genes containing gene names and log2 fold-change or adjusted p-values relative to the experimental control condition.

Examples

```
## Not run:

 #This example assumes an experimenal design of ~Condition_Time.

#Prepare a result list.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

/*
 * Aggregate data for all contrasts in the result list using union aggregation.
 * Display the density plot of p-values for the aggregated data.
 */
de_density_plot(res_list=myResList, filename="DE_density_union_pval.pdf",
                 type="pval", method="union", returnData=FALSE)

/*
 * Aggregate data for all contrasts in the result list using intersection aggregation.
 * Display the density plot of log fold-change values for the aggregated data.
 * Store the aggregate data as DE_lfc_intersect_df.
 */
DE_lfc_intersect_df <- de_density_plot(res_list=myResList,
                                        filename="DE_density_union_pval.pdf",
                                        type="lfc", method="intersection",
                                        returnData=TRUE)


## End(Not run)
```

de_diverge_plot *Visualize fold-change divergence for differentially expressed genes.*

Description

This function plots the log2 fold-change values for all differentially expressed genes for each contrast in a result set. This plot visualizes the distribution and strength of expression changes for all differentially expressed genes.

Usage

```
de_diverge_plot(res_list, filename = "de_divergence_plot.pdf",
  theme = 1, customLabels = FALSE, returnData = FALSE)
```

Arguments

`res_list`	A list of DESeq result sets. Results can be calculated individually using DESeq's results() function. Lists of results can be created by creating a list(result1, result2 ... result_N).
`filename`	Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /DE/divergence/ directory.

`theme` Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.

`customLabels` If customLabels is set to TRUE, the user will be prompted to provide a custom label for each label.

`returnData` Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return the long-form table for differentially expressed genes containing gene names, catagorical variable, and expression values.

Examples

```
## Not run:

#Prepare a result list.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

#Create the plot.
de_diverge_plot(res_list=myResList, filename="DE_divergence_plot.pdf",
                theme=1, returnData=FALSE)


## End(Not run)
```

de_filter *Apply a custom fold-change filter to an aggregated data frame.*

Description

This function allows you to filter an aggregated master result to only contain genes with fold-changes less than or greater than a specified threshold for a given metric. Several metrics are provided for flexibility of selection. The use of metrics in this function makes it possible to ask questions such as, "which genes have a mean fold-change of at least 2?", "which genes have a fold change of less than 5?", or "which genes have fold-change with a variance across all conditions of at least 10?"

Usage

```
de_filter(master_df, metric, threshold, operator, absolute = TRUE)
```

Arguments

`master_df` An aggregated master DE dataframe as produced using create_master_res().

`metric` Metric to be used for filtering. All metrics are gene-wise calculated. That is, for each gene in the aggregate data set, a metric is calculated that is then used to filter the data. Valid metrics are: "max", "min", "mean", "variance", "sd". The "max" metric is the gene-wise maximum fold-change value. The "min" metric is the gene-wise minimum fold-change value. The "mean" metric is the gene-wise fold-change mean. The "variance" metric is the gene-wise variance. The "sd" metric is the gene-wise standard deviation.

threshold	The cutoff value to use for filtering. Numeric.
operator	Operator to apply to the filtering process. Options are: "less", "greater".
absolute	Use absolute values when applying filters to filter from both negative and positive ends of the fold change distribution. Boolean. Default=TRUE.

Value

This function returns a data frame that has been filtered by the specified criteria.

See Also

create_master_res

Examples

```
## Not run:

#Prepare a result list.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

#Prepare an aggregate master data frame.
aggregate_df <- create_master_res(res_list=myResList, filename="master_DE.txt")

/*
 * Filter the aggregate data to contain only genes whose gene-wise maximum
 * fold-change is greater than 10. This will return genes whose gene-wise maximum
 * is > 10.
 */
de_filtered <- de_filter(master_df=aggregate_df, metric="max", threshold=10,
                          operator="greater", absolute=FALSE)


/*
 * Filter the aggregate data to contain only genes whose gene-wise minimum
 * fold-change is greater than an absoulte value of 4.  This will return genes
 * where the gene-wise minimum is < -4 and > +4.
 */
de_filtered <- de_filter(master_df=aggregate_df, metric="min", threshold=4,
                          operator="greater", absolute=TRUE)


/*
 * Filter the aggregate data to contain only genes whose gene-wise standard
 * deviation is less than an absoulte value of 2.  This will return genes
 * where the gene-wise standard deviation is > -2 and < +2.
 */
de_filtered <- de_filter(master_df=aggregate_df, metric="sd", threshold=2,
                          operator="less", absolute=TRUE)
```

```
## End(Not run)
```

de_heat *Create heat maps of differentially expressed genes.*

Description

This function generates heat maps for differentially expressed gene result sets based on the experimental design. Sorting and filtration can be applied to visualize genes based on specific criteria, and additional annotation can be included in the heat map based on target metadata columns. For example, aggregated results from several contrasts can have their differentially expressed genes sorted based on a specific metric, and the top N genes can be plotted based on that sorting method.

Usage

```
de_heat(res_list, filename = "heatmap.pdf", anno_columns,
  sort_choice = "none", specific_genes = "", numGenes = 30,
  theme = 2, cluster_genes = TRUE, cluster_contrasts = TRUE,
  customLabels = FALSE, returnData = FALSE)
```

Arguments

`res_list` A list of DESeq result sets. Results can be calculated individually using DESeq's results() function. Lists of results can be created by creating a list(result1, result2 ... result_N).

`filename` Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /DE/heatmaps/ directory.

`anno_columns` Annotation column names as a single string or a concatenated list of strings. Must correspond to columns in the targets metadata file. Multiple columns are allowed. I.E. c("Condition", "Timepoint")

`sort_choice` Sorting method for DE genes. Possible options are: "mean", "max", "min", "variance", "max_mean","min_mean", "sd". "max" sorts genes based on the highest expression level of any single gene in a result set. In contrast, "max_mean" first calculates mean expression across all result sets and subsequently sorts by maximum values. Min and min_mean function similarly. Variance and sd sort genes by highest gene-wise variance or standard deviation.

`specific_genes` A character vector of gene names can be passed to this parameter to plot the genes specified. This overrides sorting and numGene parameters.

`numGenes` Number of genes to include in the plot. Default: 30

`theme` Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.

`cluster_genes` Boolean. Cluster genes (rows) of the heat map. Default: TRUE

`cluster_contrasts` Boolean. Cluster contrasts (columns) of the heat map. Default: TRUE

`customLabels` If customLabels is set to TRUE, the user will be prompted to provide a custom label for each label on the x-axis.

`returnData` Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return a data frame containing expression data for the genes visualized in the heat map based on the sorting method and numGenes parameters.

Examples

```
## Not run:

#Prepare a result list for aggregation.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

/*
 * Create a heat map of the top 25 most upregulated genes based on time and condition.
 * Gene-wise max value based calculation.
 */
de_heat(res_list=myResList, anno_columns=c("Time", "Condition"), sort_choice="max",
         numGenes=25, theme=2, returnData=FALSE)


/*
 * Create a heat map of the top 50 most downregulated genes based on time and condition.
 * Mean based value calculation.
 */
de_heat(res_list=myResList, anno_columns=c("Time", "Condition"), sort_choice="min_mean",
         numGenes=50, theme=2, returnData=FALSE)


/*
 * Create a heat map of the top 100 most highly varying genes based on time and response.
 * Variance based value calculation.
 */
de_heat(res_list=myResList, anno_columns=c("Time", "Response"), sort_choice="variance",
         numGenes=100, theme=2, returnData=FALSE)

/*
 * Plot 3 specific genes, dont cluster by contrast.
 */
de_heat(res_list=myResList, anno_columns=c("Time", "Response"), sort_choice="variance",
        specific_genes=c("GEN1", "ABC2", "FuSG2"), theme=2, cluster_contrasts=FALSE)


## End(Not run)
```

---

de_profile_plot *Visualize gene-wise expression of differentially expressed genes.*

Description

This function plots log2 fold-change values for differentially expressed genes for each contrast in a result set. The set of genes displayed can be selected by means of several sorting methods. This makes it possible to view expression differences from a variety of perspectives. This function can be applied to a single or multiple result sets, making it possible to compare expression changes in specific genes across different experimental conditions.

Usage

```
de_profile_plot(res_list, filename = "de_profile_plot.pdf",
  sort_choice = "max", specific_genes = "", numGenes = 50,
  theme = 1, customLabels = FALSE, returnData = FALSE)
```

Arguments

`res_list`	A list of DESeq result sets. Results can be calculated individually using DESeq's results() function. Lists of results can be created by creating a list(result1, result2 ... result_N).
`filename`	Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /DE/profile_plots/ directory.
`sort_choice`	Gene selection is based on sorting method in cases where not all genes are displayed. sort_choice options are: "max", "min", "variance", "max_mean", "min_mean". "max" sorts genes based on the highest expression level of any single gene in a result set. In contrast, "max_mean" first calculates mean expression across all result sets and subsequently sorts by maximum mean values. Min and min_mean function similarly. Variance sorts genes by highest gene-wise variance in expression, displaying genes that showed the highest variability across all samples.
`specific_genes`	A character vector of gene names can be passed to this parameter to plot the genes specified. This overrides sorting and numGene parameters.
`numGenes`	The number of genes to include in this plot.
`theme`	Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.
`customLabels`	If customLabels is set to TRUE, the user will be prompted to provide a custom label for each label.
`returnData`	Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return a data frame for sort-selected differentially expressed genes containing gene names and log2 fold-change values relative to the experimental control condition.

Examples

```
## Not run:

 #This example assumes an experimenal design of ~Condition_Time.

#Prepare a result list.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)
```

```
/*
 * Sort data by the highest expression level for any individual gene in any sample.
 * Select the top 50 genes from this sort and visualize them in the plot.
 */
de_profile_plot(res_list=myResList, filename="DE_profile_upReg50.pdf",
                  sort_choice="max",
                  numGenes=50, theme=1, returnData=FALSE)

/*
 * Calculate the mean expression for each gene across all three time points.
 * Sort the data by minimum mean expression, select the top 25 genes,
 * and visualize them in the plot.
 */
de_profile_plot(res_list=myResList, filename="DE_profile_meanDownReg25.pdf",
                  sort_choice="min_mean",
                  numGenes=25, theme=1, returnData=FALSE)


/*
 * Calculate the variance for each gene across all three time points.
 * Sort the data by the highest gene-wise variance, select 30 genes
 * with the highest variance, and visualize them in the plot.
 * Save the data used to generate the plot as highVar_df.
 */
highVar_df <- de_profile_plot(res_list=myResList, filename="DE_profile_highVar30.pdf",
                          sort_choice="variance", numGenes=30, theme=1, returnData=TRUE)


## End(Not run)
```

`de_series` *Identify and visualize patterns of expression between differentially expressed genes across a series of result sets.*

Description

This function identifies patterns of expression across differentially expressed genes and clusters them into groups based on similar patterns of expression across multiple series, such as time series data (I.E. day1, day2, day3). The resulting group data are plotted to show how expression of groups change over the course of the series based on a generalized linear model with a displayed 95 percent confidence interval, or based on the mean expression for each point in the series. Similarity is calculated by clustering and merging genes until all differentially expressed genes have been clustered into the desired number of groups.

Usage

```
de_series(res_list, filename = "series_pattern.pdf", designVar, groupBy,
  numGroups = 5, theme = 1, method = "mean", customLabels = FALSE,
  returnData = FALSE, writeData = FALSE, fetchGroup = 0)
```

Arguments

`res_list` A list of DESeq result sets. Results can be calculated individually using DESeq's results() function. Lists of results can be created by creating a list(result1, result2 ... result_N).

`filename` Filename for output plot. Valid extensions are ".pdf" and ".png". File generation for plot file can be turned off using set_output_mode("screen"). Output will be written to the /DE/series_plots/ directory.

`designVar` The design field used when performing results() extractions. Must correspond to a metadata column. String.

`groupBy` The variable by which to group data in the plot. I.E. "Condition_Time". String.

`numGroups` The number of groups to split data into. Default = 5

`theme` Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.

`method` Method for drawing. "glm" uses generalized linear model to show overall trends over time. "mean" uses the mean expression value for each group as intersection points.

`customLabels` If customLabels is set to TRUE, the user will be prompted to provide a custom label for each label.

`returnData` If this value is true, this function will return a list of dataframes corresponding to groupwise splits. Default: FALSE

`writeData` If this value is true, this function will write data to file in the DE series folder for each group of genes. Default: FALSE

`fetchGroup` Specifically fetch gene information for a specific group. This will override standard returnData and instead return data for the specified group number.

Value

If returnData is true, this function will return a list of dataframes corresponding to groupwise splits.

Examples

```
## Not run:

 #This example assumes an experimenal design of ~Condition_Time.

#Prepare a result list.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

/*
 * Cluster genes by similarity into 5 groups, then visualize their expression over the
 * course of the series using a generalized linear model.
 */
de_series(res_list=myResList, filename="DE_series_pattern.pdf",
             designVar="Condition_Time",
             groupBy="Time", numGroups=5, theme=1, method="glm",
             returnData=FALSE, writeData=FALSE)
```

```
/*
 * Cluster genes by similarity into 3 groups, then visualize their expression over the
 * course of the series using based on mean group expression values.
 */
de_series(res_list=myResList, filename="DE_series_pattern.pdf",
            designVar="Condition_Time",
            groupBy="Time", numGroups=3, theme=2, method="mean",
            returnData=FALSE, writeData=FALSE)


## End(Not run)
```

de_volcano *Visualize the expression & significance of differentially expressed genes.*

Description

This function generates a volcano plot that displays the relation between expression and p-value for all DE genes in a result set.

Usage

```
de_volcano(res_list, filename = "volcano_plot.pdf", lfc_thresh = -1,
  strict_scale = TRUE, num_columns = 3, theme = 1,
  returnData = FALSE)
```

Arguments

`res_list`	A list of DESeq result sets. Results can be calculated individually using DESeq's results() function. Lists of results can be created by creating a list(result1, result2 ... result_N).
`filename`	Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /DE/volcano/ directory.
`lfc_thresh`	Log2 fold change threshold for highlighting genes on the volcano plot.
`strict_scale`	Boolean. If this is true scales for all samples will be forced to be identical, otherwise scales will vary depending on the dispersion of the volcano plot for each individual sample.
`num_columns`	Number of columns to use in the grid layout. Default=3.
`theme`	Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.
`returnData`	Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return the long-form table of expression containing sample names, catagorical grouping, and sample IDs.

Examples

```
## Not run:

#Make a volcano plot highlighting DE genes above 1.5 LFC threshold.
de_volcano(result_list, filename="volcano.pdf", lfc_thresh=1.5,
           strict_scale=TRUE, theme=1, returnData=FALSE)


## End(Not run)
```

---

enrich_res   *Incorporate additional data about differentially expressed genes into a DESeq2 result set.*

Description

Given a DESeq result set, as generated with DESeq2::results(), this function adds several additional components to the object. Variables created are numDE, numUp, numDown, allNames, upNames, and downNames, allDE, upDE, downDE, case, control, contrast, and designField. All calculations are performed based on significance thresholds set using the init_cutoffs() function. This function is typically used as a helper function for data aggregation.

Usage

```
enrich_res(res)
```

Arguments

res	A DESeq2 result set as generated through DESeq2::results(). Must be a DESeq object of type S4.

Value

An expanded DESeq result set of type DESeqResMeta containing the following fields: [numDE, numUp, numDown, allNames, upNames, and downNames, allDE, upDE, downDE, case, control, contrast, and designField]

See Also

init_cutoffs

Examples

```
## Not run:

#Enrich a result set. This will make several new data points available.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
enriched_result <- enrich_res(res.day1)

#Print number of differentially expressed genes.
print(enriched_result@numDE)

#Get the names of all up-regulated differentially expressed genes.
```

```
upreg_de_genes <- enriched_result@upNames

## End(Not run)
```

exclude_data_subset *Select a subset of count and target data based on metadata annotation.*

Description

This function, given count data and target metadata, will return a subset of both count and target data excluding data with values corresponding to selection parameters. For example, you can select count and target data to exclude only timepoint "day1". Multiple valid keep selections can also be used.

Usage

```
exclude_data_subset(counts, targets, target_count_id_map,
  target_exclude_col, target_exclude_val)
```

Arguments

counts	Count data as prepared with prep_counts().
targets	Target data as prepared with prep_targets().
target_count_id_map	Column in targets file whose values corresponds with count data column ids.
target_exclude_col	Column in target data to select values by. For instance, to exclude "day1" data from the "timepoint" column, this argument should be "timepoint".
target_exclude_val	Value from target_exclude_col to remove in the subset data. For instance, to select "day1" data from the "timepoint" column, this argument should be "day1". Multiple values can be selected by providing a concatenated set of values. I.E. To exclude day1 and day2 data, this argument should be, c("day1","day2").

Value

This function returns count and target data based on the target_exclude_col and target_exclude_val parameters. Counts will be returned as the [[1]] index and target data will be returned as the [[2]] index.

See Also

prep_counts, prep_targets, keep_data_subset

Examples

```
## Not run:

myCounts <- prep_counts(count_input="master_count_data.txt", delim="t")
myTargets <- prep_targets(target_input="master_count_data.txt", delim="t")

#Get data for only "timepoint" at "day1".
```

```
data_subset <- exclude_data_subset(counts=myCounts, targets=myTargets,
                                  target_count_id_map="sample_id",
                                  target_exclude_col="timepoint",
                                  target_exclude_val="day1")

#Count data is stored in the first index returned by the function.
subset_counts <- data_subset[[1]]

#Target data is stored in the second index returned by the function.
subset_targets <- data_subset[[2]]


#Get data for only "timepoint" at "day1" and "day2".
data_subset <- exclude_data_subset(counts=myCounts, targets=myTargets,
                                  target_count_id_map="sample_id",
                                  target_exclude_col="timepoint",
                                  target_exclude_val=c("day1","day2"))

#Count data is stored in the first index returned by the function.
subset_counts <- data_subset[[1]]

#Target data is stored in the second index returned by the function.
subset_targets <- data_subset[[2]]


## End(Not run)
```

get_de_data *Aggregate and retrieve data from multiple differentially expressed result sets.*

Description

This function finds the union or intersection of DE gene names for all result sets provided, extracts those rows from all DE result sets, and merges the values into a single data set containing log2 fold-change or padj values.

Usage

```
get_de_data(res_list, method = "union", type = "lfc",
  lfc_filter = FALSE)
```

Arguments

res_list A list of DESeq result sets created with DESeq2::results(). I.E: list(res1, res2, ..., resN).

method Method for aggregating data. "union" will gather expression data for all samples in results list when a gene is differentially expressed in at least one sample. "intersection" will gather expression data only for genes that are DE in all samples of the result list.

type Type of data to retrieve. Options are "lfc" (log2foldchange) and "padj" (adjusted p-value).

lfc_filter Filter genes based on a minimum log fold-change as initialized in init_cutoffs(). Boolean. Default=FALSE.

Value

Returns the aggregated data frame containing fold-change or p-values for the provided result sets.

See Also

init_cutoffs

Examples

```
## Not run:

#Prepare a result list for aggregation.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)


/*
 * Data for all result sets will be included for each gene if that gene was found to be
 * differentially expressed in at least one of the provided result sets.
 * Filter based on a minimum fold-change.
 */
aggregated_lfc_union  <- get_de_data(res_list=myResList, method="union",
                                      type="lfc", lfc_filter=TRUE)
aggregated_pval_union <- get_de_data(res_list=myResList, method="union",
                                      type="padj", lfc_filter=TRUE)


/*
 * Data for all result sets will be included for each gene only if that gene was found to
 * be differentially expressed in all provided result sets. Do not apply a fold-change filter.
 * Significance is determined only by p-value threshold.
 */
aggregated_lfc_intersect  <- get_de_data(res_list=myResList, method="intersection",
                                          type="lfc", lfc_filter=FALSE)
aggregated_pval_intersect <- get_de_data(res_list=myResList, method="intersection",
                                          type="padj", lfc_filter=FALSE)


## End(Not run)
```

init_cutoffs *Initialize cutoff values for significance and fold-change filtering.*

Description

This function initializes the p-value cutoffs and log2foldChange values for most visualizations, filtering, and aggregation steps of this package. Significant differentially expressed genes will be initially determined based on the provided p-value cutoff and filtering, when appropriate, will generally be based on the provided log fold-change cutoff initialized by this function.

Usage

```
init_cutoffs(p_signif = 0.05, lfc_cut = 1)
```

Arguments

`p_signif`	P-value cutoff for determining DE genes. Default: 0.05
`lfc_cut`	Log2 fold-change cutoff. Default: 1

Value

This function does not return a value.

Examples

```
## Not run:

init_cutoffs(p_signif=0.01, lfc_cut=1.5)


## End(Not run)
```

`init_data_paths` *Initialize data paths for count and target files.*

Description

This function initializes the paths to folders containing count and target data. This package assumes correspondence between count data column names and target data row names. In this package, metadata files are refered to as target data. Initialization by this function is required by most functions.

Usage

```
init_data_paths(count_path, target_path)
```

Arguments

`count_path`	Path to directory containing count data. Count data is assumed to be tab-delimited text represent raw expression counts for each gene, with samples indicated by column and genes identified by row. Column names in count data must correspond to row names in target files.
`target_path`	Path to directory containing target data. Target data can be tab-delimited or comma-seperated text. Target rownames must correspond to count column names. Additionally, any data can be included in target files to represent experimental conditions or additional metadata for the project.

Value

This function does not return a value.

See Also

create_dir_struct

Examples

```
## Not run:

#Initialize paths to count and target file directories.
init_data_paths(count_path="/Users/adam/projects/ebola/counts/",
                target_path="/Users/adam/projects/ebola/")


## End(Not run)
```

---

keep_data_subset *Select a subset of count and target data based on metadata annotation.*

Description

This function, given count data and target metadata, will return a subset of both count and target data including only data with values corresponding to selection parameters. For example, you can select count and target data for only timepoint "day1". Multiple valid keep selections can also be used.

Usage

```
keep_data_subset(counts, targets, target_count_id_map, target_keep_col,
  target_keep_val)
```

Arguments

`counts` Count data as prepared with prep_counts().

`targets` Target data as prepared with prep_targets().

`target_count_id_map`
Column in targets file whose values corresponds with count data column ids.

`target_keep_col`
Column in target data to select values by. For instance, to select "day1" data from the "timepoint" column, this argument should be "timepoint".

`target_keep_val`
Value from target_keep_col to keep in the subset data. For instance, to select "day1" data from the "timepoint" column, this argument should be "day1". Multiple values can be selected by providing a concatenated set of values. I.E. To keep day1 and day2 data, this argument should be, c("day1","day2").

Value

This function returns count and target data based on the target_keep_col and target_keep_val parameters. Counts will be returned as the [[1]] index and target data will be returned as the [[2]] index.

See Also

prep_counts, prep_targets, exclude_data_subset

Examples

```
## Not run:

myCounts <- prep_counts(count_input="master_count_data.txt", delim="t")
myTargets <- prep_targets(target_input="master_count_data.txt", delim="t")

#Get data for only "timepoint" at "day1".
data_subset <- keep_data_subset(counts=myCounts, targets=myTargets,
                                  target_count_id_map="sample_id",
                                  target_keep_col="timepoint",
                                  target_keep_val="day1")

#Count data is stored in the first index returned by the function.
subset_counts <- data_subset[[1]]

#Target data is stored in the second index returned by the function.
subset_targets <- data_subset[[2]]


#Get data for only "timepoint" at "day1" and "day2".
data_subset <- keep_data_subset(counts=myCounts, targets=myTargets,
                                  target_count_id_map="sample_id",
                                  target_keep_col="timepoint",
                                  target_keep_val=c("day1","day2"))

#Count data is stored in the first index returned by the function.
subset_counts <- data_subset[[1]]

#Target data is stored in the second index returned by the function.
subset_targets <- data_subset[[2]]


## End(Not run)
```

---

make_composite_field *Create a composite metadata field by merging existing data.*

Description

This function, given target metadata, will create a new composite column based on two or more existing columns in the target data. The new field will be named based on the merged fields and will be delimited using the "_" character. Requires that target data has been prepared with prep_targets().

Usage

```
make_composite_field(merge_fields)
```

Arguments

merge_fields   Column names in targets data to be included in the new composite column. Should be provided as a set of strings. I.E. c("field1","field2").

Value

This function returns target data containing the newly created composite field.

See Also

prep_targets

Examples

```
## Not run:

myCounts <- prep_counts(count_input="master_count_data.txt", delim="t")
myTargets <- prep_targets(target_input="master_count_data.txt", delim="t")

#Create a composite field based on "treatment" and "time" fields.
myTargets <- make_composite_field(targets=myTargets,
                                  merge_fields=c("treatment","time"))


#Create a composite field based on "treatment", "time", and "patientID" fields.
myTargets <- make_composite_field(targets=myTargets,
                                  merge_fields=c("treatment","time","patientID"))



## End(Not run)
```

---

plot_dendro *Create dendrograms based on hierarchical clustering.*

Description

This function plots a dendrogram for the dataset as a whole using hierarchical clustering. Metadata can be applied to the dendrogram to examine how data cluster by different conditions. By examining how data cluster, it is possible to identify patterns in the data and batch effects. For example, creating a dendrogram that is grouped by the "case" condition in a case vs control experiment, it would be possible to identify a strong effect in the overall dataset between the case and control condition if most of the "case" conditions clustered strongly together. Batch effects can also be identified in this way. For instance, if samples were prepared at two different locations and this information was incorporated into the metadata as a "batch" column, visualizing the data based on batch data could indicate whether or not differences in the data are due to differences in the sample prep location, rather than due to a biological effect, depending on how strongly the batches cluster.

Usage

```
plot_dendro(filename = "dendro_plot.pdf", id_field, groupBy)
```

Arguments

`filename`	Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to /dendrograms/ directory.

`id_field`	The unique sample ID field for your data. Should correspond to a column in targets data.
`groupBy`	A field in target meta data that you wish to cluster data by.

Value

This function does not return a value.

Examples

```
## Not run:

#Plot a dendrogram based on an "infection" metadata field.
plot_dendro(filename="infection_dendro.pdf", "sample_id", "infection")

#Plot a dendrogram based on an "batch" metadata field.
plot_dendro(filename="batch_dendro.pdf", "sample_id", "batch")


## End(Not run)
```

plot_euclid_dist *Visualize the euclidian distances between samples.*

Description

This function computes and plots sample euclidian distances in a symmetrical heatmap. This visualization can be used to examine overall similarity between samples and whether or not data are behaving as expected based on the experimental design. This view can also be used to identify potential outlying samples.

Usage

```
plot_euclid_dist(row_labels, filename = "euclidian_distance.pdf",
  theme = 1, returnData = FALSE)
```

Arguments

`row_labels`	Row labels for samples. This value should correspond to a column in the target data and will be used to label each row in the heatmap. Viewing the plot in the context of different metadata makes it possible to search for outliers and batch effects, in addition to whether or not the data behave as intended. String.
`filename`	Output filename in string format. Valid formats are .pdf and .png. File generation can be turned off using set_output_mode("screen"). Output will be written to the /sample_distance/euclidian/ directory.
`theme`	Determines color scheme used for this plot. Valid options are integers, 1-6.
`returnData`	Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return the sample distance matrix of euclidian distance measurements between samples.

See Also

plot_poisson_dist

Examples

```
## Not run:

#Visualize euclidian distances of all samples using "SampleID" target data as labels.
plot_euclid_dist("SampleID", filename="euclidian_distance.pdf",
                 theme=2, returnData=FALSE)

#Visualize euclidian distances of all samples using "timepoint" target data as labels.
#Store the resulting distance matrix data.
distMatrix <- plot_euclid_dist("timepoint", filename="euclidian_distance.pdf",
                              theme=2, returnData=TRUE)


## End(Not run)
```

plot_gene *Visualize the expression of a specific gene with regard to metadata grouping.*

Description

This function generates a box plot to display the expression of an individual gene with regard to a specified grouping that can be based on any data that exists in the targets file. For example, a plot could be created to view the expression of a gene as a function of different time points or experimental conditons.

Usage

```
plot_gene(filename = "gene_plot.pdf", gene_name, groupBy, theme = 1,
  returnData = FALSE)
```

Arguments

filename	Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /geneplots/ directory.
gene_name	The name of the gene to create boxplot for. Must match a rowname in count data. String.
groupBy	The group from target data that should be used to split data. Must match a rowname in count data. I.E. "Timepoint" or "Infection". String. If only two groups are present in this variable a wilcox test will be performed between the two groups and the p-value will be displayed on the plot as well.

`theme` Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.

`returnData` Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return the long-form table of expression containing sample names, catagorical grouping, and sample IDs.

Examples

```
## Not run:

#Plot the CAPS12 gene for each time point.
plot_gene(filename="CAPS12_time_plot.pdf", gene_name="CAPS12",
          groupBy="Time", theme=1, returnData=FALSE)

#Plot the METTL25 gene for each "response" group.  Store the long-form data table.
mettl25_dat <- plot_gene(filename="METTL25_time_plot.pdf", gene_name="METTL25",
                         groupBy="response", theme=2, returnData=TRUE)


## End(Not run)
```

plot_group_stats *Visualize overall data set as a function of a metadata grouping.*

Description

This function plots groupwise expression as with regard to a specified target data grouping. This function can be used to vizualize the overall data set according to metadata grouping. For instance, overall expression at different timepoints can be viewed, or infected vs control expression levels can be plotted. This visualization is also ideal for examining the impact of normalization or filtering.

Usage

```
plot_group_stats(filename = "group_stats_plot.pdf", id_field, groupBy,
  normalized = TRUE, theme = 1, returnData = FALSE)
```

Arguments

`filename` Filename for output plot. Valid extensions are ".pdf" and ".png". File generation can be turned off using set_output_mode("screen"). Output will be written to the /group_stats/ directory.

`id_field` The unique ID field for this data set. Should be the field that corresponds to count column names and target rownames.

`groupBy` The field by which to group samples in the boxplot. I.E. "time"

`normalized` Toggles between whether normalized or non-normalized data should be used. Boolean. Default=TRUE.

`theme` Theme for the layout and color scheme for the plot. Valid selections are integers between 1-6.

`returnData` Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return the long-form table of expression containing catagorical grouping, and sample IDs, log fold-change, and gene name.

Examples

```
## Not run:

#Plot the overall data set expression levels grouped by time.
plot_group_stats("group_stats_plot.pdf", "SampleID", groupBy="Time",
                 normalized=TRUE, theme=1, returnData=FALSE)


## End(Not run)
```

plot_mds *Visualize multi-dimensional scaled data for all samples, with group-wise metadata incorporated as colors and shapes.*

Description

This function performs multi-dimensional scaling (MDS) on data to visualize levels of similarity between individual samples. Colors and shape variables can be specified according to available metadata, making it possible to identify patterns in the data based on groups. For instance, by coloring data points based on "Infected/Non-infected" groups, and drawing data point shapes based on time point, it is possible to view the differences between infected and non-infected samples across multiple time points.

Usage

```
plot_mds(filename = "mds_plot.pdf", color_var, shape_var = "none",
  showConf = TRUE, theme = 1, customLabels = FALSE,
  returnData = FALSE)
```

Arguments

`filename` Filename for output plot. Valid extensions are ".pdf" and ".png". File generation for plot file can be turned off using set_output_mode("screen"). Output will be written to the /MDS/standard/ directory.

`color_var` The group from target data that should be indicated by color. Any column of metadata can be used, regardless of experimental design.

`shape_var` The group from target data that should be indicated by shape. Any column of metadata can be used, regardless of experimental design. (Optional). Default="none".

showConf Boolean. Draw an elipsis representing the 95 percent confidence interval around each group. Default: TRUE

theme Theme for the layout and color scheme for the MDS plot. Valid selections are integers between 1-6.

customLabels If customLabels is set to TRUE, the user will be prompted to provide a custom label for each label.

returnData If this value is true, this function will return a sample distance matrix. Default: FALSE

Value

If returnData is true, this function will return a matrix containing sample distances computed to create the MDS plot.

See Also

plot_mds_hulls

Examples

```
## Not run:

#These examples assume "Time" and "Infection" columns exist in target metadata.

#Make a MDS plot showing all samples, colored based on time point.
plot_mds(filename="mds_plot.pdf", color_var="Time", shape_var="none",
          theme=1, returnData=FALSE)

#Make a MDS plot, colored based on time point with shapes based on infection.
plot_mds(filename="mds_plot.pdf", color_var="Time", shape_var="Infection",
          theme=2, returnData=FALSE)


## End(Not run)
```

plot_mds_hulls *Visualize multi-dimensional scaled data for all samples, with group-wise metadata incorporated as colors, shapes, and convex hulls.*

Description

This function performs multi-dimensional scaling (MDS) on data to visualize levels of similarity between individual samples. Distance measurements can be calculated based only on the differentially expressed genes for each sample. Colors and shape variables can be specified according to available metadata, making it possible to identify patterns in the data based on groups. Additionally, convex hulls are computed and drawn indicating the overall clustering range of samples in a group-wise manner.

Usage

```
plot_mds_hulls(filename = "mds_hulls_plot.pdf", color_var,
  shape_var = "none", deOnly = FALSE, showLabel = FALSE,
  hullType = "solid", exclude_data = FALSE, idCol = "",
  excludeCol = "", excludeName = "", theme = 1,
  customLabels = FALSE, returnData = FALSE)
```

Arguments

`filename` Filename for output plot. Valid extensions are ".pdf" and ".png". File generation for plot file can be turned off using set_output_mode("screen"). Output will be written to the /MDS/hulls/ directory.

`color_var` The group from target data that should be indicated by color. Any column of metadata can be used, regardless of experimental design. String.

`shape_var` The group from target data that should be indicated by shape. Any column of metadata can be used, regardless of experimental design. String. Optional. Default="none".

`deOnly` Use only DE genes when computing the plot. Requires that create_master_res() be run first. Boolean.

`showLabel` Show labels indicating sample identifiers on the plot. Boolean. Default=FALSE.

`hullType` Determines hull type, either "solid" or "outline". Solid hulls are partially transparent overlaying polygons. Outline hulls trace an outline along the outermost distant samples of each group. Default="solid".

`exclude_data` Exclude some subset of data from this plot. If this is TRUE, three additional parameters must be provided: idCol, excludeCol, and excludeName. These will then be used to subset the data for the MDS as desired. For example, it is possible to remove "Day1" data from your "Timepoint" field, or "Control_Sample" from your condition field. I.E. Remove control samples from the visualization. Logical. Default=FALSE.

`idCol` Required only if exclude_data is TRUE. The ID column in target data that corresponds to column names in count data.

`excludeCol` Required only if exclude_data is TRUE. The field from which exclusion criteria will be determined. Must match a column name from target data.

`excludeName` Required only if exclude_data is TRUE. Value to exclude from the data based on the excludeCol. Must match a level in the excludeCol.

`theme` Theme for the layout and color scheme for the MDS plot. Valid selections are integers between 1-6.

`customLabels` If customLabels is set to TRUE, the user will be prompted to provide a custom label for each label.

`returnData` If this value is true, this function will return a sample distance matrix. Default: FALSE

Value

If returnData is true, this function will return a matrix containing sample distances computed to create the MDS plot.

See Also

`plot_mds`

Examples

```
## Not run:

#These examples assume "Time", "Infection", and sampleType columns exist in target metadata.

 /*
  * Create a MDS plot with convex hulls drawn based on "Time" metadata.
  * Shapes are based on "Infection".  Compute based on data from all genes.
  */
plot_mds_hulls("mds_hull_plot.pdf", "Time", "Infection",
                deOnly=FALSE, showLabel=FALSE, hullType="solid",
                theme=1)

 /*
  * Create a MDS plot with convex hulls drawn based on "Time" metadata.
  * Shapes are based on "Infection".  Compute based on data from only
  * differentailly expressed genes.  Hulls drawn only as outlines.
  */
plot_mds_hulls("mds_hull_deOnly_plot.pdf", "Time", "Infection",
                deOnly=TRUE, showLabel=FALSE, hullType="outline",
                theme=2)

 /*
  * Create a MDS plot with convex hulls drawn based on "Time" metadata.
  * Shapes are based on "Infection".  Compute based on data from only
  * differentailly expressed genes.  Do not show samples labeled
  * as "mock" from "sampleType" column of target metadata.
  */
plot_mds_hulls("mds_hull_deOnly_plot.pdf", "Time", "Infection",
                deOnly=TRUE, showLabel=FALSE, hullType="outline",
                exclude_data=TRUE, idCol="sampleID",
                excludeCol="sampleType", excludeName="mock"
                theme=4)


## End(Not run)
```

plot_poisson_dist *Visualize the poisson distances between samples.*

Description

This function computes and plots sample poisson distances in a symmetrical heatmap. This visualization can be used to examine overall similarity between samples and whether or not data are behaving as expected based on the experimental design. Unlike euclidian distance, poisson distance calculation takes into account variation between sample counts when calculating distances. This view can also be used to identify potential outlying samples.

Usage

```
plot_poisson_dist(row_labels, filename = "poisson_distance.pdf",
  theme = 1, returnData = FALSE)
```

Arguments

`row_labels`	Row labels for samples. This value should correspond to a column in the target data and will be used to label each row in the heatmap. Viewing the plot in the context of different metadata makes it possible to search for outliers and batch effects, in addition to whether or not the data behave as intended. String.
`filename`	Output filename in string format. Valid formats are .pdf and .png. File generation can be turned off using set_output_mode("screen"). Output will be written to the /sample_distance/poisson/ directory.
`theme`	Determines color scheme used for this plot. Valid options are integers, 1-6.
`returnData`	Boolean. Determines if this visualization should return data used to generate the visualization. Default=FALSE.

Value

If returnData is true, this function will return the sample distance matrix of poisson distance measurements between samples.

See Also

`plot_euclid_dist`

Examples

```
## Not run:

#Visualize euclidian distances of all samples using "SampleID" target data as labels.
plot_poisson_dist("SampleID", filename="poisson_distance.pdf", theme=2, returnData=FALSE)

#Visualize euclidian distances of all samples using "timepoint" target data as labels.
#Store the resulting distance matrix data.
distMatrix <- plot_poisson_dist("timepoint", filename="poisson_distance.pdf",
                                 theme=2, returnData=TRUE)


## End(Not run)
```

`prep_counts` *Read tab or comma delimited count data.*

Description

This function takes a count data file and returns the formatted data. Requires that init_data_paths() has been run.

Usage

```
prep_counts(count_input, delim = "t")
```

Arguments

count_input	Count data in tab-delimited or comma-delimited (CSV) format, assumes headers exist for each column. Note that column names in the input count file must match row names in the input target metadata file.
delim	Indicate if file is comma or tab delimited. "t" indicated tab delimited and "c" indicates comma seperated data. Default: "t"

Value

This function will return a properly formatted count data table based on a provided input file.

See Also

prep_targets, prep_dds_from_data, init_data_paths

Examples

```
## Not run:

#Read a tab-delimited text file as count input.
myCounts <- prep_counts(count_input="master_count_data.txt", delim="t")


## End(Not run)
```

prep_dds_from_data *Prepare a DESeq2 object based on count and target data.*

Description

This function takes count and targets data and creates a DESeq2 object based on the provided design.

Usage

```
prep_dds_from_data(count_input, target_input, experiment_design,
  collapseReps = FALSE, rep_field_vector = "Replicate",
  stabilization = "rld")
```

Arguments

count_input	Count data in dataframe format as read with prep_counts(). Column names must correspond to target rownames.
target_input	Target data in dataframe formatas read with prep_targets(). Row names must correspond to count column names.
experiment_design	Experimental design for DE comparison. Can include multiple factors. Design factors should correspond to column names in the provided targets file. Include the primary factor as the final factor in the design and any secondary or batch effects prior. Must be prefaced with a tilde (~). I.E. ~ Batch1 + Batch2 + SecondaryCondition + PrimaryCondition

`collapseReps` Boolean. Collapse technical replicates. Default: False

`rep_field_vector`
Target metadata column identifying which samples correspond to which replicates and maps samples to their replicate identifiers. Required if collapseReps is TRUE.

`stabilization` Method of normalizing transformation to perform. Possible values are "rld" and "vst". rld will execute DESeq2's rlog transformation, whereas vst will apply variance stabilizing transformation to the data. For larger data sets vst is recommended.

Value

This function will return a properly formatted DESeq2 object based on the provided normalization method and experimental design.

See Also

`prep_counts`, `prep_targets`, `init_cutoffs`

Examples

```
## Not run:

/*
 * Create a DESeq2 data object using previously read count and target data.
 * Differential expression groups indicated by "Infection" column of target data.
 * Apply a rlog transformation to the data. TMM normalization applied.
 */
dds <- prep_dds_from_data(count_input=myCounts, target_input=myTargets,
                           experiment_design= ~ Infection, stabilization="rld")


/*
 * Create a DESeq2 data object using previously read count and target data.
 * Differential expression groups indicated by "Infection" column of target data.
 * Takes into account a batch effect column of target data.
 * Apply a variance stabilizing transformation to the data. TMM normalization applied.
 */
dds <- prep_dds_from_data(count_input=myCounts, target_input=myTargets,
                           experiment_design= ~ batch + Infection, stabilization="vst")


/*
 * Create a DESeq2 data object using previously read count and target data.
 * Differential expression groups indicated by "Time" and "Infection" columns of target data.
 * Takes into account a batch effect column of target data.
 * Apply a variance stabilizing transformation to the data. TMM normalization applied.
 * Collapse technical replicates based on the "replicate" column of target data.
 */
dds <- prep_dds_from_data(count_input=myCounts, target_input=myTargets,
                           experiment_design= ~ batch + Time + Infection,
                           collapseReps=TRUE,
                           rep_field_vector="replicate",
                           stabilization="vst")
```

```
## End(Not run)
```

---

prep_targets *Read tab or comma delimited target metadata file.*

Description

This function takes a targets metadata file and returns formatted data. Requires that init_data_paths() has been run.

Usage

```
prep_targets(target_input, delim = "t")
```

Arguments

`target_input`	Target data in tab seperated format, assumes headers exist for each column. Note that column names in the input count file must match row names in the input target metadata file.
`delim`	Indicate if file is comma or tab delimited. "t" indicated tab delimited and "c" indicates comma seperated data. Default: "t"

Value

This function will return a properly formatted targets data table based on a provided input file.

See Also

`prep_counts`, `prep_dds_from_data`, `init_data_paths`

Examples

```
## Not run:

#Read a tab-delimited text file as target metadata input.
myTargets <- prep_targets(target_input="master_count_data.txt", delim="t")


## End(Not run)
```

set_output_mode *Determine if visualizations are written to file, printed to screen, or both.*

Description

This function initializes the output mode for the package to determine if visualizations should be written to file, printed to screen, or both.

Usage

```
set_output_mode(toggle = "both")
```

Arguments

toggle	Desired output mode. Determines how output plot data will be displayed. Options are "screen", "file", or "both".

Value

This function does not return a value.

Examples

```
## Not run:

#Don't save files, just print to the screen.
set_output_mode("screen")

#Print to the screen and save files to appropriate locations.
set_output_mode("both")

#Save files to appropriate locations, but do not print them to screen.
set_output_mode("file")


## End(Not run)
```

transpose_gene_ids *Rename gene IDs based on a 1-to-1 mapping file.*

Description

This function accepts a 1-to-1 mapping file of gene ids and updates count data to reflect the new IDs. The mapping file should contain two columns "Gene stable ID" and "Gene name". The "Gene stable id" should correspond to gene ids in your count data file, and "gene name" will be used to replace the existing genes. This function has can be used to translate difficult to understand gene identifiers, such as ensembl IDs to to more human readable gene names. A mapping file can be generated at http://www.ensembl.org/biomart/martview/. This function should be run using count data read by prep_counts(). The returned data can then be used in prep_dds_from_data().

Usage

```
transpose_gene_ids(count_data, meta_file)
```

Arguments

count_data   Count data as read using the prep_counts() function.

meta_file   File containing 1-to-1 gene id mapping. Tab-delimited format expected. Column 1 should be titled, "Gene stable ID" and column 2 should be titled "Gene name".

Value

This function returns a count data set with gene IDs replaced by those provided in the meta_file parameter.

See Also

http://www.ensembl.org/biomart/martview/, prep_counts, prep_dds_from_data

Examples

```
## Not run:

#Transpose IDs from current count data to their desired values based on the meta_file.
count_data  <- prep_counts(my_count_file)
new_count_data <- transpose_gene_ids(count_data, meta_file)

#Updated gene ID count information can be used in DE analysis.
dds <- prep_dds_from_data(new_count_data, target_data, ~ Design_Type, TRUE,
                           "Replicate", "vst")


## End(Not run)
```

write_all_de_results *Write differentially expressed gene data for multiple result sets to file.*

Description

This function accepts a list of DE result sets and writes their outputs individually to file. Relies on init_cutoffs() significance thresholds. Filenames will be generated automatically based on the contrasts performed during the results() function. Output will be written to the /DE/data/ folder.

Usage

```
write_all_de_results(res_list, lfc_filter = FALSE)
```

Arguments

res_list   A list of DESeq result sets.

lfc_filter   Also impose a filter on the DE set by a minimum absolute log2foldChange as determined by init_cutoffs(). Default=FALSE.

Value

Output for each result set will be written to file This function does not return a value.

See Also

`init_cutoffs`, `create_dir_struct`

Examples

```
## Not run:

#Prepare a result list for aggregation.
res.day1 <- results(dds, contrast=c("Condition_Time", "day1_disease", "day1_control"))
res.day2 <- results(dds, contrast=c("Condition_Time", "day2_disease", "day2_control"))
res.day3 <- results(dds, contrast=c("Condition_Time", "day3_disease", "day3_control"))
myResList <- list(res.day1, res.day2, res.day3)

#Write differentally expressed gene data for each contrast to file.
#Include a minimum fold change filter. This will output 3 files.
write_all_de_results(res_list=myResList, lfc_filter=TRUE)


## End(Not run)
```

Index