

# Package ‘BatchQC’

April 21, 2026

**Type** Package

**Title** Batch Effects Quality Control Software

**Version** 2.7.3

**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, ggpubr, Harman, limma, matrixStats, methods, MASS, pheatmap, RColorBrewer, reader, reshape2, scan, 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, GeneExpression, 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** devel

**git\_last\_commit** 47f3c02

**git\_last\_commit\_date** 2026-02-10

**Repository** Bioconductor 3.24

**Date/Publication** 2026-04-21

**Author** Jessica Anderson [aut] (ORCID: <<https://orcid.org/0000-0002-0542-9872>>),  
W. Evan Johnson [aut, fnd] (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],  
Sean Lu [ctb]

**Maintainer** Yaoan Leng <leng@bu.edu>

## Contents

AIC_boxplots . . . . .	4
BatchQC . . . . .	5
batchqc_explained_variation . . . . .	5

batch_correct	6
batch_design	7
batch_indicator	8
bisect	8
bladder_data_upload	9
BMI_data	9
check_valid_input	10
color_palette	10
ComBat_correction	11
ComBat_seq_correction	11
commentary	12
compute_aic	13
compute_lambda	14
confound_metrics	15
cor_props	15
counts2pvalue	16
covariates_not_confounded	16
cramers_v	17
dendrogram_alpha_numeric_check	18
dendrogram_color_palette	18
dendrogram_plotter	19
DESeq2_small_size	20
DESeq_large_analysis	21
DE_analyze	22
edgeR_large_analysis	23
edgeR_small_size	24
EV_plotter	25
EV_table	25
get.res	26
goodness_of_fit_nb	26
Harman_correction	28
heatmap_num_to_char_converter	29
heatmap_plotter	29
is_design_balanced	30
kBET	31
limma_correction	32
merged_IDS	33
nb_down_sample	33
nb_histogram	34
nb_proportion	34
normalize_SE	35
PCA_plotter	36
permuted_DESeq	37
permuted_edgeR	37
plot_data	38
plot_kBET	39
possible_distances	39
possible_k_neighbors	40
preprocess	40
process_dendrogram	41
protein_data	41
protein_sample_info	42

pvalues_all_genes . . . . .	42
pval_plotter . . . . .	43
pval_summary . . . . .	43
ratio_plotter . . . . .	44
run_AIC_models . . . . .	45
run_kBET . . . . .	45
run_lambda . . . . .	47
signature_data . . . . .	48
std_pearson_corr_coef . . . . .	48
summarized_experiment . . . . .	49
summary_stats_EV_table . . . . .	49
svaseq_correction . . . . .	50
sva_correction . . . . .	51
tb_data_upload . . . . .	52
umap . . . . .	52
variation_ratios . . . . .	53
volcano_plot . . . . .	54
<b>Index</b>	<b>56</b>

---

AIC\_boxplots

*Boxplots for the distribution of AIC for each method*


---

## Description

This function creates a boxplot of all the AIC values for each gene under each tested distribution to aid in identifying outliers

## Usage

```
AIC_boxplots(AIC_data, num_methods)
```

## Arguments

AIC_data	dataframe with the data to be plotted
num_methods	integer representing the number of distribution methods

## Value

AIC\_boxplot; a boxplot for each method showing distribution of data

---

`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, psva, num_sv, limit, numrepeats)
```

**Arguments**

se	SummarizedExperiment object
method	Normalization Method ("ComBat-Seq", "ComBat", "limma", "sva", svaseq, "Harman")
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
psva	boolean; default: FALSE. Only used if normalization method is "sva". If set to TRUE and no covariate input, psva function from the sva package will be used to remove batch effect.
num_sv	boolean; default: FALSE. Only used if normalization method is "svaseq". 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.
limit	numeric; default: 0.95. Only used if normalization method is "Harman". Indicates the limit of confidence in which to stop removing a batch effect. Must be between 0 and 1.
numrepeats	integer; default: 100000L. Only used if normalization method is "Harman". The number of repeats in which to run the simulated batch mean distribution estimator using the random selection algorithm.

**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	<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
```

---

bisect	<i>bisect - a generic bisection function</i>
--------	--

---

### 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, ..., tol_x = 5, tol_y = 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
tol_x	break condition for argument (defaults to 10)
tol_y	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	<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()
```

---

BMI_data	<i>This function returns BMI data that comes from 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"</i>
----------	---

---

### Description

This function returns BMI data that comes from 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	<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	<i>ComBat-Seq Correction This function applies ComBat-seq correction to your summarized experiment object</i>
-----------------------	---

---

**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	<i>This function creates the commentary recommendation when there are more than 20 samples.</i>
------------	---

---

**Description**

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

**Usage**

```
commentary(nb_fit_pval, count_below_value_pval, proportion, low_pval, method)
```

**Arguments**

nb_fit_pval	Boolean representing if the p-val count is below threshold
count_below_value_pval	number of features below p-val threshold
proportion	numeric; proportion of genes below the p-value
low_pval	pval threshold
method	string; method utilized for the parametric or non-parametric test; either "DE-Seq2" or "edgeR"

**Value**

a commentary string statement

---

compute_aic	<i>Compute the AIC for lognormal (ComBat) model, negative binomial (ComBat-seq) model and the Voom model</i>
-------------	--

---

### 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 models in comparison.

**min\_AIC** The number of minimum AIC across the models in comparison for individual genes.

**AIC\_score** The ratios between total\_AIC and min\_AIC for the models in comparison. The optimal model is the one has minimum value.

### 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 and yields AIC-based metric values. Must have non-negative data and non-discrete data will only analyze the lognormal and voom distribution.

**Value**

A list with a boxplot of the AIC\_boxplot and a dataframe containing the total\_AIC values, the frequency of the min\_AIC values, the AIC\_score, and the median\_AIC value, for the following three elements:

**nb** The metric value calculated based on the negative binomial model.

**lognormal** The metric value calculated based on the lognormal model.

**voom** The metric value calculated based on the voom-based model.

**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)
```

---

 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	<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	<i>This function allows you to calculate correlation properties</i>
-----------	---

---

**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	<i>This function calculates p-values for each gene given counts, estimated NB size, and estimated NB mean</i>
---------------	---

---

**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	<i>Returns list of covariates not confounded by batch; helper function for explained variation and for populating shiny app condition options</i>
---------------------------	---

---

**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,  
  model_formula,  
  num_samples,  
  small_sample_cutoff  
)
```

### 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)
model_formula	the stat formula to be used in the DESeq analysis
num_samples	total number of samples to analyze
small_sample_cutoff	value at which non-parametric test was used (considered "large sample size") vs parametric was used (considered "small sample size")

### 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 150 samples).*

---

### Description

This function calculated the goodness of fit of DESeq2 for larger sample sizes (intended for more than 150 samples).

### Usage

```
DESeq_large_analysis(  
  count_matrix,  
  condition,  
  other_variables,  
  conditions_df,  
  model_formula,  
  num_samples,  
  sampled,  
  small_sample_cutoff  
)
```

### 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)
model_formula	the stat formula to be used in the DESeq analysis
num_samples	total number of samples to analyze
sampled	the down sampled matrix
small_sample_cutoff	value at which non-parametric test was used (considered "large sample size") vs parametric was used (considered "small sample size")

### Value

a list containing the string recommendation

**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)
```

---

edgeR\_large\_analysis    *This function calculated the goodness of fit of edgeR for larger sample sizes (intended for more than 150 samples).*

---

### Description

This function calculated the goodness of fit of edgeR for larger sample sizes (intended for more than 150 samples).

### Usage

```
edgeR_large_analysis(  
  count_matrix,  
  condition,  
  other_variables,  
  conditions_df,  
  model_formula,  
  num_samples,  
  sampled,  
  small_sample_cutoff  
)
```

### 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)
model_formula	the stat formula to be used in the DESeq analysis
num_samples	total number of samples to analyze
sampled	the down sampled matrix
small_sample_cutoff	value at which non-parametric test was used (considered "large sample size") vs parametric was used (considered "small sample size")

### Value

a list containing the string recommendation

---

edgeR_small_size	<i>This function calculated the goodness of fit of edgeR for small sample sizes (intended for less than or equal to 20 samples).</i>
------------------	--

---

### Description

This function calculated the goodness of fit of edgeR for small sample sizes (intended for less than or equal to 20 samples).

### Usage

```
edgeR_small_size(  
  count_matrix,  
  condition,  
  other_variables,  
  conditions_df,  
  model_formula,  
  num_samples,  
  small_sample_cutoff  
)
```

### 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)
model_formula	the stat formula to be used in the DESeq analysis
num_samples	total number of samples to analyze
small_sample_cutoff	value at which non-parametric test was used (considered "large sample size") vs parametric was used (considered "small sample size")

### Value

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

---

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	<i>Helper function to get residuals</i>
---------	---

---

**Description**

Helper function to get residuals

**Usage**

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

**Arguments**

y	assay
X	model matrix design

**Value**

residuals

---

goodness_of_fit_nb	<i>This function calculates goodness-of-fit pvalues for all genes by looking at how the NB model by edgeR or DESeq2 fit the data</i>
--------------------	--

---

**Description**

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

**Usage**

```
goodness_of_fit_nb(
  se,
  count_matrix,
  condition,
  other_variables = NULL,
  method = "edgeR",
  num_genes = 500,
  small_sample_cutoff = 21
)
```

**Arguments**

se	the se object; se object where all the data is contained
count_matrix	string; name of the assay with gene expression matrix (in counts)
condition	string; name of the se colData with the condition status
other_variables	string; name of the se colData containing other variables of interest that should be considered in the model
method	string; method to use for the parametric or non-parametric test; either "DESeq2" or "edgeR"; default is "edgeR"
num_genes	down sample value, default is 500 (or all genes if less)
small_sample_cutoff	value at which non-parametric test will be used (considered "large sample size") vs parametric will be used (considered "small sample size"); default is 21

**Value**

a matrix of p-values where each row is a gene and each column is a level within the condition of interest

**Examples**

```
# example code for small sample
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_nb(se = se, count_matrix = "counts",
  condition = "Treatment", other_variables = "Mutation_Status",
  method = "edgeR")
nb_results[1]
nb_results[2]
nb_results[3]

# example code for large sample
library(scran)
se <- mockSCE(ncells = 150)
se$Treatment <- as.factor(se$Treatment)
se$Mutation_Status <- as.factor(se$Mutation_Status)
nb_results <- goodness_of_fit_nb(se = se, count_matrix = "counts",
  condition = "Treatment", other_variables = "Mutation_Status",
  method = "edgeR")
```

```
nb_results[1]
nb_results[2]
nb_results[3]
```

---

Harman_correction	<i>Harman Correction This function applies Harman correction to a summarized experiment object with and reconstructs the data back into the original feature space</i>
-------------------	--

---

### Description

Harman Correction This function applies Harman correction to a summarized experiment object with and reconstructs the data back into the original feature space

### Usage

```
Harman_correction(
  se,
  assay_to_normalize,
  batch,
  covar,
  output_assay_name,
  limit = 0.95,
  numrepeats = 100000L
)
```

### Arguments

se	SummarizedExperiment object
assay_to_normalize	string; name of assay that should be corrected
batch	factor; The variable that represents batch
covar	string; name of covariate.
output_assay_name	string; name of results assay
limit	numeric; default: 0.95 Indicates the limit of confidence in which to stop removing a batch effect. Must be between 0 and 1
numrepeats	integer; default: 100000L the number of repeats in which to run the simulated batch mean distribution estimator using the random selection algorithm.

### Value

SE object with an added Harman corrected reconstructed data

---

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)
```

---

<code>limma_correction</code>	<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**

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

**Value**

SE object with an added limma corrected array

---

merged_IDs	<i>BMI and matched sample names for TB data</i>
------------	---

---

**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_down_sample	<i>Performs down sampling for negative binomial model fit check.</i>
----------------	--

---

**Description**

Performs down sampling for negative binomial model fit check.

**Usage**

```
nb_down_sample(count_matrix, num_genes)
```

**Arguments**

count\_matrix    matrix; contains the feature data (should be counts)

num\_genes       integer; number of genes to use in down sampling

**Value**

list containing "sampled" a down sampled assay and "count\_matrix" with the down sampled count\_matrix

---

nb_histogram	<i>This function creates a histogram from the negative binomial goodness-of-fit adjusted pvalues.</i>
--------------	---

---

**Description**

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

**Usage**

```
nb_histogram(p_val_table)
```

**Arguments**

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

---

**Description**

This function determines the proportion of p-values below a specific value and compares to the previously determined threshold

**Usage**

```
nb_proportion(
  p_val_table,
  low_pval,
  threshold,
  num_samples,
  small_sample_cutoff,
  method
)
```

**Arguments**

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

num\_samples      the number of samples in the analysis

small\_sample\_cutoff      value at which non-parametric test will be used (considered "large sample size") vs parametric will be used (considered "small sample size"); default is 20

method            string; method utilized for the parametric or non-parametric test; either "DE-Seq2" or "edgeR"

**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,
  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

```

---

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

---

### Description

This function performs DESeq on the permuted dataset.

### Usage

```

permutated_DESeq(
  count_matrix,
  condition,
  other_variables,
  conditions_df,
  model_formula
)

```

### 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)
model_formula	the stat formula to be used in the DESeq analysis

### Value

a DESeq2 object

---

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

---

### Description

This function performs edgeR on the permuted dataset adjusted pvalues.

**Usage**

```
permuted_edgeR(
  count_matrix,
  condition,
  other_variables,
  conditions_df,
  model_formula
)
```

**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)

model\_formula    the stat formula to be used in the DESeq analysis

**Value**

edgeR fit

---

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

---

plot_kBET	<i>kBET Rejection Plotter</i>
-----------	-------------------------------

---

**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	<i>Create potential min_distance values for exploratory analysis based on the value of spread</i>
--------------------	---

---

**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	<i>Create a vector of possible nearest neighbor values from 5, 15, 25, 50, and 100</i>
----------------------	--

---

**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	<i>Preprocess assay data</i>
------------	------------------------------

---

**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	<i>Process Dendrogram</i>
--------------------	---------------------------

---

**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     *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

---

pvalues\_all\_genes     *This function calculates goodness-of-fit p-values for each condition level for each gene size, and estimated NB mean*

---

### Description

This function calculates goodness-of-fit p-values for each condition level for each gene size, and estimated NB mean

### Usage

```
pvalues_all_genes(condition, size, mu_matrix, count_matrix)
```

### Arguments

**condition**     string; name of the se colData with the condition status

**size**     numeric; an estimated size parameter of the NB distributions for the gene

**mu\_matrix**     matrix; estimated mu parameter of the NB distributions for different samples of each gene

**count\_matrix**     string; name of the assay with gene expression matrix (in counts)

### Value

a data frame of p-values for each gene under each biological condition

---

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(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	<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",
                                              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_AIC_models	<i>Helper function that contains the code to run the lognormal, voom, and negative binomial AIC models for compute_aic</i>
----------------	--

---

**Description**

Helper function that contains the code to run the lognormal, voom, and negative binomial AIC models for compute\_aic

**Usage**

```
run_AIC_models(dat, design, nb_test, maxit)
```

**Arguments**

dat	dataframe of the data to analyze
design	stats design model to be used in the analyses
nb_test	boolean; should negative binomial run (must be discrete data)
maxit	integer; the max number of IWLS iterations

**Value**

data frame; containing the AIC results of each method

---

run_kBET	<i>kBET rejection rate</i>
----------	----------------------------

---

**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	<i>Provide a recommendation on batch correction based on lambda calculation</i>
------------	---

---

### Description

This function 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)) 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	<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 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)
```

---

summary\_stats\_EV\_table

*Summary Stats EV Table Returns table with min, 1st quartile, mean, 2nd quartile, and max for each variable in the explained variation boxplot*

---

### Description

Summary Stats EV Table Returns table with min, 1st quartile, mean, 2nd quartile, and max for each variable in the explained variation boxplot

### Usage

```
summary_stats_EV_table(batchqc_ev)
```

### Arguments

batchqc_ev	explained variation results from
------------	----------------------------------

**Value**

summary\_stats\_table dataframe containing the min, 1st quartile, mean, 2nd quartile, and max for each variable in the explained variation boxplot

**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")
summary_stats_table <- BatchQC::summary_stats_EV_table(exp_var_result[[1]])

summary_stats_table
```

---

svaseq_correction	<i>svaseq Correction This function applies sva correction to a summarized experiment object with count based RNA-seq data</i>
-------------------	---

---

**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	<i>sva Correction This function applies sva correction to a summarized experiment object (implementation adapted from sva::psva)</i>
----------------	--

---

**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	<i>TB data upload This function uploads the TB data set from the curatedTBData package.</i>
----------------	---

---

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

---

**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,
  log_option = 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
log_option	Boolean; default is FALSE, if TRUE, log(assay_of_interest + 1) is computed and used for plotting; useful for count data

**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", log_option = TRUE)
umap_plot
```

---

variation_ratios	<i>Creates Ratios of batch to variable variation statistic</i>
------------------	--

---

**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

- \* **Internal**
  - check\_valid\_input, 10
- \* **datasets**
  - batch\_indicator, 8
  - merged\_IDs, 33
  - protein\_data, 41
  - protein\_sample\_info, 42
  - signature\_data, 48
- \* **internal**
  - counts2pvalue, 16
  - nb\_down\_sample, 33
  - pvalues\_all\_genes, 42
- AIC\_boxplots, 4
- batch\_correct, 6
- batch\_design, 7
- batch\_indicator, 8
- BatchQC, 5
- batchqc\_explained\_variation, 5
- bisect, 8
- bladder\_data\_upload, 9
- BMI\_data, 9
- check\_valid\_input, 10
- color\_palette, 10
- ComBat\_correction, 11
- ComBat\_seq\_correction, 11
- commentary, 12
- compute\_aic, 13
- compute\_lambda, 14
- confound\_metrics, 15
- cor\_props, 15
- counts2pvalue, 16
- covariates\_not\_confounded, 16
- cramers\_v, 17
- DE\_analyze, 22
- dendrogram\_alpha\_numeric\_check, 18
- dendrogram\_color\_palette, 18
- dendrogram\_plotter, 19
- DESeq2\_small\_size, 20
- DESeq\_large\_analysis, 21
- edgeR\_large\_analysis, 23
- edgeR\_small\_size, 24
- EV\_plotter, 25
- EV\_table, 25
- get.res, 26
- goodness\_of\_fit\_nb, 26
- Harman\_correction, 28
- heatmap\_num\_to\_char\_converter, 29
- heatmap\_plotter, 29
- is\_design\_balanced, 30
- kBET, 31
- limma\_correction, 32
- merged\_IDs, 33
- nb\_down\_sample, 33
- nb\_histogram, 34
- nb\_proportion, 34
- normalize\_SE, 35
- PCA\_plotter, 36
- permuted\_DESeq, 37
- permuted\_edgeR, 37
- plot\_data, 38
- plot\_kBET, 39
- possible\_distances, 39
- possible\_k\_neighbors, 40
- preprocess, 40
- process\_dendrogram, 41
- protein\_data, 41
- protein\_sample\_info, 42
- pval\_plotter, 43
- pval\_summary, 43
- pvalues\_all\_genes, 42
- ratio\_plotter, 44
- run\_AIC\_models, 45
- run\_kBET, 45
- run\_lambda, 47
- signature\_data, 48

std\_pearson\_corr\_coef, [48](#)  
summarized\_experiment, [49](#)  
summary\_stats\_EV\_table, [49](#)  
sva\_correction, [51](#)  
svaseq\_correction, [50](#)  
  
tb\_data\_upload, [52](#)  
  
umap, [52](#)  
  
variation\_ratios, [53](#)  
volcano\_plot, [54](#)