

# Package ‘BatchQC’

April 3, 2025

**Type** Package

**Title** Batch Effects Quality Control Software

**Version** 2.3.5

**Date** 2025-03-06

**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 (>= 4.4.0)

**Imports** data.table, DESeq2, dplyr, EBSeq, gg dendro, ggnewscale, ggplot2, limma, matrixStats, pheatmap, RColorBrewer, reader, reshape2, scan, shiny, shinyjs, shinythemes, stats, SummarizedExperiment, sva, S4Vectors, tibble, tidyr, tidyverse, umap, utils

**Suggests** BiocManager, BiocStyle, bladderbatch, devtools, knitr, lintr, 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** 7.3.2

**git\_url** <https://git.bioconductor.org/packages/BatchQC>

**git\_branch** devel

**git\_last\_commit** fc9ba94

**git\_last\_commit\_date** 2025-03-18

**Repository** Bioconductor 3.21

**Date/Publication** 2025-04-03

**Author** Jessica Anderson [aut, cre] (ORCID:

<https://orcid.org/0000-0002-0542-9872>),

W. Evan Johnson [aut] (ORCID: <https://orcid.org/0000-0002-6247-6595>),

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 [aut],

Solomon Joseph [ctb]

**Maintainer** Jessica Anderson <[anderson.jessica@rutgers.edu](mailto:anderson.jessica@rutgers.edu)>

## Contents

BatchQC . . . . .	4
batchqc_explained_variation . . . . .	4
batch_correct . . . . .	5
batch_design . . . . .	6
batch_indicator . . . . .	7
bladder_data_upload . . . . .	7
check_valid_input . . . . .	8
color_palette . . . . .	8
ComBat_correction . . . . .	9
ComBat_seq_correction . . . . .	10

commentary . . . . .	10
confound_metrics . . . . .	11
cor_props . . . . .	12
counts2pvalue . . . . .	12
covariates_not_confounded . . . . .	13
cramers_v . . . . .	13
dendrogram_alpha_numeric_check . . . . .	14
dendrogram_color_palette . . . . .	15
dendrogram_plotter . . . . .	15
DESeq2_small_size . . . . .	16
DESeq_large_analysis . . . . .	17
DE_analyze . . . . .	18
EV_plotter . . . . .	19
EV_table . . . . .	19
get.res . . . . .	20
goodness_of_fit_DESeq2 . . . . .	21
heatmap_num_to_char_converter . . . . .	22
heatmap_plotter . . . . .	22
limma_correction . . . . .	23
nb_histogram . . . . .	24
nb_proportion . . . . .	25
normalize_SE . . . . .	25
PCA_plotter . . . . .	26
permuted_DESeq . . . . .	27
plot_data . . . . .	28
possible_distances . . . . .	29
possible_k_neighbors . . . . .	29
preprocess . . . . .	30
process_dendrogram . . . . .	30
protein_data . . . . .	31
protein_sample_info . . . . .	31
pval_plotter . . . . .	32
pval_summary . . . . .	32
ratio_plotter . . . . .	33
signature_data . . . . .	34
std_pearson_corr_coef . . . . .	34
summarized_experiment . . . . .	35
sva_correction . . . . .	35
umap . . . . .	36
variation_ratios . . . . .	37
volcano_plot . . . . .	38

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	<i>Batch Correct This function allows you to Add batch corrected count matrix to the SE object</i>
---------------	--

---

**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")
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

**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
batch	batch variable
covariate	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	<i>Batch and Condition indicator for signature data</i>
-----------------	---

---

### 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

---

bladder_data_upload	<i>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</i>
---------------------	--

---

### 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()
```

---

check_valid_input	<i>Helper function to save variables as factors if not already factors</i>
-------------------	--

---

**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	<i>Color palette</i>
---------------	----------------------

---

**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	<i>ComBat Correction This function applies ComBat correction to your summarized experiment object</i>
-------------------	---

---

**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

---

confound_metrics	<i>Combine std. Pearson correlation coefficient and Cramer's V</i>
------------------	--

---

**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	<i>This function calculated the goodness of fit of DESeq2 for small sample sizes (intended for less than 20 samples).</i>
-------------------	---

---

**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  
`sampld` 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) or a normalized log or log-CPM matrix (limma) contained in the se object

**Usage**

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

**Arguments**

`se` SummarizedExperiment object  
`method` DE analysis method option (either 'DESeq2' or 'limma')  
`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) for DE analysis

**Value**

A named list containing the log2FoldChange, pvalue and adjusted pvalue (padj) for each analysis returned by DESeq2 or limma

**Examples**

```

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

pval_summary(differential_expression)
pval_plotter(differential_expression)
  
```

---

EV_plotter	<i>This function allows you to plot explained variation</i>
------------	---

---

**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	<i>EV Table Returns table with percent variation explained for specified number of genes</i>
----------	--

---

**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

<code>se</code>	the se object where all the data is contained
<code>count_matrix</code>	name of the assay with gene expression matrix (in counts)
<code>condition</code>	name of the se colData with the condition status
<code>other_variables</code>	name of the se colData containing other variables of interest that should be considered in the DESeq2 model
<code>num_genes</code>	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
```

---

limma_correction	<i>Limma Correction This function applies limma batch correction to your provided assay</i>
------------------	---

---

**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

---

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	<i>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.</i>
---------------	---

---

### 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	<i>This function allows you to add normalized count matrix to the SE object</i>
--------------	---

---

### 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)
```

**Arguments**

se SummarizedExperiment Object  
method Normalization Method, either 'CPM' or 'DESeq' or 'none' for log only  
log\_bool True or False; True to log normalize the data set after normalization method  
assay\_to\_normalize Which SE assay to do normalization on  
output\_assay\_name name for the resulting normalized assay

**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
```

---

permutated_DESeq	<i>This function performs DESeq on the permuted dataset adjusted p-values.</i>
------------------	--

---

**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	<i>This function formats the PCA plot using ggplot</i>
-----------	--

---

**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

---

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	<i>Protein data with 39 protein expression levels</i>
--------------	---

---

**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	<i>Batch and Condition indicator for protein expression data</i>
---------------------	--

---

**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	<i>P-value Plotter This function allows you to plot p-values of explained variation</i>
--------------	---

---

**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(scrn)
se <- mockSCE()
differential_expression <- BatchQC::DE_analyze(se = se,
                                             method = "DESeq2",
                                             batch = "Treatment",
                                             conditions = c(
                                               "Mutation_Status"),
                                             assay_to_analyze = "counts")

pval_summary(differential_expression)
pval_plotter(differential_expression)
```

---

pval_summary	<i>Returns summary table for p-values of explained variation</i>
--------------	--

---

**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")

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

```

---

signature_data	<i>Signature data with 1600 gene expression levels</i>
----------------	--

---

### 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	<i>Calculate a standardized Pearson correlation coefficient</i>
-----------------------	---

---

### 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 dataframe
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)
```

---

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
)
```

**Arguments**

se	SummarizedExperiment object
assay_to_normalize	Assay that should be corrected
var_of_interest	The experimental variable of interest
covar	list of covariates to include in sva analysis
output_assay_name	name of results assay

**Value**

SE object with an added sva corrected array

---

umap	<i>Create a umap plot; wrapper function for umap package pplus custom plotting</i>
------	--

---

**Description**

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

**Usage**

```
umap(
  se_object,
  assay_of_interest,
  batch,
  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
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")
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	<i>Volcano plot</i>
--------------	---------------------

---

### 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")

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

## \* **Internal**

check\_valid\_input, 8

## \* **datasets**

batch\_indicator, 7

protein\_data, 31

protein\_sample\_info, 31

signature\_data, 34

## \* **internal**

counts2pvalue, 12

batch\_correct, 5

batch\_design, 6

batch\_indicator, 7

BatchQC, 4

batchqc\_explained\_variation, 4

bladder\_data\_upload, 7

check\_valid\_input, 8

color\_palette, 8

ComBat\_correction, 9

ComBat\_seq\_correction, 10

commentary, 10

confound\_metrics, 11

cor\_props, 12

counts2pvalue, 12

covariates\_not\_confounded, 13

cramers\_v, 13

DE\_analyze, 18

dendrogram\_alpha\_numeric\_check, 14

dendrogram\_color\_palette, 15

dendrogram\_plotter, 15

DESeq2\_small\_size, 16

DESeq\_large\_analysis, 17

EV\_plotter, 19

EV\_table, 19

get.res, 20

goodness\_of\_fit\_DESeq2, 21

heatmap\_num\_to\_char\_converter, 22

heatmap\_plotter, 22

limma\_correction, 23

nb\_histogram, 24

nb\_proportion, 25

normalize\_SE, 25

PCA\_plotter, 26

permuted\_DESeq, 27

plot\_data, 28

possible\_distances, 29

possible\_k\_neighbors, 29

preprocess, 30

process\_dendrogram, 30

protein\_data, 31

protein\_sample\_info, 31

pval\_plotter, 32

pval\_summary, 32

ratio\_plotter, 33

signature\_data, 34

std\_pearson\_corr\_coef, 34

summarized\_experiment, 35

sva\_correction, 35

umap, 36

variation\_ratios, 37

volcano\_plot, 38