

# Package ‘scPipe’

January 21, 2022

**Title** pipeline for single cell RNA-seq data analysis

**Date** 2019-10-12

**Version** 1.17.0

**Type** Package

**Maintainer** Luyi Tian <tian.l@wehi.edu.au>

**Author** Luyi Tian

**biocViews** ImmunoOncology, Software, Sequencing, RNASeq,  
GeneExpression, SingleCell, Visualization, SequenceMatching,  
Preprocessing, QualityControl, GenomeAnnotation

**Description** A preprocessing pipeline for single cell RNA-seq data that starts from the fastq files and produces a gene count matrix with associated quality control information. It can process fastq data generated by CEL-seq, MARS-seq, Drop-seq, Chromium 10x and SMART-seq protocols.

**Depends** R (>= 3.4), ggplot2, methods, SingleCellExperiment

**LinkingTo** Rcpp, Rhtslib (>= 1.13.1), zlibbioc, testthat

**Imports** Rhtslib, biomaRt, GGally, MASS, mclust, Rcpp (>= 0.11.3),  
reshape, BiocGenerics, robustbase, scales, utils, stats,  
S4Vectors, SummarizedExperiment, AnnotationDbi, org.Hs.eg.db,  
org.Mm.eg.db, stringr, rtracklayer, hash, dplyr, GenomicRanges,  
magrittr, glue (>= 1.3.0), rlang, scater (>= 1.11.0)

**SystemRequirements** C++11, GNU make

**License** GPL (>= 2)

**Encoding** UTF-8

**RoxygenNote** 6.1.1

**NeedsCompilation** yes

**URL** <https://github.com/LuyiTian/scPipe>

**BugReports** <https://github.com/LuyiTian/scPipe>

**Suggests** Rsubread, knitr, rmarkdown, testthat

**VignetteBuilder** knitr

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

**git\_branch** master

**git\_last\_commit** 051135a

**git\_last\_commit\_date** 2021-10-26

**Date/Publication** 2022-01-21

## R topics documented:

anno_import . . . . .	3
anno_to_saf . . . . .	3
calculate_QC_metrics . . . . .	4
cell_barcode_matching . . . . .	5
convert_geneid . . . . .	7
create_processed_report . . . . .	8
create_report . . . . .	8
create_sce_by_dir . . . . .	10
demultiplex_info . . . . .	11
detect_outlier . . . . .	12
gene_id_type . . . . .	14
get_ercc_anno . . . . .	15
get_genes_by_GO . . . . .	15
get_read_str . . . . .	16
organism.sce . . . . .	17
plot_demultiplex . . . . .	18
plot_mapping . . . . .	18
plot_QC_pairs . . . . .	19
plot_UMI_dup . . . . .	20
QC_metrics . . . . .	21
remove_outliers . . . . .	22
scPipe . . . . .	22
sc_correct_bam_bc . . . . .	23
sc_count_aligned_bam . . . . .	24
sc_demultiplex . . . . .	25
sc_demultiplex_and_count . . . . .	26
sc_detect_bc . . . . .	28
sc_exon_mapping . . . . .	29
sc_gene_counting . . . . .	30
sc_sample_data . . . . .	31
sc_sample_qc . . . . .	32
sc_trim_barcode . . . . .	33
UMI_duplication . . . . .	34
UMI_dup_info . . . . .	35

---

anno_import	<i>Import gene annotation</i>
-------------	-------------------------------

---

### Description

Because of the variations in data format depending on annotation source, this function has only been tested with human annotation from ENSEMBL, RefSeq and Gencode. If it behaves unexpectedly with any annotation please submit an issue at [www.github.com/LuyiTian/scPipe](http://www.github.com/LuyiTian/scPipe) with details.

### Usage

```
anno_import(filename)
```

### Arguments

filename            The name of the annotation gff3 or gtf file. File can be gzipped.

### Details

Imports and GFF3 or GTF gene annotation file and transforms it into a SAF formatted data.frame. SAF described at <http://bioinf.wehi.edu.au/featureCounts/>. SAF contains positions for exons, strand and the GeneID they are associated with.

### Value

data.frame containing exon information in SAF format

### Examples

```
ens_chrY <- anno_import(system.file("extdata", "ensembl_hg38_chrY.gtf.gz", package = "scPipe"))
```

---

anno_to_saf	<i>Convert annotation from GenomicRanges to Simple Annotation Format (SAF)</i>
-------------	--

---

### Description

This function converts a GRanges object into a data.frame of the SAF format for scPipe's consumption. The GRanges object should contain a "type" column where at least some features are annotated as "exon", in addition there should be a gene\_id column specifying the gene to which the exon belongs. In the SAF only the gene ID, chromosome, start, end and strand are recorded, this is a gene-exon centric format, with all entries containing the same gene ID treated as exons of that gene. It is possible to count alternative features by setting the gene\_id column to an arbitrary feature name and having alternative features in the SAF table, the main caveat is that the features are still treated as exons, and the mapping statistics for exon and intron will not reflect biological exons and introns but rather the annotation features.

**Usage**

```
anno_to_saf(anno)
```

**Arguments**

anno                    The GRanges object containing exon information

**Details**

Convert a GRanges object containing type and gene\_id information into a SAF format data.frame. SAF described at <http://bioinf.wehi.edu.au/featureCounts/>. SAF contains positions for exons, strand and the GeneID they are associated with.

**Value**

data.frame containing exon information in SAF format

**Examples**

```
## Not run:
anno <- system.file("extdata", "ensembl_hg38_chrY.gtf.gz", package = "scPipe")
saf_chrY <- anno_to_saf(rtracklayer::import(anno))

## End(Not run)
```

---

calculate\_QC\_metrics    *Calculate QC metrics from gene count matrix*

---

**Description**

Calculate QC metrics from gene count matrix

**Usage**

```
calculate_QC_metrics(sce)
```

**Arguments**

sce                    a SingleCellExperiment object containing gene counts

**Details**

get QC metrics using gene count matrix. The QC statistics added are

- `number_of_genes` number of genes detected.
- `total_count_per_cell` sum of read number after UMI deduplication.
- `non_mt_percent` 1 - percentage of mitochondrial gene counts. Mitochondrial genes are retrieved by GO term GO:0005739
- `non_ERCC_percent` ratio of exon counts to ERCC counts
- `non_ribo_percent` 1 - percentage of ribosomal gene counts ribosomal genes are retrieved by GO term GO:0005840.

**Value**

an `SingleCellExperiment` with updated QC metrics

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

# The sample qc data already run through function `calculate_QC_metrics`.
# So we delete these columns and run `calculate_QC_metrics` to get them again:
colnames(colnames(QC_metrics(sce)))
QC_metrics(sce) = QC_metrics(sce)[,c("unaligned", "aligned_unmapped", "mapped_to_exon")]
sce = calculate_QC_metrics(sce)
colnames(QC_metrics(sce))
```

---

`cell_barcode_matching` *cell barcode demultiplex statistics for a small sample scRNA-seq dataset to demonstrate capabilities of scPipe*

---

**Description**

This data.frame contains cell barcode demultiplex statistics with several rows:

- `barcode_unmatch_ambiguous_mapping` is the number of reads that do not match any barcode, but aligned to the genome and mapped to multiple features.
- `barcode_unmatch_mapped_to_intron` is the number of reads that do not match any barcode, but aligned to the genome and mapped to intron.
- `barcode_match` is the number of reads that match the cell barcodes

- `barcode_unmatch_unaligned` is the number of reads that do not match any barcode, and not aligned to the genome
- `barcode_unmatch_aligned` is the number of reads that do not match any barcode, but aligned to the genome and do not mapped to any feature
- `barcode_unmatch_mapped_to_exon` is the number of reads that do not match any barcode, but aligned to the genome and mapped to the exon

**Usage**

```
sc_sample_qc
```

**Format**

a `data.frame` instance, one row per cell.

**Value**

NULL, but makes a data frame with cell barcode demultiplex statistics

**Author(s)**

Luyi Tian

**Source**

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

demultiplex_info(sce)
```

---

convert_geneid	<i>convert the gene ids of a SingleCellExperiment object</i>
----------------	--

---

### Description

convert the gene ids of a SingleCellExperiment object

### Usage

```
convert_geneid(sce, returns = "external_gene_name", all = TRUE)
```

### Arguments

sce	a SingleCellExperiment object
returns	the gene id which is set as return. Default to be 'external_gene_name'. A possible list of attributes can be retrieved using the function <code>listAttributes</code> from <code>biomaRt</code> package. The commonly used id types are 'external_gene_name', 'ensembl_gene_id' or 'entrezgene'.
all	logic. For genes that cannot convert to new gene id, keep them with the old id or delete them. The default is keep them.

### Details

convert the gene id of all datas in the SingleCellExperiment object

### Value

sce with converted id

### Examples

```
# the gene id in example data are `external_gene_name`
# the following example will convert it to `external_gene_name`.
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
head(rownames(sce))
sce = convert_geneid(sce, return="external_gene_name")
head(rownames(sce))
```

---

```
create_processed_report
      create_processed_report
```

---

### Description

create an HTML report summarising pro-processed data. This is an alternative to the more verbose `create_report` that requires only the processed counts and stats folders.

### Usage

```
create_processed_report(outdir = ".", organism, gene_id_type,
  report_name = "report")
```

### Arguments

<code>outdir</code>	output folder
<code>organism</code>	the organism of the data. List of possible names can be retrieved using the function <code>'listDatasets'</code> from <code>'biomaRt'</code> package. (i.e <code>'mmusculus_gene_ensembl'</code> or <code>'hsapiens_gene_ensembl'</code> )
<code>gene_id_type</code>	gene id type of the data A possible list of ids can be retrieved using the function <code>'listAttributes'</code> from <code>'biomaRt'</code> package. the commonly used id types are <code>'external_gene_name'</code> , <code>'ensembl_gene_id'</code> or <code>'entrezgene'</code>
<code>report_name</code>	the name of the report .Rmd and .html files.

### Examples

```
## Not run:
create_report(
  outdir="output_dir_of_scPipe",
  organism="mmusculus_gene_ensembl",
  gene_id_type="ensembl_gene_id")

## End(Not run)
```

---

```
create_report      create_report
```

---

### Description

create an HTML report using data generated by preprocessing step.



**Usage**

```
create_report(sample_name, outdir, r1 = "NA", r2 = "NA", outfq = "NA",
  read_structure = list(bs1 = 0, bl1 = 0, bs2 = 0, bl2 = 0, us = 0, ul = 0),
  filter_settings = list(rmlow = TRUE, rmN = TRUE, minq = 20, numbq = 2),
  align_bam = "NA", genome_index = "NA", map_bam = "NA",
  exon_anno = "NA", stnd = TRUE, fix_chr = FALSE, barcode_anno = "NA",
  max_mis = 1, UMI_cor = 1, gene_fl = FALSE, organism, gene_id_type)
```

**Arguments**

sample_name	sample name
outdir	output folder
r1	file path of read1
r2	file path of read2 default to be NULL
outfq	file path of the output of sc_trim_barcode
read_structure	a list contains read structure configuration. For more help see ‘?sc_trim_barcode’
filter_settings	a list contains read filter settings for more help see ‘?sc_trim_barcode’
align_bam	the aligned bam file
genome_index	genome index used for alignment
map_bam	the mapped bam file
exon_anno	the gff exon annotation used. Can have multiple files
stnd	whether to perform strand specific mapping
fix_chr	add ‘chr’ to chromosome names, fix inconsistant names.
barcode_anno	cell barcode annotation file path.
max_mis	maximum mismatch allowed in barcode. Default to be 1
UMI_cor	correct UMI sequence error: 0 means no correction, 1 means simple correction and merge UMI with distance 1.
gene_fl	whether to remove low abundant gene count. Low abundant is defined as only one copy of one UMI for this gene
organism	the organism of the data. List of possible names can be retrieved using the function ‘listDatasets’ from ‘biomaRt’ package. (i.e ‘mmusculus_gene_ensembl’ or ‘hsapiens_gene_ensembl’)
gene_id_type	gene id type of the data A possible list of ids can be retrieved using the function ‘listAttributes’ from ‘biomaRt’ package. the commonly used id types are ‘external_gene_name’, ‘ensembl_gene_id’ or ‘entrezgene’

**Value**

no return

**Examples**

```
## Not run:
create_report(sample_name="sample_001",
  outdir="output_dir_of_scPipe",
  r1="read1.fq",
  r2="read2.fq",
  outfq="trim.fq",
  read_structure=list(bs1=-1, b11=2, bs2=6, b12=8, us=0, u1=6),
  filter_settings=list(rmlow=TRUE, rmN=TRUE, minq=20, numbq=2),
  align_bam="align.bam",
  genome_index="mouse.index",
  map_bam="aligned.mapped.bam",
  exon_anno="exon_anno.gff3",
  stnd=TRUE,
  fix_chr=FALSE,
  barcode_anno="cell_barcode.csv",
  max_mis=1,
  UMI_cor=1,
  gene_fl=FALSE,
  organism="mmusculus_gene_ensembl",
  gene_id_type="ensembl_gene_id")

## End(Not run)
```

---

create\_sce\_by\_dir      *create a SingleCellExperiment object from data folder generated by preprocessing step*

---

**Description**

after we run `sc_gene_counting` and finish the preprocessing step. `create_sce_by_dir` can be used to generate the [SingleCellExperiment](#) object from the folder that contains gene count matrix and QC statistics. it can also generate the html report based on the gene count and quality control statistics

**Usage**

```
create_sce_by_dir(datadir, organism = NULL, gene_id_type = NULL,
  pheno_data = NULL, report = FALSE)
```

**Arguments**

`datadir`            the directory that contains all the data and 'stat' subfolder.

`organism`           the organism of the data. List of possible names can be retrieved using the function 'listDatasets' from 'biomaRt' package. (i.e 'mmusculus\_gene\_ensembl' or 'hsapiens\_gene\_ensembl')

gene_id_type	gene id type of the data A possible list of ids can be retrieved using the function 'listAttributes' from 'biomaRt' package. the commonly used id types are 'external_gene_name', 'ensembl_gene_id' or 'entrezgene'
pheno_data	the external phenotype data that linked to each single cell. This should be an AnnotatedDataFrame object
report	whether to generate the html report in the data folder

### Details

after we run `sc_gene_counting` and finish the preprocessing step. `create_sce_by_dir` can be used to generate the `SingleCellExperiment` object from the folder that contains gene count matrix and QC statistics.

### Value

a `SingleCellExperiment` object

### Examples

```
## Not run:
# the sce can be created from the output folder of scPipe
# please refer to the vignettes
sce = create_sce_by_dir(datadir="output_dir_of_scPipe",
  organism="mmusculus_gene_ensembl",
  gene_id_type="ensembl_gene_id")

## End(Not run)
# or directly from the gene count and quality control matrix:
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
dim(sce)
```

---

demultiplex\_info      *demultiplex\_info*

---

### Description

Get or set cell barcode demultiplex results in a `SingleCellExperiment` object

**Usage**

```
demultiplex_info(object)

demultiplex_info(object) <- value

demultiplex_info.sce(object)

## S4 method for signature 'SingleCellExperiment'
demultiplex_info(object)

## S4 replacement method for signature 'SingleCellExperiment'
demultiplex_info(object) <- value
```

**Arguments**

object	A <a href="#">SingleCellExperiment</a> object.
value	Value to be assigned to corresponding object.

**Value**

a dataframe of cell barcode demultiplex information  
A DataFrame of cell barcode demultiplex results.

**Author(s)**

Luyi Tian

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

demultiplex_info(sce)
```

**Description**

This algorithm will try to find `comp` number of components in quality control metrics using a Gaussian mixture model. Outlier detection is performed on the component with the most genes detected. The rest of the components will be considered poor quality cells. More cells will be classified low quality as you increase `comp`.

**Usage**

```
detect_outlier(sce, comp = 1, sel_col = NULL, type = c("low", "both",
  "high"), conf = c(0.9, 0.99), batch = FALSE)
```

**Arguments**

<code>sce</code>	a <code>SingleCellExperiment</code> object containing QC metrics.
<code>comp</code>	the number of component used in GMM. Depending on the quality of the experiment.
<code>sel_col</code>	a vector of column names which indicate the columns to use for QC. By default it will be the statistics generated by <code>'calculate_QC_metrics()'</code>
<code>type</code>	only looking at low quality cells ('low') or possible doublets ('high') or both ('both')
<code>conf</code>	confidence interval for linear regression at lower and upper tails. Usually, this is smaller for lower tail because we hope to pick out more low quality cells than doublets.
<code>batch</code>	whether to perform quality control separately for each batch. Default is FALSE. If set to TRUE then you should have a column called 'batch' in the <code>'colData(sce)'</code> .

**Details**

detect outlier using Mahalanobis distances

**Value**

an updated `SingleCellExperiment` object with an 'outlier' column in `colData`

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
# the sample qc data already run through function `calculate_QC_metrics`
# for a new sce please run `calculate_QC_metrics` before `detect_outlier`
sce = detect_outlier(sce)
table(QC_metrics(sce)$outliers)
```

---

gene_id_type	<i>Get or set gene_id_type from a SingleCellExperiment object</i>
--------------	---

---

### Description

Get or set gene\_id\_type from a SingleCellExperiment object

### Usage

```
gene_id_type(object)

gene_id_type(object) <- value

gene_id_type.sce(object)

## S4 method for signature 'SingleCellExperiment'
gene_id_type(object)

## S4 replacement method for signature 'SingleCellExperiment'
gene_id_type(object) <- value
```

### Arguments

object	A <a href="#">SingleCellExperiment</a> object.
value	Value to be assigned to corresponding object.

### Value

the gene id type used by Biomart  
gene id type string

### Author(s)

Luyi Tian

### Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

gene_id_type(sce)
```

---

get_ercc_anno	<i>Get ERCC annotation table</i>
---------------	----------------------------------

---

**Description**

Helper function to retrieve ERCC annotation as a dataframe in SAF format

**Usage**

```
get_ercc_anno()
```

**Value**

data.frame containing ERCC annotation

**Examples**

```
## Not run:  
ercc_anno <- get_ercc_anno()  
  
## End(Not run)
```

---

get_genes_by_GO	<i>Get genes related to certain GO terms from biomart database</i>
-----------------	--

---

**Description**

Get genes related to certain GO terms from biomart database

**Usage**

```
get_genes_by_GO(returns = "ensembl_gene_id",  
dataset = "mmusculus_gene_ensembl", go = NULL)
```

**Arguments**

returns	the gene id which is set as return. Default to be ensembl id A possible list of attributes can be retrieved using the function <code>listAttributes</code> from <code>biomaRt</code> package. The commonly used id types are 'external_gene_name', 'ensembl_gene_id' or 'entrezgene'.
dataset	Dataset you want to use. List of possible datasets can be retrieved using the function <code>listDatasets</code> from <code>biomaRt</code> package.
go	a vector of GO terms

**Details**

Get genes related to certain GO terms from biomart database

**Value**

a vector of gene ids.

**Examples**

```
# get all genes under GO term GO:0005739 in mouse, return ensembl gene id
get_genes_by_GO(returns="ensembl_gene_id",
  dataset="mmusculus_gene_ensembl",
  go=c('GO:0005739'))
```

---

get\_read\_str

*Get read structure for particular scRNA-seq protocol*

---

**Description**

The supported protocols are:

- CelSeq
- CelSeq2
- DropSeq
- 10x (also called ChromiumV1)

If you know the structure of a specific protocol and would like it supported, please leave a issue post at [www.github.com/luyitian/scPipe](http://www.github.com/luyitian/scPipe).

**Usage**

```
get_read_str(protocol)
```

**Arguments**

protocol            name of the protocol

**Value**

list of UMI and Barcode locations for use in other scPipe functions

**Examples**

```
get_read_str("celseq")
```



---

organism.sce	<i>Get or set organism from a SingleCellExperiment object</i>
--------------	---

---

## Description

Get or set organism from a SingleCellExperiment object

## Usage

```
organism.sce(object)

## S4 method for signature 'SingleCellExperiment'
organism(object)

## S4 replacement method for signature 'SingleCellExperiment'
organism(object) <- value
```

## Arguments

object	A <a href="#">SingleCellExperiment</a> object.
value	Value to be assigned to corresponding object.

## Value

organism string

## Author(s)

Luyi Tian

## Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

organism(sce)
```

---

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

---

**Description**

Plot cell barcode demultiplexing result for the SingleCellExperiment. The barcode demultiplexing result is shown using a barplot, with the bars indicating proportions of total reads. Barcode matches and mismatches are summarised along with whether or not the read mapped to the genome. High proportion of genome aligned reads with no barcode match may indicate barcode integration failure.

**Usage**

```
plot_demultiplex(sce)
```

**Arguments**

sce                    a SingleCellExperiment object

**Value**

a ggplot2 bar chart

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

plot_demultiplex(sce)
```

---

plot_mapping	<i>Plot mapping statistics for SingleCellExperiment object.</i>
--------------	---

---

**Description**

Plot mapping statistics for SingleCellExperiment object.

**Usage**

```
plot_mapping(sce, sel_col = NULL, percentage = FALSE, dataname = "")
```

**Arguments**

sce	a SingleCellExperiment object
sel_col	a vector of column names, indicating the columns to use for plot. by default it will be the mapping result.
percentage	TRUE to convert the number of reads to percentage
dataname	the name of this dataset, used as plot title

**Value**

a ggplot2 object

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

plot_mapping(sce, percentage=TRUE, dataname="sc_sample")
```

---

plot_QC_pairs	<i>Plot GGally pairs plot of QC statistics from SingleCellExperiment object</i>
---------------	---

---

**Description**

Plot GGally pairs plot of QC statistics from SingleCellExperiment object

**Usage**

```
plot_QC_pairs(sce, sel_col = NULL)
```

**Arguments**

sce	a SingleCellExperiment object
sel_col	a vector of column names which indicate the columns to use for plot. By default it will be the statistics generated by ‘calculate_QC_metrics()’

**Value**

a ggplot2 object

## Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
sce = detect_outlier(sce)

plot_QC_pairs(sce)
```

---

plot_UMI_dup	<i>Plot UMI duplication frequency</i>
--------------	---------------------------------------

---

## Description

Plot the UMI duplication frequency.

## Usage

```
plot_UMI_dup(sce, log10_x = TRUE)
```

## Arguments

sce	a SingleCellExperiment object
log10_x	whether to use log10 scale for x axis

## Value

a line chart of the UMI duplication frequency

## Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

plot_UMI_dup(sce)
```

---

QC\_metrics                      *Get or set quality control metrics in a SingleCellExperiment object*

---

**Description**

Get or set quality control metrics in a SingleCellExperiment object

**Usage**

```
QC_metrics(object)

QC_metrics(object) <- value

QC_metrics.sce(object)

## S4 method for signature 'SingleCellExperiment'
QC_metrics(object)

## S4 replacement method for signature 'SingleCellExperiment'
QC_metrics(object) <- value
```

**Arguments**

object                      A [SingleCellExperiment](#) object.  
value                      Value to be assigned to corresponding object.

**Value**

a dataframe of quality control metrics  
A DataFrame of quality control metrics.

**Author(s)**

Luyi Tian

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
QC_metrics(sce) = sc_sample_qc

head(QC_metrics(sce))
```

remove\_outliers            *Remove outliers in SingleCellExperiment*

---

**Description**

Removes outliers flagged by detect\_outliers()

**Usage**

```
remove_outliers(sce)
```

**Arguments**

sce                    a SingleCellExperiment object

**Value**

a SingleCellExperiment object without outliers

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
sce = detect_outlier(sce)
dim(sce)
sce = remove_outliers(sce)
dim(sce)
```

---

scPipe                    *scPipe - single cell RNA-seq pipeline*

---

**Description**

The scPipe will do cell barcode demultiplexing, UMI deduplication and quality control on fastq data generated from all protocols

**Author(s)**

Luyi Tian <tian.l@wehi.edu.au>; Shian Su <su.s@wehi.edu.au>

---

sc\_correct\_bam\_bc      *sc\_correct\_bam\_bc*

---

### Description

update the cell barcode tag in bam file with corrected barcode output to a new bam file. the function will be useful for methods that use the cell barcode information from bam file, such as ‘Demuxlet’

### Usage

```
sc_correct_bam_bc(inbam, outbam, bc_anno, max_mis = 1, bam_tags = list(am =
  "YE", ge = "GE", bc = "BC", mb = "OX"), mito = "MT", nthreads = 1)
```

### Arguments

inbam	input bam file. This should be the output of <code>sc_exon_mapping</code>
outbam	output bam file with updated cell barcode
bc_anno	barcode annotation, first column is cell id, second column is cell barcode sequence
max_mis	maximum mismatch allowed in barcode. (default: 1)
bam_tags	list defining BAM tags where mapping information is stored. <ul style="list-style-type: none"> <li>• "am": mapping status tag</li> <li>• "ge": gene id</li> <li>• "bc": cell barcode tag</li> <li>• "mb": molecular barcode tag</li> </ul>
mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
nthreads	number of threads to use. (default: 1)

### Value

no return

### Examples

```
data_dir="celseq2_demo"
barcode_annotation_fn = system.file("extdata", "barcode_anno.csv",
  package = "scPipe")
## Not run:
# refer to the vignettes for the complete workflow
...
sc_correct_bam_bc(file.path(data_dir, "out.map.bam"),
  file.path(data_dir, "out.map.clean.bam"),
  barcode_annotation_fn)
...

```

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

---

```
sc_count_aligned_bam  sc_count_aligned_bam
```

---

## Description

Wrapper to run [sc\\_exon\\_mapping](#), [sc\\_demultiplex](#) and [sc\\_gene\\_counting](#) with a single command

## Usage

```
sc_count_aligned_bam(inbam, outbam, annofn, bam_tags = list(am = "YE", ge =
  "GE", bc = "BC", mb = "OX"), bc_len = 8, UMI_len = 6, stnd = TRUE,
  fix_chr = FALSE, outdir, bc_anno, max_mis = 1, mito = "MT",
  has_UMI = TRUE, UMI_cor = 1, gene_fl = FALSE, keep_mapped_bam = TRUE,
  nthreads = 1)
```

## Arguments

inbam	input aligned bam file. can have multiple files as input
outbam	output bam filename
annofn	single string or vector of gff3 annotation filenames, data.frame in SAF format or GRanges object containing complete gene_id metadata column.
bam_tags	list defining BAM tags where mapping information is stored. <ul style="list-style-type: none"> <li>• "am": mapping status tag</li> <li>• "ge": gene id</li> <li>• "bc": cell barcode tag</li> <li>• "mb": molecular barcode tag</li> </ul>
bc_len	total barcode length
UMI_len	UMI length
stnd	TRUE to perform strand specific mapping. (default: TRUE)
fix_chr	TRUE to add 'chr' to chromosome names, MT to chrM. (default: FALSE)
outdir	output folder
bc_anno	barcode annotation, first column is cell id, second column is cell barcode sequence
max_mis	maximum mismatch allowed in barcode. (default: 1)
mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
has_UMI	whether the protocol contains UMI (default: TRUE)



UMI_cor	correct UMI sequencing error: 0 means no correction, 1 means simple correction and merge UMI with distance 1. 2 means merge on both UMI alignment position match.
gene_fl	whether to remove low abundance genes. A gene is considered to have low abundance if only one copy of one UMI is associated with it.
keep_mapped_bam	TRUE if feature mapped bam file should be retained.
nthreads	number of threads to use. (default: 1)

**Value**

no return

**Examples**

```
## Not run:
sc_count_aligned_bam(
  inbam = "aligned.bam",
  outbam = "mapped.bam",
  annofn = c("MusMusculus-GRCm38p4-UCSC.gff3", "ERCC92_anno.gff3"),
  outdir = "output",
  bc_anno = "barcodes.csv"
)

## End(Not run)
```

---

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

---

**Description**

Process bam file by cell barcode, output to outdir/count/[cell\_id].csv. the output contains information for all reads that can be mapped to exons. including the gene id, UMI of that read and the distance to transcript end position.

**Usage**

```
sc_demultiplex(inbam, outdir, bc_anno, max_mis = 1, bam_tags = list(am =
  "YE", ge = "GE", bc = "BC", mb = "OX"), mito = "MT", has_UMI = TRUE,
  nthreads = 1)
```

**Arguments**

inbam	input bam file. This should be the output of sc_exon_mapping
outdir	output folder

bc_anno	barcode annotation, first column is cell id, second column is cell barcode sequence
max_mis	maximum mismatch allowed in barcode. (default: 1)
bam_tags	list defining BAM tags where mapping information is stored. <ul style="list-style-type: none"> <li>• "am": mapping status tag</li> <li>• "ge": gene id</li> <li>• "bc": cell barcode tag</li> <li>• "mb": molecular barcode tag</li> </ul>
mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
has_UMI	whether the protocol contains UMI (default: TRUE)
nthreads	number of threads to use. (default: 1)

**Value**

no return

**Examples**

```

data_dir="celseq2_demo"
barcode_annotation_fn = system.file("extdata", "barcode_anno.csv",
  package = "scPipe")
## Not run:
# refer to the vignettes for the complete workflow
...
sc_demultiplex(file.path(data_dir, "out.map.bam"),
  data_dir,
  barcode_annotation_fn, has_UMI=FALSE)
...

## End(Not run)

```

---

```

sc_demultiplex_and_count
      sc_demultiplex_and_count

```

---

**Description**

Wrapper to run `sc_demultiplex` and `sc_gene_counting` with a single command

**Usage**

```

sc_demultiplex_and_count(inbam, outdir, bc_anno, max_mis = 1,
  bam_tags = list(am = "YE", ge = "GE", bc = "BC", mb = "OX"), mito = "MT",
  has_UMI = TRUE, UMI_cor = 1, gene_fl = FALSE, nthreads = 1)

```

**Arguments**

inbam	input bam file. This should be the output of sc_exon_mapping
outdir	output folder
bc_anno	barcode annotation, first column is cell id, second column is cell barcode sequence
max_mis	maximum mismatch allowed in barcode. (default: 1)
bam_tags	list defining BAM tags where mapping information is stored. <ul style="list-style-type: none"> <li>• "am": mapping status tag</li> <li>• "ge": gene id</li> <li>• "bc": cell barcode tag</li> <li>• "mb": molecular barcode tag</li> </ul>
mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
has_UMI	whether the protocol contains UMI (default: TRUE)
UMI_cor	correct UMI sequencing error: 0 means no correction, 1 means simple correction and merge UMI with distance 1. 2 means merge on both UMI alignment position match.
gene_fl	whether to remove low abundance genes. A gene is considered to have low abundance if only one copy of one UMI is associated with it.
nthreads	number of threads to use. (default: 1)

**Value**

no return

**Examples**

```
## Not run:
refer to the vignettes for the complete workflow, replace demultiplex and
count with single command:
...
sc_demultiplex_and_count(
  file.path(data_dir, "out.map.bam"),
  data_dir,
  barcode_annotation_fn,
  has_UMI = FALSE
)
...
## End(Not run)
```

---

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

---

## Description

Detect cell barcode and generate the barcode annotation

## Usage

```
sc_detect_bc(infq, outcsv, prefix = "CELL_", bc_len, max_reads = 1e+06,
             min_count = 10, number_of_cells = 10000, max_mismatch = 1,
             white_list_file = NULL)
```

## Arguments

infq	input fastq file, should be the output file of sc_trim_barcode
outcsv	output barcode annotation
prefix	the prefix of cell name (default: 'CELL_')
bc_len	the length of cell barcode, should be consistent with b11+b12 in sc_trim_barcode
max_reads	the maximum of reads processed (default: 1,000,000)
min_count	minimum counts to keep, barcode will be discarded if it has lower count. Default value is 10. This should be set according to max_reads.
number_of_cells	number of cells kept in result. (default: 10000)
max_mismatch	the maximum mismatch allowed. Barcodes within this number will be considered as sequence error and merged. (default: 1)
white_list_file	a file that list all the possible barcodes each row is a barcode sequence. the list for 10x can be found at: <a href="https://community.10xgenomics.com/t5/Data-Sharing/List-of-valid-cellular-barcodes/td-p/527">https://community.10xgenomics.com/t5/Data-Sharing/List-of-valid-cellular-barcodes/td-p/527</a> (default: NULL)

## Value

no return

## Examples

```
## Not run:
# `sc_detect_bc` should run before `sc_demultiplex` for
# Drop-seq or 10X protocols
sc_detect_bc("input.fastq", "output.cell_index.csv", bc_len=8)
sc_demultiplex(..., "output.cell_index.csv")

## End(Not run)
```

---

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

---

## Description

Map aligned reads to exon annotation. The result will be written into optional fields in bam file with different tags. Following this link for more information regarding to bam file format: <http://samtools.github.io/hts-specs>

The function can accept multiple bam file as input, if multiple bam file is provided and the 'bc\_len' is zero, then the function will use the barcode in the 'barcode\_vector' to insert into the 'bc' bam tag. So the length of 'barcode\_vector' and the length of 'inbam' should be the same. If this is the case then the 'max\_mis' argument in 'sc\_demultiplex' should be zero. If 'bc\_len' is larger than zero, then the function will still seek for barcode in fastq headers with given length. In this case each bam file is not treated as from a single cell.

## Usage

```
sc_exon_mapping(inbam, outbam, annofn, bam_tags = list(am = "YE", ge = "GE",
  bc = "BC", mb = "OX"), bc_len = 8, barcode_vector = "", UMI_len = 6,
  stnd = TRUE, fix_chr = FALSE, nthreads = 1)
```

## Arguments

inbam	input aligned bam file. can have multiple files as input
outbam	output bam filename
annofn	single string or vector of gff3 annotation filenames, data.frame in SAF format or GRanges object containing complete gene_id metadata column.
bam_tags	list defining BAM tags where mapping information is stored. <ul style="list-style-type: none"> <li>• "am": mapping status tag</li> <li>• "ge": gene id</li> <li>• "bc": cell barcode tag</li> <li>• "mb": molecular barcode tag</li> </ul>
bc_len	total barcode length
barcode_vector	a list of barcode if each individual bam is a single cell. (default: NULL). The barcode should be of the same length for each cell.
UMI_len	UMI length
stnd	TRUE to perform strand specific mapping. (default: TRUE)
fix_chr	TRUE to add 'chr' to chromosome names, MT to chrM. (default: FALSE)
nthreads	number of threads to use. (default: 1)

## Value

generates a bam file with exons assigned

**Examples**

```

data_dir="celseq2_demo"
ERCCanno_fn = system.file("extdata", "ERCC92_anno.gff3",
  package = "scPipe")
## Not run:
# for the complete workflow, refer to the vignettes
...
sc_exon_mapping(file.path(data_dir, "out.aln.bam"),
  file.path(data_dir, "out.map.bam"),
  ERCCanno_fn)
...

## End(Not run)

```

---

```

sc_gene_counting      sc_gene_counting

```

---

**Description**

Generate gene counts matrix with UMI deduplication

**Usage**

```
sc_gene_counting(outdir, bc_anno, UMI_cor = 2, gene_fl = FALSE)
```

**Arguments**

outdir	output folder containing sc_demultiplex output
bc_anno	barcode annotation comma-separated-values, first column is cell id, second column is cell barcode sequence
UMI_cor	correct UMI sequencing error: 0 means no correction, 1 means simple correction and merge UMI with distance 1. 2 means merge on both UMI alignment position match.
gene_fl	whether to remove low abundance genes. A gene is considered to have low abundance if only one copy of one UMI is associated with it.

**Value**

no return

**Examples**

```

data_dir="celseq2_demo"
barcode_annotation_fn = system.file("extdata", "barcode_anno.csv",
  package = "scPipe")
## Not run:
# refer to the vignettes for the complete workflow

```

```
...
sc_gene_counting(data_dir, barcode_annotation_fn)
...

## End(Not run)
```

---

sc_sample_data	<i>a small sample scRNA-seq counts dataset to demonstrate capabilities of scPipe</i>
----------------	--

---

### Description

This data set contains counts for high variable genes for 100 cells. The cells have different cell types. The data contains raw read counts. The cells are chosen randomly from 384 cells and they did not go through quality controls. The rows names are Ensembl gene ids and the columns are cell names, which is the well position in the 384 plates.

### Usage

```
sc_sample_data
```

### Format

a matrix instance, one row per gene.

### Value

NULL, but makes a matrix of count data

### Author(s)

Luyi Tian

### Source

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

### Examples

```
# use the example dataset to perform quality control
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
```

```
UMI_dup_info(sce) = UMI_duplication
sce = detect_outlier(sce)

plot_QC_pairs(sce)
```

---

sc_sample_qc	<i>quality control information for a small sample scRNA-seq dataset to demonstrate capabilities of scPipe.</i>
--------------	--

---

### Description

This data.frame contains cell quality control information for the 100 cells. For each cell it has:

- unaligned the number of unaligned reads.
- aligned\_unmapped the number of reads that aligned to genome but fail to map to any features.
- mapped\_to\_exon is the number of reads that mapped to exon.
- mapped\_to\_intron is the number of reads that mapped to intron.
- ambiguous\_mapping is the number of reads that mapped to multiple features. They are not considered in the following analysis.
- mapped\_to\_ERCC is the number of reads that mapped to ERCC spike-in controls.
- mapped\_to\_MT is the number of reads that mapped to mitochondrial genes.
- total\_count\_per\_cell is the number of reads that mapped to exon after UMI deduplication. In contrast, 'mapped\_to\_exon' is the number of reads mapped to exon before UMI deduplication.
- number\_of\_genes is the number of genes detected for each cells
- non\_ERCC\_percent is 1 - (percentage of ERCC reads). Reads are UMI deduplicated.
- non\_mt\_percent is 1 - (percentage of mitochondrial reads). Reads are UMI deduplicated.
- non\_ribo\_percent is 1 - (percentage of ribosomal reads). Reads are UMI deduplicated.

### Usage

```
sc_sample_qc
```

### Format

a data.frame instance, one row per cell.

### Value

NULL, but makes a data frame with cell quality control data.frame

### Author(s)

Luyi Tian



**Source**

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
head(QC_metrics(sce))
plot_mapping(sce,percentage=TRUE,dataname="sc_sample")
```

---

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

---

**Description**

Reformat fastq files so barcode and UMI sequences are moved from the sequence into the read name.

**Usage**

```
sc_trim_barcode(outfq, r1, r2 = NULL, read_structure = list(bs1 = -1, b11 = 0, bs2 = 6, b12 = 8, us = 0, ul = 6), filter_settings = list(rmlow = TRUE, rmN = TRUE, minq = 20, num bq = 2))
```

**Arguments**

outfq	the output fastq file, which reformat the barcode and UMI into the read name. Files ending in .gz will be automatically compressed.
r1	read one for pair-end reads. This read should contain the transcript.
r2	read two for pair-end reads, NULL if single read. (default: NULL)
read_structure	a list containing the read structure configuration: <ul style="list-style-type: none"> <li>• bs1: starting position of barcode in read one. -1 if no barcode in read one.</li> <li>• b11: length of barcode in read one, if there is no barcode in read one this number is used for trimming beginning of read one.</li> <li>• bs2: starting position of barcode in read two</li> <li>• b12: length of barcode in read two</li> <li>• us: starting position of UMI</li> <li>• ul: length of UMI</li> </ul>
filter_settings	A list contains read filter settings:

- rmlow whether to remove the low quality reads.
- rmN whether to remove reads that contains N in UMI or cell barcode.
- minq the minimum base pair quality that we allowed
- numbq the maximum number of base pair that have quality below numbq

### Details

Positions used in this function are 0-indexed, so they start from 0 rather than 1. The default read structure in this function represents CEL-seq paired-ended reads. This contains a transcript in the first read, a UMI in the first 6bp of the second read followed by a 8bp barcode. So the read structure will be : `list(bs1=-1, b11=0, bs2=6, b12=8, us=0, ul=6)`. `bs1=-1, b11=0` indicates negative start position and zero length for the barcode on read one, this is used to denote "no barcode" on read one. `bs2=6, b12=8` indicates there is a barcode in read two that starts at the 7th base with length 8bp. `us=0, ul=6` indicates a UMI from first base of read two and the length in 6bp.

For a typical Drop-seq experiment the read structure will be `list(bs1=-1, b11=0, bs2=0, b12=12, us=12, ul=8)`, which means the read one only contains transcript, the first 12bp in read two are cell barcode, followed by a 8bp UMI.

### Value

generates a trimmed fastq file named outfq

### Examples

```
data_dir="celseq2_demo"
## Not run:
# for the complete workflow, refer to the vignettes
...
sc_trim_barcode(file.path(data_dir, "combined.fastq"),
  file.path(data_dir, "simu_R1.fastq"),
  file.path(data_dir, "simu_R2.fastq"))
...
## End(Not run)
```

---

UMI_duplication	<i>UMI duplication statistics for a small sample scRNA-seq dataset to demonstrate capabilities of scPipe</i>
-----------------	--

---

### Description

This data.frame contains UMI duplication statistics, where the first column is the number of duplication, and the second column is the count of UMIs.

### Usage

```
sc_sample_qc
```

**Format**

a data.frame instance, one row per cell.

**Value**

NULL, but makes a data frame with UMI duplication statistics

**Author(s)**

Luyi Tian

**Source**

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

head(UMI_dup_info(sce))
```

---

UMI\_dup\_info

*Get or set UMI duplication results in a SingleCellExperiment object*

---

**Description**

Get or set UMI duplication results in a SingleCellExperiment object

**Usage**

```
UMI_dup_info(object)

UMI_dup_info(object) <- value

UMI_dup_info.sce(object)

## S4 method for signature 'SingleCellExperiment'
UMI_dup_info(object)

## S4 replacement method for signature 'SingleCellExperiment'
UMI_dup_info(object) <- value
```

**Arguments**

object            A [SingleCellExperiment](#) object.  
value            Value to be assigned to corresponding object.

**Value**

a dataframe of cell UMI duplication information  
A DataFrame of UMI duplication results.

**Author(s)**

Luyi Tian

**Examples**

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

head(UMI_dup_info(sce))
```

# Index

anno\_import, 3  
anno\_to\_saf, 3

calculate\_QC\_metrics, 4  
cell\_barcode\_matching, 5  
convert\_geneid, 7  
create\_processed\_report, 8  
create\_report, 8  
create\_sce\_by\_dir, 10

demultiplex\_info, 11  
demultiplex\_info, SingleCellExperiment-method  
(demultiplex\_info), 11  
demultiplex\_info.sce  
(demultiplex\_info), 11  
demultiplex\_info<- (demultiplex\_info),  
11  
demultiplex\_info<- , SingleCellExperiment-method  
(demultiplex\_info), 11  
detect\_outlier, 12

gene\_id\_type, 14  
gene\_id\_type, SingleCellExperiment-method  
(gene\_id\_type), 14  
gene\_id\_type.sce (gene\_id\_type), 14  
gene\_id\_type<- (gene\_id\_type), 14  
gene\_id\_type<- , SingleCellExperiment-method  
(gene\_id\_type), 14  
get\_ercc\_anno, 15  
get\_genes\_by\_GO, 15  
get\_read\_str, 16

organism (organism.sce), 17  
organism, SingleCellExperiment-method  
(organism.sce), 17  
organism.sce, 17  
organism<- , SingleCellExperiment-method  
(organism.sce), 17

plot\_demultiplex, 18  
plot\_mapping, 18

plot\_QC\_pairs, 19  
plot\_UMI\_dup, 20

QC\_metrics, 21  
QC\_metrics, SingleCellExperiment-method  
(QC\_metrics), 21  
QC\_metrics.sce (QC\_metrics), 21  
QC\_metrics<- (QC\_metrics), 21  
QC\_metrics<- , SingleCellExperiment-method  
(QC\_metrics), 21

remove\_outliers, 22

sc\_correct\_bam\_bc, 23  
sc\_count\_aligned\_bam, 24  
sc\_demultiplex, 24, 25, 26  
sc\_demultiplex\_and\_count, 26  
sc\_detect\_bc, 28  
sc\_exon\_mapping, 24, 29  
sc\_gene\_counting, 24, 26, 30  
sc\_sample\_data, 31  
sc\_sample\_qc, 32  
sc\_trim\_barcode, 33  
scPipe, 22  
scPipe-package (scPipe), 22  
SingleCellExperiment, 10, 12, 14, 17, 21, 36

UMI\_dup\_info, 35  
UMI\_dup\_info, SingleCellExperiment-method  
(UMI\_dup\_info), 35  
UMI\_dup\_info.sce (UMI\_dup\_info), 35  
UMI\_dup\_info<- (UMI\_dup\_info), 35  
UMI\_dup\_info<- , SingleCellExperiment-method  
(UMI\_dup\_info), 35  
UMI\_duplication, 34