Package 'BatchQC'

April 5, 2026

Type Package

Title Batch Effects Quality Control Software

Version 2.6.1

Date 2026-02-10

Description Sequencing and microarray samples often are collected or processed in multiple batches or at different times. This often produces technical biases that can lead to incorrect results in the downstream analysis. BatchQC is a software tool that streamlines batch preprocessing and evaluation by providing interactive diagnostics, visualizations, and statistical analyses to explore the extent to which batch variation impacts the data. BatchQC diagnostics help determine whether batch adjustment needs to be done, and how correction should be applied before proceeding with a downstream analysis. Moreover, BatchQC interactively applies multiple common batch effect approaches to the data and the user can quickly see the benefits of each method. BatchQC is developed as a Shiny App. The output is organized into multiple tabs and each tab features an important part of the batch effect analysis and visualization of the data. The BatchQC interface has the following analysis groups: Summary, Differential Expression, Median Correlations, Heatmaps, Circular Dendrogram, PCA Analysis, Shape, ComBat and SVA.

License MIT + file LICENSE

URL https://github.com/wejlab/BatchQC

BugReports https://github.com/wejlab/BatchQC/issues

Depends R (>= 4.5.0)

Imports data.table, DESeq2, dplyr, EBSeq, edgeR, FNN, ggdendro, ggnewscale, ggplot2, limma, matrixStats, methods, MASS, pheatmap, RColorBrewer, reader, reshape2, scran, shiny, shinyjs, shinythemes, stats, SummarizedExperiment, sva, S4Vectors, tibble, tidyr, tidyverse, umap, utils

Suggests BiocManager, BiocStyle, bladderbatch, curatedTBData, devtools, knitr, lintr, MultiAssayExperiment, plotly, rmarkdown, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews BatchEffect, GraphAndNetwork, Microarray, Normalization, PrincipalComponent, Sequencing, Software, Visualization, QualityControl, RNASeq, Preprocessing, DifferentialExpression, ImmunoOncology

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Collate 'Normalization.R' 'batch_correction.R' 'color_palette.R' 'compute_aic.R' 'covariates_not_confounded.R' 'data.R' 'dendrogram.R' 'dendrogram_alpha_numeric_check.R' 'dendrogram_color_palette.R' 'differential_expression.R' 'explained_variation.R' 'global_vars.R' 'heatmap.R' 'heatmap_num_to_char_convertor.R' 'import.R' 'kBET-utils.R' 'kBET.R' 'lambda_statistic.R' 'negative_binomial_check.R' 'pca.R' 'preprocess.R' 'runApp.R' 'summary_statistics.R' 'umap.R' 'variation_ratios.R' 'volcano_plot.R'

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

git_branch RELEASE_3_22

git_last_commit 7a6c41a

git_last_commit_date 2026-02-10

Repository Bioconductor 3.22

Date/Publication 2026-04-05

Author Jessica Anderson [aut] (ORCID: <https://orcid.org/0000-0002-0542-9872>), W. Evan Johnson [aut] (ORCID: <https://orcid.org/0000-0002-6247-6595>), Yaoan Leng [ctb, cre] (ORCID: <https://orcid.org/0009-0002-3957-5250>), Solaiappan Manimaran [aut], Heather Selby [ctb], Claire Ruberman [ctb], Kwame Okrah [ctb], Hector Corrada Bravo [ctb], Michael Silverstein [ctb], Regan Conrad [ctb], Zhaorong Li [ctb], Evan Holmes [ctb], Solomon Joseph [ctb], Howard Fan [ctb]

Maintainer Yaoan Leng <leng@bu.edu>

Contents

`BatchQC` *Run BatchQC shiny app*

Description

Run BatchQC shiny app

Usage

```
BatchQC(dev = FALSE)
```

Arguments

`dev`	Run the application in developer mode

Value

The shiny app will open

Examples

```
if(interactive()){
BatchQC()
}
```

`batchqc_explained_variation` *Returns a list of explained variation by batch and condition combinations*

Description

Returns a list of explained variation by batch and condition combinations

Usage

```
batchqc_explained_variation(se, batch, condition = NULL, assay_name)
```

Arguments

se	Summarized experiment object
batch	Batch covariate
condition	Condition covariate(s) of interest if desired, default is NULL
assay_name	Assay of choice

Value

List of explained variation by batch and condition

Examples

```
library(scran)
se <- mockSCE()
batchqc_explained_variation <- BatchQC::batchqc_explained_variation(se,
                                        batch = "Mutation_Status",
                                        condition = "Treatment",
                                        assay_name = "counts")
batchqc_explained_variation
```

---

batch_correct *Batch Correct This function allows you to Add batch corrected count matrix to the SE object*

Description

Batch Correct This function allows you to Add batch corrected count matrix to the SE object

Usage

```
batch_correct(se, method, assay_to_normalize, batch, group = NULL,
covar, output_assay_name, ...)
```

Arguments

se	SummarizedExperiment object
method	Normalization Method ("ComBat-Seq", "ComBat", "limma", "sva", svaseq)
assay_to_normalize	Which assay use to do normalization
batch	The batch
group	The group variable
covar	list of covariates
output_assay_name	name of results assay
...	Arguments to be passed to specific methods, such as num_sv for svaseq_correction and psva for sva_correction.

Value

a summarized experiment object with normalized assay appended

Examples

```
library(scran)
se <- mockSCE()
se <- BatchQC::batch_correct(se, method = "ComBat-Seq",
                                    assay_to_normalize = "counts",
                                    batch = "Mutation_Status",
                                    covar = "Treatment",
                                    output_assay_name =
                                        "ComBat_Seq_Corrected")
se <- BatchQC::batch_correct(se, method = "ComBat",
                                    assay_to_normalize = "counts",
                                    batch = "Mutation_Status",
                                    covar = "Treatment",
                                    output_assay_name =
                                        "ComBat_Corrected")
se
```

---

batch_design *This function allows you to make a batch design matrix*

Description

This function allows you to make a batch design matrix

Usage

```
batch_design(se, batch, covariate)
```

Arguments

se	summarized experiment object
batch	string, batch variable
covariate	string, biological covariate

Value

design table

Examples

```
library(scran)
se <- mockSCE()
batch_design_tibble <- batch_design(se, batch = "Mutation_Status",
                                                    covariate = "Treatment")
batch_design_tibble
```

batch_indicator *Batch and Condition indicator for signature data*

Description

This dataset is from signature data captured when activating different growth pathway genes in human mammary epithelial cells (GEO accession: GSE73628). This data consists of three batches and ten different conditions corresponding to control and nine different pathways.

Usage

```
data(batch_indicator)
```

Format

A data frame with 89 rows and 2 variables:

batch batch

condition condition

bisect *bisect - a generic bisection function*

Description

adapted from kBET package (https://github.com/theislab/kBET). Provides recursive bisection algorithm for an arbitrary function. It evaluates the function `foo` at the bounds and replaces one of the boundaries until a maximum is found or the interval becomes too small

Usage

```
bisect(foo, bounds, known = NULL, ..., tolx = 5, toly = 0.01)
```

Arguments

`foo`	a function mapping a one-dim argument to one-dim value
`bounds`	a vector of length 2 with real valued numbers (i.e. two arguments of `foo`)
`known`	tells for which of the arguments a value is known (defaults to NULL)
`...`	additional parameters for `foo`
`tolx`	break condition for argument (defaults to 10)
`toly`	break condition for value (defaults to 0.01)

Value

A range of bounds where `foo` is maximal.

Examples

```
get_maximum <- bisect(function(x) {
  -(x - 2)^2
}, c(-5, 50))
```

bladder_data_upload *Bladder data upload This function uploads the Bladder data set from the bladderbatch package. This dataset is from bladder cancer data with 22,283 different microarray gene expression data. It has 57 bladder samples with 3 metadata variables (batch, outcome and cancer). It contains 5 batches, 3 cancer types (cancer, biopsy, control), and 5 outcomes (Biopsy, mTCC, sTCC-CIS, sTCC+CIS, and Normal). Batch 1 contains only cancer, 2 has cancer and controls, 3 has only controls, 4 contains only biopsy, and 5 contains cancer and biopsy*

Description

Bladder data upload This function uploads the Bladder data set from the bladderbatch package. This dataset is from bladder cancer data with 22,283 different microarray gene expression data. It has 57 bladder samples with 3 metadata variables (batch, outcome and cancer). It contains 5 batches, 3 cancer types (cancer, biopsy, control), and 5 outcomes (Biopsy, mTCC, sTCC-CIS, sTCC+CIS, and Normal). Batch 1 contains only cancer, 2 has cancer and controls, 3 has only controls, 4 contains only biopsy, and 5 contains cancer and biopsy

Usage

```
bladder_data_upload()
```

Value

a SE object with counts data and metadata

Examples

```
library(bladderbatch)
se_object <- bladder_data_upload()
```

BMI_data *This function returns BMI data that comes form the data in "Comparing tuberculosis gene signatures in malnourished individuals using the TBSignatureProfiler" paper. Subject IDs were matched as shown on "github.com/jessmcc22/BatchQCv2_Manuscript/blob/devel/R/subjectID_match.R"*

Description

This function returns BMI data that comes form the data in "Comparing tuberculosis gene signatures in malnourished individuals using the TBSignatureProfiler" paper. Subject IDs were matched as shown on "github.com/jessmcc22/BatchQCv2_Manuscript/blob/devel/R/subjectID_match.R"

Usage

```
BMI_data(meta)
```

Arguments

meta	dataframe; metadata that needs to be matched to BMI

Value

dataframe provided as input with BMI info added

---

check_valid_input *Helper function to save variables as factors if not already factors*

Description

Helper function to save variables as factors if not already factors

Usage

```
check_valid_input(se, batch, condition)
```

Arguments

se	se object
batch	batch
condition	condition

Value

se se object

---

color_palette *Color palette*

Description

This function creates the base color palette used in BatchQC

Usage

```
color_palette(n, first_hue = 25, last_hue = 360)
```

Arguments

n	numeric object representing number of colors to be created
first_hue	numeric object to set the first hue value
last_hue	numeric object to set the final hue value

Value

color_list list of colors generated

Examples

```
library(scran)
n <- 100
color_list <- color_palette(n)
color_list
```

`ComBat_correction` *ComBat Correction This function applies ComBat correction to your summarized experiment object*

Description

ComBat Correction This function applies ComBat correction to your summarized experiment object

Usage

```
ComBat_correction(se, assay_to_normalize, batch, covar, output_assay_name)
```

Arguments

`se`	SummarizedExperiment object
`assay_to_normalize`	Assay that should be corrected
`batch`	The variable that represents batch
`covar`	list of covariates
`output_assay_name`	name of results assay

Value

SE object with an added ComBat corrected array

`ComBat_seq_correction` *ComBat-Seq Correction This function applies ComBat-seq correction to your summarized experiment object*

Description

ComBat-Seq Correction This function applies ComBat-seq correction to your summarized experiment object

Usage

```
ComBat_seq_correction(se, assay_to_normalize, batch, group, covar,
output_assay_name)
```

Arguments

se	SummarizedExperiment object
assay_to_normalize	Assay that should be corrected
batch	The variable that represents batch
group	The group variable
covar	list of covariates
output_assay_name	name of results assay

Value

SE object with an added ComBat-seq corrected array

---

commentary *This function creates the commentary recommendation when there are more than 20 samples.*

Description

This function creates the commentary recommendation when there are more than 20 samples.

Usage

```
commentary(
  nb_fit,
  nb_fit_pval,
  count_below_value,
  count_below_value_pval,
  low_pval
)
```

Arguments

nb_fit	Boolean representing if the count is below the threshold
nb_fit_pval	Boolean representing if the p-val count is below threshold
count_below_value	number of features below threshold
count_below_value_pval	number of features below p-val threshold
low_pval	pval threshold

Value

a commentary string statement

`compute_aic` *Compute the AIC for lognormal (ComBat) model, negative binomial (ComBat-seq) model and the Voom model*

Description

nb_result A vector contains the AIC based on negative binomial model for individual genes.

lognormal_result A vector contains the AIC based on lognormal model for individual genes.

voom_result A vector contains the AIC based on voom transformation for individual genes.

total_AIC The sum of AICs across all genes for the three models in comparison.

min_AIC The number of minimum AIC across the three models in comparison for individual genes.

Usage

```
compute_aic(
  se,
  assay_of_interest,
  batchind,
  groupind,
  maxit = 25,
  zero_filt_percent = 100
)
```

Arguments

`se`	SummarizedExperiment object
`assay_of_interest`	The assay name from se that you are interested in analyzing. This assay need to be a counts assay containing only non-negative integers.
`batchind`	Factor or numeric vector of length = ncol(dat); batch indicator for each sample.
`groupind`	Factor or numeric vector of length = ncol(dat); biological group label/indicator for each sample.
`maxit`	Integer giving the maximal number of IWLS iterations. Default is 25.
`zero_filt_percent`	Numeric value between 0 and 100, the percentage of zeros allowed for each gene to be included in the AIC calculation. Genes with more than this percentage of zeros will be filtered out. Default is 100.

Details

This function calculates the AIC based on lognormal distribution, negative-binomial distribution as well as the Voom transformation. It then compares the AICs of the three models across different genes.

Value

A list with the following two elements:

total_AIC The sum of AICs across all genes for the three models in comparison.

min_AIC The number of minimum AIC across the three models in comparison for individual genes.

Examples

```
library(scran)
se <- mockSCE()
compare_aic <- compute_aic(se, assay_of_interest = "counts",
                           batchind = "Cell_Cycle",
                           groupind = c("Treatment", "Mutation_Status"))
print(compare_aic["total_AIC"])
print(compare_aic["min_AIC"])
```

`compute_lambda` *Compute the lambda index for determining a need for batch correction*

Description

This function calculates the proportions of variation explained by batch, group, and residual for each gene using two-way ANOVA and computes the lambda index based on these three proportions.

Usage

```
compute_lambda(dat, batchind, groupind)
```

Arguments

`dat`	Numeric matrix of dimension (genes x samples) where each row represents one gene's expression across samples.
`batchind`	Factor or numeric vector of length = ncol(dat); batch indicator for each sample
`groupind`	Factor or numeric vector of length = ncol(dat); biological group label/indicator for each sample.

Value

dataframe with columns:

BatchV Proportion of total variance explained by batch effects.

GroupV Proportion of total variance explained by group effects.

ResidV Proportion of total variance that is residual noise.

lambda_raw Raw lambda index = total SS_batch / total SS_group.

lambda_adj Adjusted lambda = lambda_raw * ResidV/(1-ResidV).

Examples

```
library(scran)
se <- mockSCE()
res <- BatchQC::compute_lambda(assays(se)[["counts"]],
  colData(se)$Mutation_Status,
  colData(se)$Treatment)
print(res)
```

`confound_metrics` *Combine std. Pearson correlation coefficient and Cramer's V*

Description

Combine std. Pearson correlation coefficient and Cramer's V

Usage

```
confound_metrics(se, batch)
```

Arguments

se	summarized experiment
batch	batch variable

Value

metrics of confounding

Examples

```
library(scran)
se <- mockSCE()
confound_table <- BatchQC::confound_metrics(se, batch = "Mutation_Status")
confound_table
```

`cor_props` *This function allows you to calculate correlation properties*

Description

This function allows you to calculate correlation properties

Usage

```
cor_props(bd)
```

Arguments

bd	batch design

Value

correlation properties

Examples

```
library(scran)
se <- mockSCE()
batch_design_tibble <- batch_design(se, batch = "Mutation_Status",
                                                    covariate = "Treatment")
correlation_property <- BatchQC::cor_props(batch_design_tibble)
correlation_property
```

`counts2pvalue` *This function calculates p-values for each gene given counts, estimated NB size, and estimated NB mean*

Description

This function calculates p-values for each gene given counts, estimated NB size, and estimated NB mean

Usage

```
counts2pvalue(counts, size, mu)
```

Arguments

`counts`	a vector of gene expression values (in counts)
`size`	an estimated size parameter of the NB distributions for the gene
`mu`	a vector of estimated mu parameter of the NB distributions for different samples of the gene

Value

a p-value based on estimated NB size and mean

`covariates_not_confounded` *Returns list of covariates not confounded by batch; helper function for explained variation and for populating shiny app condition options*

Description

Returns list of covariates not confounded by batch; helper function for explained variation and for populating shiny app condition options

Usage

```
covariates_not_confounded(se, batch)
```

Arguments

se	Summarized experiment object
batch	Batch variable

Value

List of explained variation by batch and condition

Examples

```
library(scran)
se <- mockSCE()
covariates_not_confounded <- BatchQC::covariates_not_confounded(se,
                                                batch = "Mutation_Status")
covariates_not_confounded
```

---

cramers_v *This function allows you to calculate Cramer's V*

Description

This function allows you to calculate Cramer's V

Usage

```
cramers_v(bd)
```

Arguments

bd	batch design

Value

Cramer's V

Examples

```
library(scran)
se <- mockSCE()
batch_design_tibble <- batch_design(se, batch = "Mutation_Status",
                                                    covariate = "Treatment")
cramers_v_result <- BatchQC::cramers_v(batch_design_tibble)
cramers_v_result
```

dendrogram_alpha_numeric_check *Dendrogram alpha or numeric checker*

Description

This function checks if there is any numeric or strings for plotting legend

Usage

```
dendrogram_alpha_numeric_check(dendro_var)
```

Arguments

dendro_var	column from dendrogram object representing category

Value

geom_label label for the legend of category variable

Examples

```
library(scran)
se <- mockSCE()
dendro_alpha_numeric_check <- dendrogram_alpha_numeric_check(
                                        dendro_var = "Treatment")
dendro_alpha_numeric_check
```

dendrogram_color_palette *Dendrogram color palette*

Description

This function creates the color palette used in the dendrogram plotter

Usage

```
dendrogram_color_palette(col, dendrogram_info)
```

Arguments

col	string object representing color of the label
dendrogram_info	dendrogram_ends object

Value

annotation_color vector of colors corresponding to col variable

Examples

```
library(scran)
se <- mockSCE()
process_dendro <- BatchQC::process_dendrogram(se, "counts")
dendrogram_ends <- process_dendro$dendrogram_ends
col <- process_dendro$condition_var
dendro_colors <- dendrogram_color_palette(col = "Treatment",
                                    dendrogram_info = dendrogram_ends)
dendro_colors
```

`dendrogram_plotter` *Dendrogram Plot*

Description

This function creates a dendrogram plot

Usage

```
dendrogram_plotter(se, assay, batch_var, category_var)
```

Arguments

`se`	SummarizedExperiment object
`assay`	assay to plot
`batch_var`	sample metadata column representing batch
`category_var`	sample metadata column representing category of interest

Value

named list of dendrogram plots

dendrogram is a dendrogram ggplot

circular_dendrogram is a circular dendrogram ggplot

Examples

```
library(scran)
se <- mockSCE()
dendrogram_plot <- BatchQC::dendrogram_plotter(se,
                                               "counts",
                                               "Mutation_Status",
                                               "Treatment")
dendrogram_plot$dendrogram
dendrogram_plot$circular_dendrogram
```

`DESeq2_small_size` *This function calculated the goodness of fit of DESeq2 for small sample sizes (intended for less than 20 samples).*

Description

This function calculated the goodness of fit of DESeq2 for small sample sizes (intended for less than 20 samples).

Usage

```
DESeq2_small_size(
  count_matrix,
  condition,
  other_variables,
  conditions_df,
  formula_for_DESeq,
  num_samples
)
```

Arguments

`count_matrix`	matrix containing the data to be analyzed
`condition`	a vector containing a factor of the condition of interest (typically batch)
`other_variables`	a vector of strings of other variables of interest
`conditions_df`	data frame containing information for the other variables of interest (columns in order of the other_variables vector)
`formula_for_DESeq`	the stat formula to be used in the DESeq analysis
`num_samples`	total number of samples to analyze

Value

a list containing the string recommendation, the histogram and a reference for the original source of the test

`DESeq_large_analysis` *This function calculated the goodness of fit of DESeq2 for larger sample sizes (intended for more than 20 samples).*

Description

This function calculated the goodness of fit of DESeq2 for larger sample sizes (intended for more than 20 samples).

Usage

```
DESeq_large_analysis(
  count_matrix,
  condition,
  other_variables,
  conditions_df,
  formula_for_DESeq,
  num_samples,
  sampled
)
```

Arguments

count_matrix	matrix containing the data to be analyzed
condition	a vector containing a factor of the condition of interest (typically batch)
other_variables	a vector of strings of other variables of interest
conditions_df	data frame containing information for the other variables of interest (columns in order of the other_variables vector)
formula_for_DESeq	the stat formula to be used in the DESeq analysis
num_samples	total number of samples to analyze
sampled	the down sampled matrix

Value

a list containing the string recommendation

DE_analyze *Differential Expression Analysis*

Description

This function runs DE analysis on a count matrix (DESeq), a normalized log or log-CPM matrix (limma), an edgeR TMM-normalized matrix (edgeR) or perform ANOVA or Kruskal-Wallis test on the data contained in the se object.

Usage

```
DE_analyze(se, method, batch, conditions, assay_to_analyze, padj_method)
```

Arguments

se	SummarizedExperiment object
method	DE analysis method option ('DESeq2', 'limma', 'edgeR', 'ANOVA', or 'Kruskal-Wallis')
batch	metadata column in the se object representing batch
conditions	metadata columns in the se object representing additional analysis covariates

assay_to_analyze
: Assay in the se object (either counts for DESeq2 or normalized data for limma or edgeR) for DE analysis

padj_method
: correction method for adjusted p-value from p.adjust.methods

Value

A named list containing the log2FoldChange, fvalue (ANOVA only), pvalue and adjusted pvalue (padj) for each analysis returned by DESeq2, limma, edgeR, ANOVA, or Kruskal-Wallis.

Examples

```
library(scran)
se <- mockSCE()
differential_expression <- BatchQC::DE_analyze(se = se,
                                               method = "DESeq2",
                                               batch = "Treatment",
                                               conditions = c(
                                               "Mutation_Status"),
                                               assay_to_analyze = "counts",
                                               padj_method = "BH")
pval_summary(differential_expression)
pval_plotter(differential_expression)
```

EV_plotter *This function allows you to plot explained variation*

Description

This function allows you to plot explained variation

Usage

```
EV_plotter(batchqc_ev)
```

Arguments

batchqc_ev
: table of explained variation from batchqc_explained_variation

Value

boxplot of explained variation

Examples

```
library(scran)
se <- mockSCE()
se$Mutation_Status <- as.factor(se$Mutation_Status)
se$Treatment <- as.factor(se$Treatment)
expl_var_result <- batchqc_explained_variation(se, batch = "Mutation_Status",
                            condition = "Treatment", assay_name = "counts")
EV_boxplot <- BatchQC::EV_plotter(expl_var_result[[1]])
EV_boxplot
```

EV_table *EV Table Returns table with percent variation explained for specified number of genes*

Description

EV Table Returns table with percent variation explained for specified number of genes

Usage

```
EV_table(batchqc_ev)
```

Arguments

batchqc_ev	explained variation results from batchqc_explained_variation

Value

List of explained variation by batch and condition

Examples

```
library(scran)
se <- mockSCE()
se$Mutation_Status <- as.factor(se$Mutation_Status)
se$Treatment <- as.factor(se$Treatment)
exp_var_result <- BatchQC::batchqc_explained_variation(se,
                                    batch = "Mutation_Status",
                                    condition = "Treatment",
                                    assay_name = "counts")
EV_table <- BatchQC::EV_table(exp_var_result[[1]])

EV_table
```

get.res *Helper function to get residuals*

Description

Helper function to get residuals

Usage

```
get.res(y, X)
```

Arguments

y	assay
X	model matrix design

Value

residuals

`goodness_of_fit_DESeq2` *This function calculates goodness-of-fit pvalues for all genes by looking at how the NB model by DESeq2 fit the data*

Description

This function calculates goodness-of-fit pvalues for all genes by looking at how the NB model by DESeq2 fit the data

Usage

```
goodness_of_fit_DESeq2(
  se,
  count_matrix,
  condition,
  other_variables = NULL,
  num_genes = 500
)
```

Arguments

`se`	the se object where all the data is contained
`count_matrix`	name of the assay with gene expression matrix (in counts)
`condition`	name of the se colData with the condition status
`other_variables`	name of the se colData containing other variables of interest that should be considered in the DESeq2 model
`num_genes`	downsample value, default is 500 (or all genes if less)

Value

a matrix of pvalues where each row is a gene and each column is a level within the condition of interest

Examples

```
# example code
library(scran)
se <- mockSCE(ncells = 20)
se$Treatment <- as.factor(se$Treatment)
se$Mutation_Status <- as.factor(se$Mutation_Status)
nb_results <- goodness_of_fit_DESeq2(se = se, count_matrix = "counts",
  condition = "Treatment", other_variables = "Mutation_Status")
nb_results[1]
nb_results[2]
nb_results[3]
```

heatmap_num_to_char_converter *Heatmap numeric to character converter*

Description

This function converts any found numerics to characters

Usage

```
heatmap_num_to_char_converter(ann_col)
```

Arguments

ann_col	column data of heatmap

Value

ann_col modified column data of heatmap

Examples

```
library(scran)
se <- mockSCE()
col_info <- colData(se)
ann_col <- heatmap_num_to_char_converter(ann_col = col_info)
ann_col
```

heatmap_plotter *Heatmap Plotter*

Description

This function allows you to plot a heatmap

Usage

```
heatmap_plotter(se, assay, nfeature, annotation_column, log_option)
```

Arguments

se	SummarizedExperiment
assay	normalized or corrected assay
nfeature	number of features to display
annotation_column	choose column
log_option	TRUE if data should be logged before plotting (recommended for sequencing counts), FALSE if data should not be logged (for instance, data is already logged)

Value

heatmap plot

Examples

```
library(scran)
se <- mockSCE()
heatmaps <- BatchQC::heatmap_plotter(se,
                                assay = "counts",
                                nfeature = 15,
                                annotation_column = c("Mutation_Status",
                                "Treatment"), log_option = FALSE)
correlation_heatmap <- heatmaps$correlation_heatmap
correlation_heatmap

heatmap <- heatmaps$topn_heatmap
heatmap
```

---

is_design_balanced *Check if the experimental design is balanced or unbalanced*

Description

Used in conjunction with the lambda

Usage

```
is_design_balanced(se, batch, covariate)
```

Arguments

se	summarized experiment object
batch	string, batch variable
covariate	string, biological covariate

Value

Boolean Value, TRUE if the experimental design is balanced, FALSE if the experimental design is not balanced

Examples

```
library(scran)
se <- mockSCE()
balanced_design_check <- is_design_balanced(se, batch = "Mutation_Status",
                                            covariate = "Treatment")
balanced_design_check
```

kBET *kBET - k-nearest neighbour batch effect test*

Description

adapted from kBET package (https://github.com/theislab/kBET). kBET runs a chi square test to evaluate the probability of a batch effect.

Usage

```
kBET(
  df,
  batch,
  k0 = NULL,
  knn = NULL,
  testSize = NULL,
  do.pca = TRUE,
  dim.pca = 50,
  heuristic = TRUE,
  n_repeat = 100,
  alpha = 0.05,
  addTest = FALSE,
  verbose = FALSE,
  plot = TRUE,
  adapt = TRUE
)
```

Arguments

df	dataset (rows: cells, columns: features)
batch	batch id for each cell or a data frame with both condition and replicates
k0	number of nearest neighbours to test on (neighbourhood size)
knn	an n x k matrix of nearest neighbours for each cell (optional)
testSize	number of data points to test, (10 percent sample size default, but at least 25)
do.pca	perform a pca prior to knn search? (defaults to TRUE)
dim.pca	if do.pca=TRUE, choose the number of dimensions to consider (defaults to 50)
heuristic	compute an optimal neighbourhood size k (defaults to TRUE)
n_repeat	to create a statistics on batch estimates, evaluate 'n_repeat' subsets
alpha	significance level
addTest	perform an LRT-approximation to the multinomial test AND a multinomial exact test (if appropriate)
verbose	displays stages of current computation (defaults to FALSE)
plot	if stats > 10, then a boxplot of the resulting rejection rates is created
adapt	In some cases, a number of cells do not contribute to any neighbourhood and this may cause an imbalance in observed and expected batch label frequencies. Frequencies will be adapted if adapt=TRUE (default).

Value

list object

1. summary - a rejection rate for the data, an expected rejection rate for random labeling and the significance for the observed result
2. results - detailed list for each tested cells; p-values for expected and observed label distribution
3. average.pval - significance level over the averaged batch label distribution in all neighbourhoods
4. stats - extended test summary for every sample
5. params - list of input parameters and adapted parameters, respectively
6. outsider - only shown if adapt=TRUE. List of samples without mutual nearest neighbour:
   - index - index of each outsider sample)
   - categories - tabularised labels of outsiders
   - p.val - Significance level of outsider batch label distribution vs expected frequencies. If the significance level is lower than alpha, expected frequencies will be adapted

If the optimal neighbourhood size (k0) is smaller than 10, NA is returned.

Examples

```
library(scran)
se <- mockSCE()
df <- as.matrix(assays(se)[["counts"]])
batch <- data.frame(colData(se))[, "Treatment"]

batch.estimate <- kBET(df, batch)
```

---

limma_correction *Limma Correction This function applies limma batch correction to your provided assay*

Description

Limma Correction This function applies limma batch correction to your provided assay

Usage

```
limma_correction(se, assay_to_normalize, batch, covar, output_assay_name)
```

Arguments

se	SummarizedExperiment object
assay_to_normalize	Log assay that should be corrected
batch	Factor containing batch information
covar	list of covariates
output_assay_name	name of results assay

Value

SE object with an added limma corrected array

`merged_IDs` *BMI and matched sample names for TB data*

Description

This is support data for the TB data set that contains the BMI data and ID numbers from both the curatedTBData database and the original study the data was used in

Usage

```
data(merged_IDs)
```

Format

A data frame with 91 rows and 3 columns

subjectID_curatedTBData Subject ID found in curatedTBData

subjectID_TB_Paper Subject ID in the original paper

BMI subject's BMI from the original study

`nb_histogram` *This function creates a histogram from the negative binomial goodness-of-fit adjusted pvalues.*

Description

This function creates a histogram from the negative binomial goodness-of-fit adjusted pvalues.

Usage

```
nb_histogram(adj_p_val_table)
```

Arguments

`adj_p_val_table`
: table of adjusted p-values from the nb test

Value

a histogram of the number of genes within a p-value range

`nb_proportion` *This function determines the proportion of p-values below a specific value and compares to the previously determined threshold of 0.42 for extreme low values.*

Description

This function determines the proportion of p-values below a specific value and compares to the previously determined threshold of 0.42 for extreme low values.

Usage

```
nb_proportion(
  adj_p_val_table,
  p_val_table,
  low_pval = 0.01,
  threshold = 0.42,
  num_samples
)
```

Arguments

`adj_p_val_table`	table of adjusted p-values from the nb test
`p_val_table`	table of p-values from the nb test
`low_pval`	value of the p-value cut off to use in proportion
`threshold`	the value to compare the proportion of p-values to for data sets less than 20, default is 0.42
`num_samples`	the number of samples in the analysis

Value

a statement about whether DESeq2 is appropriate to use for analysis

`normalize_SE` *This function allows you to add normalized count matrix to the SE object*

Description

This function allows you to add normalized count matrix to the SE object

Usage

```
normalize_SE(
  se,
  method,
  log_bool,
  assay_to_normalize,
  output_assay_name,
  condition = NULL,
  batch = NULL
)
```

Arguments

se	SummarizedExperiment Object
method	string; Normalization Method, either 'CPM', 'DESeq', 'edgeR', 'voom', or 'none' for log(x+1) only
log_bool	True or False; True to log normalize the data set after normalization method
assay_to_normalize	string; SE assay to do normalization on
output_assay_name	string; name for the resulting normalized assay
condition	string; the biological variable of interest, required for voom, default 'NULL'
batch	string; the batch variable, required for voom, default 'NULL'

Value

the original SE object with normalized assay appended

Examples

```
library(scran)
se <- mockSCE()
se_CPM_normalized <- BatchQC::normalize_SE(se, method = "CPM",
                                log_bool = FALSE,
                                assay_to_normalize = "counts",
                                output_assay_name =
                                    "CPM_normalized_counts")
se_DESeq_normalized <- BatchQC::normalize_SE(se, method = "DESeq",
                                log_bool = FALSE,
                                assay_to_normalize = "counts",
                                output_assay_name =
                                    "DESeq_normalized_counts")
se_CPM_normalized
se_DESeq_normalized
```

PCA_plotter *This function allows you to plot PCA*

Description

This function allows you to plot PCA

Usage

```
PCA_plotter(se, nfeature, color, shape, batch, assays, xaxisPC,
yaxisPC, log_option = FALSE)
```

Arguments

se	SummarizedExperiment object
nfeature	number of features
color	choose a color
shape	choose a shape
batch	variable representing batch (for ellipses)
assays	array of assay names from se
xaxisPC	the PC to plot as the x axis
yaxisPC	the PC to plot as the y axis
log_option	TRUE if data should be logged before plotting (recommended for sequencing counts), FALSE if data should not be logged (for instance, data is already logged); FALSE by default

Value

List containing PCA info, PCA variance and PCA plot

Examples

```
library(scran)
se <- mockSCE()
se_object_ComBat_Seq <- BatchQC::batch_correct(se, method = "ComBat-Seq",
                                          assay_to_normalize = "counts",
                                          batch = "Mutation_Status",
                                          covar = "Treatment",
                                          output_assay_name =
                                              "ComBat_Seq_Corrected")
pca_plot <- BatchQC::PCA_plotter(se = se_object_ComBat_Seq,
                            nfeature = 2, color = "Mutation_Status",
                            shape = "Treatment", batch = "batch",
                            assays = c("counts", "ComBat_Seq_Corrected"),
                            xaxisPC = 1, yaxisPC = 2, log_option = FALSE)
pca_plot$plot
pca_plot$var_explained
```

`permuted_DESeq` *This function performs DESeq on the permuted dataset adjusted pvalues.*

Description

This function performs DESeq on the permuted dataset adjusted pvalues.

Usage

```
permuted_DESeq(
  count_matrix,
  condition,
  other_variables,
  conditions_df,
  formula_for_DESeq
)
```

Arguments

`count_matrix`	matrix containing the data to be analyzed
`condition`	a vector containing a factor of the condition of interest (typically batch)
`other_variables`	a vector of strings of other variables of interest
`conditions_df`	data frame containing information for the other variables of interest (columns in order of the other_variables vector)
`formula_for_DESeq`	the stat formula to be used in the DESeq analysis

Value

a DESeq2 object

`plot_data` *This function formats the PCA plot using ggplot*

Description

This function formats the PCA plot using ggplot

Usage

```
plot_data(pca_plot_data, color, shape, batch, xaxisPC, yaxisPC)
```

Arguments

pca_plot_data	Data for all assays to plot
color	variable that will be plotted as color
shape	variable that will be plotted as shape
batch	variable representing batch for the ellipses
xaxisPC	the PC to plot as the x axis
yaxisPC	the PC to plot as the y axis

Value

PCA plot

---

plot_kBET *kBET Rejection Plotter*

Description

This function generates a boxplot of observed and expected rejection rates for the provided kBET output list object

Usage

```
plot_kBET(kBET_res)
```

Arguments

kBET_res	list object output from kBET function

Value

ggplot object containing kBET rejection boxplot

Examples

```
library(scran)
se <- mockSCE()
df <- as.matrix(assays(se)[["counts"]])
batch <- data.frame(colData(se))[, "Treatment"]

batch.estimate <- kBET(df, batch)
plot_kBET(batch.estimate)
```

`possible_distances` *Create potential min_distance values for exploratory analysis based on the value of spread*

Description

Create potential min_distance values for exploratory analysis based on the value of spread

Usage

```
possible_distances(spread)
```

Arguments

`spread`	numeric; the value of spread used in the exploratory analysis

Value

vector of min_distance values to use in exploratory analysis

`possible_k_neighbors` *Create a vector of possible nearest neighbor values from 5, 15, 25, 50, and 100*

Description

Create a vector of possible nearest neighbor values from 5, 15, 25, 50, and 100

Usage

```
possible_k_neighbors(data_size)
```

Arguments

`data_size`	size of the data set used to create umaps

Value

k nearest neighbor list

preprocess *Preprocess assay data*

Description

Preprocess assay data

Usage

```
preprocess(se, assay, nfeature, log_option)
```

Arguments

se	Summarized Experiment object
assay	Assay from SummarizedExperiment object
nfeature	Number of variable features to use
log_option	"True" if data should be logged, "False" otherwise

Value

Returns processed data

process_dendrogram *Process Dendrogram*

Description

This function processes count data for dendrogram plotting

Usage

```
process_dendrogram(se, assay)
```

Arguments

se	SummarizedExperiment object
assay	assay to plot

Value

named list of dendrogram data

dendrogram_segments is data representing segments of the dendrogram

dendrogram_ends is data representing ends of the dendrogram

Examples

```
library(scran)
se <- mockSCE()
process_dendro <- BatchQC::process_dendrogram(se, "counts")
process_dendro
```

protein_data *Protein data with 39 protein expression levels*

Description

This data consists of two batches and two conditions corresponding to case and control. The columns are case/control samples, and the rows represent 39 different proteins.

Usage

```
data(protein_data)
```

Format

A data frame with 39 rows and 24 variables

protein_sample_info *Batch and Condition indicator for protein expression data*

Description

This data consists of two batches and two conditions corresponding to case and control for the protein expression data

Usage

```
data(protein_sample_info)
```

Format

A data frame with 24 rows and 2 variables:

batch Batch Indicator

category Condition (Case vs Control) Indicator

pval_plotter *P-value Plotter This function allows you to plot p-values of explained variation*

Description

P-value Plotter This function allows you to plot p-values of explained variation

Usage

```
pval_plotter(DE_results)
```

Arguments

DE_results	Differential Expression analysis result (a named list of dataframes corresponding to each analysis completed with a "pvalue" column)

Value

boxplots of pvalues for each condition

Examples

```
library(scran)
se <- mockSCE()
differential_expression <- BatchQC::DE_analyze(se = se,
                                               method = "DESeq2",
                                               batch = "Treatment",
                                               conditions = c(
                                               "Mutation_Status"),
                                               assay_to_analyze = "counts",
                                               padj_method = "BH")
pval_summary(differential_expression)
pval_plotter(differential_expression)
```

pval_summary *Returns summary table for p-values of explained variation*

Description

Returns summary table for p-values of explained variation

Usage

```
pval_summary(res_list)
```

Arguments

res_list	Differential Expression analysis result (a named list of dataframes corresponding to each analysis completed with a "pvalue" column)

Value

summary table for p-values of explained variation for each analysis

Examples

```
library(scran)
se <- mockSCE()
differential_expression <- BatchQC::DE_analyze(se = se,
                                               method = "DESeq2",
                                               batch = "Treatment",
                                               conditions = c(
                                               "Mutation_Status"),
                                               assay_to_analyze = "counts",
                                               padj_method = "BH")
pval_summary(differential_expression)
```

`ratio_plotter` *This function allows you to plot ratios of explained variation*

Description

This function allows you to plot ratios of explained variation

Usage

```
ratio_plotter(ev_ratio)
```

Arguments

`ev_ratio`	table of ratios from variation_ratios()

Value

boxplot of ratios

Examples

```
library(scran)
se <- mockSCE()
se$Mutation_Status <- as.factor(se$Mutation_Status)
se$Treatment <- as.factor(se$Treatment)
expl_var_result <- batchqc_explained_variation(se, batch = "Mutation_Status",
                            condition = "Treatment", assay_name = "counts")
ratios_results <- variation_ratios(expl_var_result[[1]],
    batch = "Mutation_Status")
ratio_boxplot <- BatchQC::ratio_plotter(ratios_results)
ratio_boxplot
```

run_kBET *kBET rejection rate*

Description

This function runs the k-nearest neighbor batch effect test (kBET) to evaluate whether the data has detectable batch effect.

Usage

```
run_kBET(
  se,
  assay_to_normalize,
  batch,
  k0 = NULL,
  knn = NULL,
  testSize = NULL,
  do.pca = TRUE,
  dim.pca = 50,
  heuristic = TRUE,
  n_repeat = 100,
  alpha = 0.05,
  addTest = FALSE,
  verbose = FALSE,
  adapt = TRUE
)
```

Arguments

se	SummarizedExperiment object
assay_to_normalize	string; assay from se object to do normalization
batch	character string of column name that represents batch
k0	integer representing number of nearest neighbors to test on (neighborhood size)
knn	n x k matrix of nearest neighbors for each cell (optional)
testSize	integer representing number of data points to test
do.pca	Boolean, if TRUE, perform a pca prior to knn search (defaults to TRUE)
dim.pca	Boolean, if do.pca=TRUE, choose the number of dimensions to consider (defaults to 50)
heuristic	Boolean, if true, compute an optimal neighborhood size k (defaults to TRUE)
n_repeat	numeric representing 'n_repeat' subsets to evaluate in order to create a statistics on batch estimates
alpha	numeric for significance level
addTest	Boolean, if TRUE, perform an LRT-approximation to the multinomial test AND a multinomial exact test (if appropriate)
verbose	Boolean, if TRUE, display stages of current computation (defaults to FALSE)
adapt	Boolean, if TRUE, frequencies will be adapted (defaults to TRUE)

Value

list object from kBET() function

1. `summary` - a rejection rate for the data, an expected rejection rate for random labeling and the significance for the observed result
2. `results` - detailed list for each tested cells; p-values for expected and observed label distribution
3. `average.pval` - significance level over the averaged batch label distribution in all neighbourhoods
4. `stats` - extended test summary for every sample
5. `params` - list of input parameters and adapted parameters, respectively
6. `outsider` - only shown if `adapt=TRUE`. List of samples without mutual nearest neighbour:
   - `index` - index of each outsider sample)
   - `categories` - tabularised labels of outsiders
   - `p.val` - Significance level of outsider batch label distribution vs expected frequencies. If the significance level is lower than `alpha`, expected frequencies will be adapted

Examples

```
library(scran)
se <- mockSCE()
kBET_result <- BatchQC::run_kBET(
  se=se,
  assay_to_normalize="counts",
  batch="Treatment"
)

BatchQC::plot_kBET(kBET_result)
```

---

run_lambda *Provide a recommendation on batch correction based on lambda calculation*

Description

This functions determines if an experimental design is balanced, then calculates the lambda statistic for balanced designs and provides a recommendation on if batch correction should be utilized. In general, unbalanced designs always benefit from batch correction, while balanced designs with a lambda greater than -2 benefit from batch correction.

Usage

```
run_lambda(se, assay, batch, condition)
```

Arguments

`se`	summarized experiment object
`assay`	string, the assay to analyze
`batch`	string, batch variable
`condition`	string, condition variable

Value

a named list with:

lambda_stat provides the output of `compute_lambda` function

correction_recommendation string, rec for batch correction

a list with 2 parameters, 'lambda_stat' which contains the adj lambda value from lambda_compute (ln(lambda)) or 'NULL' if the design is balanced, and 'correction_recommendation' which contains a string with a recommendation on if batch correction should be completed

Examples

```
library(scran)
se <- mockSCE()
lambda_calculation <- run_lambda(se,
                                 assay = "counts",
                                 batch = "Mutation_Status",
                                 condition = "Treatment")
 print(lambda_calculation$correction_recommendation)
 print(lambda_calculation$lambda_stat)
```

---

signature_data *Signature data with 1600 gene expression levels*

Description

This data consists of three batches and ten conditions. The columns are samples, and the rows represent 1600 different genes.

Usage

```
data(signature_data)
```

Format

A data frame with 1600 rows and 89 variables

---

std_pearson_corr_coef *Calculate a standardized Pearson correlation coefficient*

Description

Calculate a standardized Pearson correlation coefficient

Usage

```
std_pearson_corr_coef(bd)
```

Arguments

bd	batch design

Value

standardized Pearson correlation coefficient

Examples

```
library(scran)
se <- mockSCE()
batch_design_tibble <- batch_design(se, batch = "Mutation_Status",
                                                covariate = "Treatment")
pearson_cor_result <- BatchQC::std_pearson_corr_coef(batch_design_tibble)
pearson_cor_result
```

---

summarized_experiment *This function creates a summarized experiment object from count and metadata files uploaded by the user*

Description

This function creates a summarized experiment object from count and metadata files uploaded by the user

Usage

```
summarized_experiment(counts, columndata)
```

Arguments

counts	counts matrix
columndata	metadata dataframe

Value

a summarized experiment object

Examples

```
data(protein_data)
data(protein_sample_info)
se_object <- summarized_experiment(protein_data, protein_sample_info)
```

`svaseq_correction` *svaseq Correction This function applies sva correction to a summarized experiment object with count based RNA-seq data*

Description

svaseq Correction This function applies sva correction to a summarized experiment object with count based RNA-seq data

Usage

```
svaseq_correction(
  se,
  assay_to_normalize,
  var_of_interest,
  covar,
  output_assay_name,
  num_sv = FALSE
)
```

Arguments

`se`	SummarizedExperiment object
`assay_to_normalize`	string; name of assay that should be corrected
`var_of_interest`	string; name of experimental variable of interest
`covar`	list; sting list of covariates to include in sva analysis
`output_assay_name`	string; name of results assay
`num_sv`	boolean; Default is FALSE: the number of estimated latent factor is set to 1 for a small number of samples. If set to TRUE, svaseq function will estimate the number of latent factors for you.

Value

SE object with an added sva corrected array

`sva_correction` *sva Correction This function applies sva correction to a summarized experiment object (implementation adapted from sva::psva)*

Description

sva Correction This function applies sva correction to a summarized experiment object (implementation adapted from sva::psva)

Usage

```
sva_correction(
  se,
  assay_to_normalize,
  var_of_interest,
  covar,
  output_assay_name,
  psva = FALSE
)
```

Arguments

se	SummarizedExperiment object
assay_to_normalize	string; name of assay that should be corrected
var_of_interest	string; name of experimental variable of interest
covar	list; sting list of covariates to include in sva analysis
output_assay_name	string; name of results assay
psva	boolean; default: FALSE. If set to TRUE and no covariate input, psva function from the sva package will be used to remove batch effect.

Value

SE object with an added sva corrected array

---

tb_data_upload *TB data upload This function uploads the TB data set from the curatedTBData package.*

Description

TB data upload This function uploads the TB data set from the curatedTBData package.

Usage

```
tb_data_upload()
```

Value

a SE object with raw counts data and metadata

Examples

```
library(curatedTBData)
se_object <- tb_data_upload()
```

umap *Create a umap plot; wrapper function for umap package plus custom plotting*

Description

Create a umap plot; wrapper function for umap package plus custom plotting

Usage

```
umap(
  se_object,
  assay_of_interest,
  batch,
  covar,
  neighbors = 15,
  min_distance = 0.1,
  spread = 1,
  exploratory = FALSE
)
```

Arguments

se_object	se_object; containing data of interest
assay_of_interest	string; the assay in the se_object to plot
batch	string; representing batch
covar	string; representing biological variable
neighbors	integer; number of nearest neighbors, default 15 per umap; lower values prioritize local structure, higher values will represent bigger picture but lose finer details
min_distance	numeric; how close points appear in final layout; higher values puts less emphasis on global structure; must be less than spread
spread	numeric; dispersion of points in umap
exploratory	Boolean; default is FALSE, if True, a 5x5 grid with k = 15, 25, 50, 100 and min_distance = 0.1, .2, .5, .75, .99 will be plotted

Value

umap plot

Examples

```
library(scran)
se <- mockSCE()
se$Treatment <- as.factor(se$Treatment)
se$Mutation_Status <- as.factor(se$Mutation_Status)
umap_plot <- BatchQC::umap(se_object = se, assay_of_interest = "counts",
batch = "Treatment", covar = "Mutation_Status")
umap_plot
```

variation_ratios *Creates Ratios of batch to variable variation statistic*

Description

Creates Ratios of batch to variable variation statistic

Usage

```
variation_ratios(ex_variation_table, batch)
```

Arguments

ex_variation_table	table of explained variation results from batchqc_explained_variation
batch	batch

Value

dataframe with condition/batch ratios

Examples

```
library(scran)
se <- mockSCE()
se$Mutation_Status <- as.factor(se$Mutation_Status)
se$Treatment <- as.factor(se$Treatment)
expl_var_result <- batchqc_explained_variation(se, batch = "Mutation_Status",
                            condition = "Treatment", assay_name = "counts")
ratios_results <- variation_ratios(expl_var_result[[1]],
    batch = "Mutation_Status")
ratios_results
```

volcano_plot *Volcano plot*

Description

This function allows you to plot DE analysis results as a volcano plot

Usage

```
volcano_plot(DE_results, pslider = 0.05, fcslider)
```

Arguments

DE_results	a dataframe with the results of one of the DE Analysis; must include "log2FoldChange" and "pvalue" columns
pslider	Magnitude of significance value threshold, default is 0.05
fcslider	Magnitude of expression change value threshold

Value

A volcano plot of expression change and significance value data

Examples

```
library(scran)
se <- mockSCE()
differential_expression <- BatchQC::DE_analyze(se = se,
                                               method = "DESeq2",
                                               batch = "Treatment",
                                               conditions = c(
                                               "Mutation_Status",
                                               "Cell_Cycle"),
                                               assay_to_analyze = "counts",
                                               padj_method = "BH")
value <- round((max(abs(
    differential_expression[[length(differential_expression)]][, 1]))
    + min(abs(
    differential_expression[[length(differential_expression)]][, 1]))) / 2)

volcano_plot(differential_expression[[1]], pslider = 0.05, fcslider = value)
```

Index