

# Package ‘BatchQC’

May 17, 2024

**Type** Package

**Title** Batch Effects Quality Control Software

**Version** 2.0.0

**Date** 2024-04-02

**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 (>= 4.3.0)

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

**Suggests** BiocManager, BiocStyle, bladderbatch, dendextend, devtools, knitr, lintr, plotly, rmarkdown, shinythemes, 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** 7.3.1

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

**git\_branch** RELEASE\_3\_19

**git\_last\_commit** 906d8d9

**git\_last\_commit\_date** 2024-04-30

**Repository** Bioconductor 3.19

**Date/Publication** 2024-05-17

**Author** Jessica McClintock [aut, cre] (<<https://orcid.org/0000-0002-0542-9872>>),  
W. Evan Johnson [aut] (<<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 McClintock <jessica.mcclintock@rutgers.edu>

## Contents

BatchQC . . . . .	3
batchqc_explained_variation . . . . .	4
batch_correct . . . . .	5
batch_design . . . . .	6
batch_indicator . . . . .	6
bladder_data_upload . . . . .	7
check_valid_input . . . . .	8
color_palette . . . . .	8
combat_correction . . . . .	9
combat_seq_correction . . . . .	9
confound_metrics . . . . .	10

cor_props . . . . .	11
covariates_not_confounded . . . . .	11
cramers_v . . . . .	12
dendrogram_alpha_numeric_check . . . . .	13
dendrogram_color_palette . . . . .	13
dendrogram_plotter . . . . .	14
DE_analyze . . . . .	15
EV_plotter . . . . .	16
EV_table . . . . .	16
get.res . . . . .	17
heatmap_num_to_char_converter . . . . .	18
heatmap_plotter . . . . .	18
normalize_SE . . . . .	19
PCA_plotter . . . . .	20
plot_data . . . . .	21
preprocess . . . . .	22
process_dendrogram . . . . .	23
protein_data . . . . .	23
protein_sample_info . . . . .	24
pval_plotter . . . . .	24
pval_summary . . . . .	25
signature_data . . . . .	26
std_pearson_corr_coef . . . . .	26
summarized_experiment . . . . .	27
volcano_plot . . . . .	27
<b>Index</b>	<b>29</b>

---

BatchQC	<i>Run BatchQC shiny app</i>
---------	------------------------------

---

## 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
assay_to_normalize	Which assay use to do normalization
batch	The batch
group	The group variable
covar	Covariate Matrix
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	<i>This function allows you to make a batch design matrix</i>
--------------	---

---

**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 check for valid input</i>
-------------------	---

---

**Description**

Helper function to check for valid input

**Usage**

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

**Arguments**

se	se object
batch	batch
condition	condition

**Value**

True/False boolean; True if all input is valid, False if invalid

---

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(scrn)
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	Covariate Matrix
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	Covariate Matrix
output_assay_name	name of results assay

**Value**

SE object with an added combat-seq corrected array

---

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
```

---

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



```

pval_summary(differential_expression)
pval_plotter(differential_expression)
assay_to_analyze = "counts")

```

---

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(scraper)
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

---

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
```

---

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, assays, xaxisPC, yaxisPC,
            log_option = FALSE)
```

**Arguments**

se	SummarizedExperiment object
nfeature	number of features
color	choose a color
shape	choose a shape
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",
                                assays = c("counts", "ComBat_Seq_Corrected"),
                                xaxisPC = 1, yaxisPC = 2, log_option = FALSE)

pca_plot$plot
pca_plot$var_explained
```

---

plot\_data

*This function formats the PCA plot using ggplot*

---

**Description**

This function formats the PCA plot using ggplot

**Usage**

```
plot_data(pca_plot_data, color, shape, 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
xaxisPC	the PC to plot as the x axis
yaxisPC	the PC to plot as the y axis

**Value**

PCA plot

---

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      *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(scraper)  
se <- mockSCE()  
process_dendro <- BatchQC::process_dendrogram(se, "counts")  
process_dendro
```

---

protein\_data      *Protein data with 39 protein expression levels*

---

**Description**

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

**Usage**

```
data(protein_data)
```

**Format**

A data frame with 39 rows and 24 variables

---

protein_sample_info	<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(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)
```

---

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

---

**Description**

Returns summary table for p-values of explained variation

**Usage**

```
pval_summary(res_list)
```

**Arguments**

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

**Value**

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

**Examples**

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

pval_summary(differential_expression)
```

---

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)
```

---

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")

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

## \* datasets

- batch\_indicator, [6](#)
- protein\_data, [23](#)
- protein\_sample\_info, [24](#)
- signature\_data, [26](#)

batch\_correct, [5](#)  
batch\_design, [6](#)  
batch\_indicator, [6](#)  
BatchQC, [3](#)  
batchqc\_explained\_variation, [4](#)  
bladder\_data\_upload, [7](#)

check\_valid\_input, [8](#)  
color\_palette, [8](#)  
combat\_correction, [9](#)  
combat\_seq\_correction, [9](#)  
confound\_metrics, [10](#)  
cor\_props, [11](#)  
covariates\_not\_confounded, [11](#)  
cramers\_v, [12](#)

DE\_analyze, [15](#)  
dendrogram\_alpha\_numeric\_check, [13](#)  
dendrogram\_color\_palette, [13](#)  
dendrogram\_plotter, [14](#)

EV\_plotter, [16](#)  
EV\_table, [16](#)

get.res, [17](#)

heatmap\_num\_to\_char\_converter, [18](#)  
heatmap\_plotter, [18](#)

normalize\_SE, [19](#)

PCA\_plotter, [20](#)  
plot\_data, [21](#)  
preprocess, [22](#)  
process\_dendrogram, [23](#)

protein\_data, [23](#)  
protein\_sample\_info, [24](#)  
pval\_plotter, [24](#)  
pval\_summary, [25](#)

signature\_data, [26](#)  
std\_pearson\_corr\_coef, [26](#)  
summarized\_experiment, [27](#)

volcano\_plot, [27](#)