Package: CleanUpRNAseq (via r-universe)

November 5, 2026

Type Package

Title Detect and Correct Genomic DNA Contamination in RNA-seq Data

Version 1.6.0

Description RNA-seq data generated by some library preparation methods, such as rRNA-depletion-based method and the SMART-seq method, might be contaminated by genomic DNA (gDNA), if DNase I disgestion is not performed properly during RNA preparation. CleanUpRNAseq is developed to check if RNA-seq data is suffered from gDNA contamination. If so, it can perform correction for gDNA contamination and reduce false discovery rate of differentially expressed genes.

License GPL-3

BugReports https://github.com/haibol2016/CleanUpRNAseq/issues

Depends R (>= 4.4.0)

Imports AnnotationFilter, BiocGenerics, Biostrings, BSgenome, DESeq2, edgeR, ensembldb, Seqinfo, GenomicRanges, ggplot2, ggrepel, graphics, grDevices, KernSmooth, limma, methods, pheatmap, qsmooth, R6, RColorBrewer, Rsamtools, Rsubread, reshape2, SummarizedExperiment, stats, tximport, utils

Suggests BiocStyle, BSgenome.Hsapiens.UCSC.hg38, EnsDb.Hsapiens.v86, ggplotify, knitr, patchwork, R.utils, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews QualityControl, Sequencing, GeneExpression

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Collate 'CleanUpRNAseq-package.R' 'R6_class.R' 'misc.R' 'check_gDNA.R' 'correction_methods.R' 'data.R'

Config/testthat/edition 3

Config/pak/sysreqs cmake make libbz2-dev libicu-dev liblzma-dev libpng-dev libuv1-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

Repository Bioconductor 3.23

Date/Publication 2026-04-28 13:03:27 UTC

RemoteUrl https://github.com/bioc/CleanUpRNAseq

RemoteRef RELEASE_3_23

RemoteSha 769640e8832c7c57d1428395dd9e2973048aabcd

Contents

calc_gene_gc . . . 2
calc_region_gc . . . 4
correct_GC . . . 6
correct_global . . . 7
correct_IR . . . 8
correct_stranded . . . 10
create_summarizedcounts . . . 11
feature_counts_list . . . 12
gene_GC . . . 12
get_region_stat . . . 13
get_saf . . . 14
intergenic_GC . . . 15
plot_assignment_stat . . . 16
plot_expr_distr . . . 17
plot_gene_content . . . 18
plot_pca_heatmap . . . 20
plot_read_distr . . . 21
plot_sample_corr . . . 22
salmon_quant . . . 23
salmon_tximport . . . 24
style_BSgenome . . . 25
summarize_reads . . . 27
SummarizedCounts . . . 29

Index **38**

`calc_gene_gc` *Calculate GC content of genes*

Description

Calculate GC content of all genes based on sequences of collapsed exons of each gene.

Usage

```
calc_gene_gc(ensdb_sqlite = NULL, BSgenome = NULL, batch_size = 2000)
```

Arguments

ensdb_sqlite	A character(1) specifying a path to an SQLite file to store the an object of the ensembldb::EnsDb or an object of the ensembldb::EnsDb.
BSgenome	An object of BSgenome::BSgenome. Make sure the chromosome names (aka seqnames) in the BSgenome object match those in the ensembldb::EnsDb specified by *ensdb_sqlite*.
batch_size	An integer(1) vector, specifying how many regions are processed each batch.

Value

A data frame contains two columns: gc_content and width.

gc_content GC contents (proportion) of genes

width widths of genes

Examples

```
require("ensembldb")
ucsc_BSgenome <- BSgenome.Hsapiens.UCSC.hg38
ucsc_seqnames <-
    seqnames(ucsc_BSgenome)[!grepl("_alt|fix|hap\\d+",
                            seqnames(ucsc_BSgenome))]

## Wierd: BSgenome.Hsapiens.UCSC.hg19 has both chrM and chrMT for
## mitochondrial genome. renomve chrMT.

if (all(c("chrM", "chrMT") %in% ucsc_seqnames)) {
    ucsc_seqnames <- ucsc_seqnames[!ucsc_seqnames %in% "chrMT"]
}

ensembl_seqnames <- gsub(
    "^chr", "",
    gsub(
        "chrM$", "MT",
        gsub(
            "v", ".",
            gsub("_random|chr[^_]+_", "", ucsc_seqnames)
        )
    )
)
## for BSgenome.Hsapiens.UCSC.hg19, scaffold seqnames start with lower case,
## should be changed to upper case. For example, change "gl000231" to
## "GL000231". Add a suffix ".1" to these scaffold seqnames.

seqname_alias <- data.frame(ucsc = ucsc_seqnames,
                            ensembl = ensembl_seqnames)

bsgenome <- style_BSgenome(
    UCSC_BSgenome = ucsc_BSgenome,
    genome_version = "GRCh38",
    seqname_alias = seqname_alias
```

```
)


tmp_dir <- tempdir()
gtf <- system.file("extdata", "example.gtf.gz",
                   package = "CleanUpRNAseq")
hs_ensdb_sqlite <-
    ensembldb::ensDbFromGtf(
        gtf = gtf,
        outfile = file.path(tmp_dir, "EnsDb.hs.v110.sqlite"),
        organism = "Homo_Sapiens",
        genomeVersion = "GRCh38",
        version = 110
    )
gene_gc <- calc_gene_gc(
    ensdb_sqlite = hs_ensdb_sqlite,
    BSgenome = bsgenome,
    batch_size = 2000
)
```

`calc_region_gc` *Calculate GC content of genomic regions*

Description

Calculate GC content of genomic regions

Usage

```
calc_region_gc(region = NULL, BSgenome = NULL, batch_size = 2000)
```

Arguments

`region`	A data frame containing the columns: "Chr", "Start", "End", and "Strand", such as a data frame contained in a sublist named *intergenic_region* from the output of the `get_saf()` function, or a object of GenomicRanges::GRangesList or GenomicRanges::GRanges.
`BSgenome`	An object of BSgenome::BSgenome. Make sure the chromosome names (aka seqnames) in the BSgenome object match those in the *region_saf* data frame and the *region_gr* object.
`batch_size`	An integer(1) vector, specifying how many regions are processed each batch.

Value

A data.frame contains two columns: gc_content and width.

gc_content GC contents (proportion) of genomic regions

width widths of genomic regions

Examples

```
ucsc_BSgenome <- BSgenome.Hsapiens.UCSC.hg38
ucsc_seqnames <-
    seqnames(ucsc_BSgenome)[!grepl(
        "_alt|fix|hap\\d+",
        seqnames(ucsc_BSgenome)
    )]

## Wierd: BSgenome.Hsapiens.UCSC.hg19 has both chrM and chrMT for
## mitochondrial genome. renomve chrMT.

if (all(c("chrM", "chrMT") %in% ucsc_seqnames)) {
    ucsc_seqnames <- ucsc_seqnames[!ucsc_seqnames %in% "chrMT"]
}

ensembl_seqnames <- gsub(
    "^chr", "",
    gsub(
        "chrM$", "MT",
        gsub(
            "v", ".",
            gsub("_random|chr[^_]+_", "", ucsc_seqnames)
        )
    )
)
## for BSgenome.Hsapiens.UCSC.hg19, scaffold seqnames start with lower case,
## should be changed to upper case. For example, change "gl000231" to
## "GL000231". Add a suffix ".1" to these scaffold seqnames.

seqname_alias <- data.frame(ucsc = ucsc_seqnames,
                            ensembl = ensembl_seqnames)
bsgenome <- style_BSgenome(
    UCSC_BSgenome = BSgenome.Hsapiens.UCSC.hg38,
    genome_version = "GRCh38",
    seqname_alias = seqname_alias
)
region <- data.frame(
    GeneID = as.character(seq_len(10)),
    Chr = rep("1", 10),
    Start = 100000 * (seq_len(10)),
    End = 100000 * (seq_len(10)) + 1000,
    Strand = rep("+", 10)
)

gc_contents <- calc_region_gc(
    region = region,
    BSgenome = bsgenome,
    batch_size = 2000
)
```

correct_GC *Correct DNA contamination considering GC-bias effect*

Description

Correct DNA contamination considering GC-bias effect on fragment amplification. Intergenic regions are binned based on their GC content ranging from 0 to 100%, with a bin size of 5%. Per gene DNA contamination is estimated as the product of count per base in a GC content matching bin of intergenic regions and the total collapsed exons of a gene and is subtracted away from the gene count matrix.

Usage

```
correct_GC(
  SummarizedCounts = NULL,
  gene_gc = NULL,
  intergenic_gc = NULL,
  plot = FALSE
)
```

Arguments

SummarizedCounts	An object of SummarizedCounts..
gene_gc	A data frame or matrix with two columns: gc_content in proportion between 0 and 1, and width in basepair, containing GC-content and total exon lengths of each gene. An output of the `calc_gene_gc()` function.
intergenic_gc	A data frame or matrix with two columns: gc_content in proportion between 0 and 1, and width in basepair, containing GC-content and lengths of each intergenic region, such as an output from the `calc_region_gc()` function.
plot	A logical(1) vector, specifying whether to output a panel of scatter plot showing a bi-variate distribution of GC content and count per base of intergenic regions in each sample. Default is TRUE.

Value

A data frame containing corrected count for each gene (row) of each sample (column).

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
```

```
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)

data("gene_GC")
data("intergenic_GC")
gc_bias_corrected_counts <-
    correct_GC(
        SummarizedCounts = sc,
        gene_gc = gene_GC,
        intergenic_gc = intergenic_GC,
        plot = FALSE
    )
```

`correct_global` *Global correction for DNA contamination*

Description

Correct for DNA contamination in RNA-seq data using the 2.5 times of median counts per base of intergenic regions with at least one count.

Usage

```
correct_global(SummarizedCounts = NULL, lambda = 1)
```

Arguments

`SummarizedCounts`
: An object of SummarizedCounts..

`lambda` A positive number specifying how many times of the median read coverage of non-zero count intergenic regions used as an estimate of DNA contamination. The default of *lambda* is 1,but it could be adjusted based on the gene-level count distributions of the resulting corrected count table output by the `plot_read_distr()` function. A value between 1 and 3 can be tried. Ideally the distributions of all samples from a given condition should be very similar.

Value

A matrix containing an RNA-seq count table corrected for DNA contamination, with rows for genes and columns for samples.

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)
corrected_counts <- correct_global(SummarizedCounts = sc)
```

`correct_IR` *Correct gene expression using a linear model*

Description

Correct gene expression with IR% as a co-variate in linear model in the limma framework.

Usage

```
correct_IR(SummarizedCounts = NULL, design = NULL)
```

Arguments

SummarizedCounts	An object of SummarizedCounts..
design	A design matrix with rows corresponding to samples and columns to coefficients to be estimated. See `limma::lmFit()` and Law et al. 2020, F1000Research, doi: 10.12688/f1000research.27893.1.

Value

A numeric matrix of normalized expression values on the log2 scale, corrected for gDNA contamination, and batch (if any).

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)
assigned_per_region <- get_region_stat(SummarizedCounts = sc)
design = model.matrix(~ group + batch +IR_rate, data = sc$col_data)
ir_corrected <- correct_IR(sc, design)
```

correct_stranded *Correct for gDNA contamination in stranded libraries*

Description

Correct for gDNA contamination in stranded libraries based on Salmon quantitation using the real and opposite library strandedness information

Usage

```
correct_stranded(SummarizedCounts = NULL)
```

Arguments

SummarizedCounts
: An object of SummarizedCounts..

Value

A list of of matrices of gene-level abundance, counts, and length. See `tximport::tximport()`. The count matrix is corrected for DNA contamination and rounded into integers.

abundance A numeric matrix containing corrected abundance (TPM) for each gene of each sample

counts An *integer* matrix containing *corrected* read count for each gene of each sample

length A numeric matrix containing length (bp) for each gene of each sample

Examples

```
lib_strand <- 1
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)
```

```
data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)
stranded_correction <- correct_stranded(SummarizedCounts = sc)
```

`create_summarizedcounts` *Create an object of SummarizedCounts*

Description

Set up an analysis by creating an object of SummarizedCounts, which is used to store summary output from featureCounts and tximport, and store gDNA-corrected expression data. This is a convenience wrapper for `SummarizedCounts$new()`

Usage

```
create_summarizedcounts(lib_strand = 0, colData = NULL)
```

Arguments

`lib_strand`	An integer(1), specifying the library's strandedness. It has three possible values: • 0: unstranded, the default; • 1: stranded, read 1 (or single-end read) comes from the forward strand; • 2: reversely stranded, read 1 (or single-end read) comes from the reverse strand For more details, See https://sailfish.readthedocs.io/en/master/library_type.html.
`colData`	A data frame with rows corresponding to samples. For unstranded RNA-seq data, it at least contains the following columns: `sample_name`, `BAM_file`, `group`, `salmon_quant_file`, and `batch` if the data were generated in more than one batches. For stranded RNA-seq data, an extra column, `salmon_quant_file_opposite_strand`, should be included.

Value

An object of SummarizedCounts, which is used to store summary output from featureCounts and tximport, and store gDNA-corrected expression data.

Examples

```
in_dir <- system.file("extdata", package = "CleanUpRNAseq")
BAM_file <- dir(in_dir, ".bam$", full.name = TRUE)
salmon_quant_file <- dir(in_dir, ".sf$", full.name = TRUE)
sample_name = gsub(".+/(.+?).srt.bam", "\\1", BAM_file)
salmon_quant_file_opposite_strand <- salmon_quant_file
```

```
col_data <- data.frame(sample_name = sample_name,
                       BAM_file = BAM_file,
                       salmon_quant_file = salmon_quant_file,
                       salmon_quant_file_opposite_strand =
                           salmon_quant_file_opposite_strand,
                       group = c("CD1N", "CD1P"))

sc <- create_summarizedcounts(lib_strand = 0, colData = col_data)
```

---

feature_counts_list *GC content and lengths of 2000 intergenic regions*

Description

GC content and lengths of 2000 human intergenic regions calculated using the `calc_region_gc()` function.

Usage

```
data(feature_counts_list)
```

Format

"intergenic_region", "intronic_region", "rRNA", "mitochondrion", "gtf", "chloroplast" (plant only). For genomic features, gene, exon, intron, rRNA, mitochondrion, and chloroplast, only the `stat` elements of the `Rsubread::featureCounts()` output is stored. For intergenic region, the `stat`, `annotation` and `counts` elements are stored; for GTF-based summarization, the `stat` and `counts` elements are stored.

---

gene_GC *GC content and lengths of 2000 human genes*

Description

GC content and lengths of 2000 human Ensembl genes. For each gene exons are collapsed to get disjoint meta-exons and GC content is calculated for each meta-exon. A exon length-weighted GC content is calculated for each gene using the `calc_gene_gc()` function.

Usage

```
data(gene_GC)
```

Format

A data frame with 2000 rows and 2 columns:

gc_content metagene-level GC content, values are between 0 and 1

width length of metagenes in base pair(bp)

Source

Homo_sapiens.GRCh38.110.gtf.gz

---

get_region_stat *Calculate read distribution over different types of genomic features*

Description

Calculate read distribution over different types of genomic features: genes, exons, introns, intergenic regions,rRNA regions, and organelle genome(s).

Usage

```
get_region_stat(SummarizedCounts = NULL)
```

Arguments

`SummarizedCounts`
: An object of SummarizedCounts..

Value

A data frame described as below.

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf
```

```
## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)

assigned_per_region <- get_region_stat(SummarizedCounts = sc)
assigned_per_region
```

`get_saf` *Generate a SAF file for genomic features*

Description

Generate a SAF (simplified annotation format) file for genomic features: genicregions, intergenic regions, exonic regions, intronic regions, rRNA genes, mitochondrial genome, and chloroplast genome (only for plants).

Usage

```
get_saf(
  ensdb_sqlite = NULL,
  bamfile = NULL,
  mitochondrial_genome = c("MT", "chrM"),
  chloroplast_genome = c("chrPltd", "Pltd")
)
```

Arguments

`ensdb_sqlite`	A character(1) specifying a path to an SQLite file to store the an object of the ensembldb::EnsDb or an object of the ensembldb::EnsDb.
`bamfile`	A character(1), a path to a BAM file for the experiment of interest. The BAM file is used to extract genome information: chromosome/scaffold names and lengths.
`mitochondrial_genome`	A character(1), mitochondrial genome name in the EnsDb database (ie, the mitochondrial name in column 1 of the GTF file used to generate the EnsDb SQLite database).
`chloroplast_genome`	A character(1), chloroplast genome name in the EnsDb database (ie, the chloroplast name in column 1 of the GTF file used to generate the EnsDb SQLite database). This is only relevant for plants.

Value

A list of data frames containing SAF for genomic features: genic regions, intergenic regions, exonic regions, intronic regions, rRNA genes, mitochondrial genome, chloroplast genome (only for plants), respectively.

gene a data frame containing a SAF for genes

exon a data frame containing a SAF for exons

intergenic_region a data frame containing a SAF for intergenic regions

intronic_region a data frame containing a SAF for intronic region

rRNA a data frame containing a SAF for rRNA exons

mitochondrion a data frame containing a SAF for the mitochodrion

chloroplast (optional) a data frame containing a SAF for chloroplast, plnat only

Examples

```
require("ensembldb")
tmp_dir <- tempdir()
gtf <- system.file("extdata", "example.gtf.gz",
                   package = "CleanUpRNAseq")
hs_ensdb_sqlite <-
    ensembldb::ensDbFromGtf(
        gtf = gtf,
        outfile = file.path(tmp_dir, "EnsDb.hs.v110.sqlite"),
        organism = "Homo_Sapiens",
        genomeVersion = "GRCh38",
        version = 110
    )
bam_file <- system.file("extdata", "K084CD7PCD1N.srt.bam",
    package = "CleanUpRNAseq"
)
saf_list <- get_saf(
    ensdb_sqlite = hs_ensdb_sqlite,
    bamfile = bam_file,
    mitochondrial_genome = "MT"
)
```

---

intergenic_GC *GC content and lengths of 2000 intergenic regions*

Description

GC content and lengths of 2000 human intergenic regions calculated using the `calc_region_gc()` function.

Usage

```
data(intergenic_GC)
```

Format

A data frame with 2000 rows and 2 columns:

gc_content metagene-level GC content, values are between 0 and 1

width length of metagenes in base pair(bp)

Source

Homo_sapiens.GRCh38.110.gtf.gz

`plot_assignment_stat` *Visualize assignment statistics of reads/fragments by featureCounts*

Description

Visualize assignment statistics of reads/fragments by featureCounts

Usage

```
plot_assignment_stat(SummarizedCounts = NULL)
```

Arguments

`SummarizedCounts`
: An object of SummarizedCounts..

Value

A ggplot object, showing percentages and number of fragments in each assignment category as determined by `Rsubread::featureCounts()` based on a GTF.

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                         package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf
```

```
## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)

plot_assignment_stat(SummarizedCounts = sc)
```

`plot_expr_distr` *Visualize expression distribution*

Description

Compare expression distribution by boxplot, density plot and empirical cumulative distribution plot.

Usage

```
plot_expr_distr(
  SummarizedCounts = NULL,
  normalization = c("DESeq2", "qsmooth", "none")
)
```

Arguments

`SummarizedCounts`	An object of SummarizedCounts..
`normalization`	A character(1) vector, specifying a between-sample normalization methods: DESeq2's median of ratios method, smooth quantile normalization method `qsmooth::qsmooth()`, or none.

Value

A list of 3 ggplot objects.

box_plot boxplots showing DESeq2-normalized gene-level count distribution, on a log scale

density_plot density plots showing DESeq2-normalized gene-level count distribution, on a log scale

ecd_plot plots showing empricial cumulative distribution of fraction of genes with CPM greater than or equal to a given CPM on a log scale

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)

wrap_plots(plot_expr_distr(SummarizedCounts = sc,
                normalization = "DESeq2"), ncol = 1)
```

---

plot_gene_content *Check the percentage of genes with counts greater than minimal CPM*

Description

Check the percentage of genes with counts greater than minimal CPM

Usage

```
plot_gene_content(SummarizedCounts = NULL, min_cpm = 1, min_tpm = 1)
```

Arguments

SummarizedCounts	An object of SummarizedCounts..
min_cpm	A numeric(1), minimal CPM threshold.
min_tpm	A numeric(1), minimal TPM threshold.

Details

The axis title contains unicode, so please output the plot in the svg format using the svglite package, instead of `grDevices::pdf()`, for high- resolution plot.

Value

A ggplot object if `counts` is not `NULL`, showing percentages of genes with counts above the user-specified minimal CPM (count per million) in each sample. Or a ggplot object of two panels if both `counts` and `abundance` are not `NULL`, showing percentages of genes with counts above the user-specified minimal CPM and minimal TPM (transcript per million) in each sample.

A ggplot object.

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)
plot_gene_content(
    SummarizedCounts = sc,
    min_cpm = 1,
    min_tpm =1
)
```

`plot_pca_heatmap` *Check sample similarity and variation*

Description

Perform sample-level exploratory analysis of RNA-seq data, generating heatmap showing sample distances and PCA plot showing sample variations. Internally, DESeq2 is used for vst transformation of count data.

Usage

```
plot_pca_heatmap(SummarizedCounts = NULL, silent = TRUE)
```

Arguments

`SummarizedCounts`	An object of SummarizedCounts..
`silent`	A logical(1), specify whether to draw the plot. It is useful to set it to FALSE useful when using the gtable output.

Value

A list of a ggplot object and a `gtable::gtable()` object.

pca A *ggplot* object containing the PCA score plot showing sample similarity

heatmap A *gtable* object containing the heatmap showing pairwise sample distances

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                         package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)
```

```
data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)
p<- plot_pca_heatmap(SummarizedCounts = sc,
                     silent = TRUE)
wrap_plots(p[["pca"]], as.ggplot(p[["heatmap"]]), ncol = 1)
```

`plot_read_distr` *Visualize read distribution among different genomic regions*

Description

Generate ggplot plots showing percentages of fragments assigned to different type of genomic features: genes, exons, introns, intergenic regions, rRNA regions, and organelle genome(s).

Usage

```
plot_read_distr(assigned_per_region = NULL)
```

Arguments

`assigned_per_region`
: A data frame, outputted by `get_region_stat()`.

Value

A ggplot object, showing percentages of fragments assigned to different genomic features, such as genic regions, intergenic regions, exonic regions, intronic regions, rRNA genes, mitochondrial genome, chloroplast genome (only for plants)

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
```

```
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)

assigned_per_region <- get_region_stat(SummarizedCounts = sc)
p <- plot_read_distr(assigned_per_region)
p
```

`plot_sample_corr` *Visualize sample correlation*

Description

Generate a panel of plots based on a count table, with the diagonal showing the sample names, the lower triangle showing smoothed scatterplots for gene expression of pairwise samples, and the upper triangle showing Pearson correlation coefficients of gene expression of pairwise samples.

Usage

```
plot_sample_corr(SummarizedCounts = NULL)
```

Arguments

`SummarizedCounts`
: An object of SummarizedCounts..

Value

NULL. A plot with pairwise scatter plots and Pearson correlation coefficients is generated. When the sample size is big, save it as tiff file.

Examples

```
lib_strand <- 0
col_data_f <- system.file("extdata", "example.colData.txt",
                          package = "CleanUpRNAseq")
col_data <- read.delim(col_data_f, as.is = TRUE)
```

```
## create fake bam files
tmp_dir <- tempdir()
bamfiles <- gsub(".+/", "", col_data$BAM_file)
null <- lapply(file.path(tmp_dir, bamfiles), file.create)
## create fake quant.sf files
quant_sf <- file.path(tmp_dir, gsub(".srt.bam$",
                                    "quant.sf",
                                    bamfiles))
null <- lapply(quant_sf, file.create)
col_data$BAM_file <- file.path(tmp_dir, bamfiles)
col_data$salmon_quant_file <- quant_sf

## pretend this is stranded RA=NA-seq data
col_data$salmon_quant_file_opposite_strand <- quant_sf

sc <- create_summarizedcounts(lib_strand, col_data)

data("feature_counts_list")
data("salmon_quant")

sc$set_feature_counts(feature_counts_list)
sc$set_salmon_quant(salmon_quant)
sc$set_salmon_quant_opposite(salmon_quant)
plot_sample_corr(SummarizedCounts = sc)
```

`salmon_quant` *GC content and lengths of 2000 intergenic regions*

Description

GC content and lengths of 2000 human intergenic regions calculated using the `calc_region_gc()` function.

Usage

```
data(salmon_quant)
```

Format

A list of three elements:

abundance A numeric matrix containing abundance (TPM) for each gene of each sample

counts A numeric matrix containing read count (fraction) for each gene of each sample

length A numeric matrix containing length (bp) for each gene of each sample

`salmon_tximport` *Import Salmon quantification output into R*

Description

Import Salmon quantification output into R using tximport

Usage

```
salmon_tximport(
  SummarizedCounts = NULL,
  ensdb_sqlite = NULL,
 filtered_gene_biotypes = c("artifact", "TEC", "miRNA", "tRNA", "misc_RNA", "Mt_rRNA",
  "Mt_tRNA", "rRNA", "rRNA_pseudogene", "scaRNA", "scRNA", "snoRNA", "snRNA", "sRNA",
    "vault_RNA", "TR_V_pseudogene", "IG_C_pseudogene", "IG_J_pseudogene",
    "IG_V_pseudogene", "IG_pseudogene", "TR_J_pseudogene", "TR_V_pseudogene")
)
```

Arguments

`SummarizedCounts`
: An object of SummarizedCounts.

`ensdb_sqlite`
: An object of the ensembldb::EnsDb or a character(1) vector, specifying a path to an SQLite database for an object of the ensembldb::EnsDb.

`filtered_gene_biotypes`
: A character(n) vector,specifying the biotypes of genes which will not be considered for downstream gene expression analysis. By default, genes of the following biotypes are excluded: "artifact","TEC","miRNA","tRNA", "misc_RNA", "Mt_rRNA", "Mt_tRNA", "rRNA", "rRNA_pseudogene", "scaRNA", "scRNA", "snoRNA", "snRNA", "sRNA", "vault_RNA", "TR_V_pseudogene", "IG_C_pseudogene", "IG_J_pseudogene", "IG_V_pseudogene", "IG_pseudogene", "TR_J_pseudogene", "TR_V_pseudogene", because their transcripts are too short (< 200 nt) or not likely expressed at all.

Value

An object of SummarizedCounts with the feature_counts field populated by a list of of matrices of gene-level abundance, counts, and length outputted by `tximport::tximport()`, as described in the folowing:

abundance A numeric matrix containing abundance (TPM) for each gene of each sample

counts A numeric matrix containing read count (fraction) for each gene of each sample

length A numeric matrix containing length (bp) for each gene of each sample

Examples

```
require("ensembldb")
in_dir <- system.file("extdata", package = "CleanUpRNAseq")
BAM_file <- dir(in_dir, ".bam$", full.name = TRUE)
salmon_quant_file <- dir(in_dir, ".sf$", full.name = TRUE)
sample_name = gsub(".+/(.+?).srt.bam", "\\1", BAM_file)
salmon_quant_file_opposite_strand <- salmon_quant_file
col_data <- data.frame(sample_name = sample_name,
                       BAM_file = BAM_file,
                       salmon_quant_file = salmon_quant_file,
                       salmon_quant_file_opposite_strand =
                           salmon_quant_file_opposite_strand,
                       group = c("CD1N", "CD1P"))

sc <- create_summarizedcounts(lib_strand = 0, colData = col_data)

tmp_dir <- tempdir()
gtf <- system.file("extdata", "example.gtf.gz",
                   package = "CleanUpRNAseq")
hs_ensdb_sqlite <-
    ensembldb::ensDbFromGtf(
        gtf = gtf,
        outfile = file.path(tmp_dir, "EnsDb.hs.v110.sqlite"),
        organism = "Homo_Sapiens",
        genomeVersion = "GRCh38",
        version = 110
    )
salmon_counts <- salmon_tximport(
    SummarizedCounts = sc,
    ensdb_sqlite = hs_ensdb_sqlite
)
```

---

style_BSgenome *Convert a BSgenome from the UCSC to the Ensembl style*

Description

Convert a BSgenome from the UCSC style to the Ensembl style, changing "chrM" to "MT", removing the "chr" prefix from chromosome/scaffold seqnames, only keeping seqnames in the primary genome assembly, and changing the genome version name form the UCSC name to the Ensembl name, such as from "hg38" to "GRCh38".

Usage

```
style_BSgenome(
  UCSC_BSgenome = NULL,
  genome_version = "GRCh38",
 seqname_alias = data.frame(ucsc = paste0("chr", c(seq_len(22), "X", "Y", "M")), ensembl
```

```
    = c(seq_len(22), "X", "Y", "MT"))
)
```

Arguments

UCSC_BSgenome	An object of the BSgenome::BSgenome in the UCSC style, such as BSgenome.Hsapiens.UCSC.hg38.
genome_version	A character(1), specifying the genome version, such as "GRCh38".
seqname_alias	A data frame or a tab-delimited file to a data frame, with two columns: ucsc and ensembl for UCSC-style seqnames and Ensembl-style seqnames, respectively.

Value

An object of the BSgenome::BSgenome in the Ensembl style, such as BSgenome.Dvirilis.Ensembl.dvircaf1.

Examples

```
ucsc_BSgenome <- BSgenome.Hsapiens.UCSC.hg38
ucsc_seqnames <-
    seqnames(ucsc_BSgenome)[!grepl("_alt|fix|hap\\d+",
                            seqnames(ucsc_BSgenome))]

## Wierd: BSgenome.Hsapiens.UCSC.hg19 has both chrM and chrMT for
## mitochondrial genome. renomve chrMT.

if (all(c("chrM", "chrMT") %in% ucsc_seqnames)) {
    ucsc_seqnames <- ucsc_seqnames[!ucsc_seqnames %in% "chrMT"]
}

ensembl_seqnames <- gsub(
    "^chr", "",
    gsub(
        "chrM$", "MT",
        gsub(
            "v", ".",
            gsub("_random|chr[^_]+_", "", ucsc_seqnames)
        )
    )
)
## for BSgenome.Hsapiens.UCSC.hg19, scaffold seqnames start with lower case,
## should be changed to upper case. For example, change "gl000231" to
## "GL000231". Add a suffix ".1" to these scaffold seqnames.

seqname_alias <- data.frame(ucsc = ucsc_seqnames,
                            ensembl = ensembl_seqnames)

bsgenome <- style_BSgenome(
    UCSC_BSgenome = ucsc_BSgenome,
    genome_version = "GRCh38",
    seqname_alias = seqname_alias
)
```

summarize_reads *Summarize reads for different genomic features*

Description

Summarize reads in alignment files, SAM or BAM, to different genomic regions, such as genic regions, intergenic regions, exonic regions, intronic regions, rRNA genes, mitochrondrial genome, chloroplast genome (only for plants), and gene-level exonic regions using `Rsubread::featureCounts()`.

Usage

```
summarize_reads(
  SummarizedCounts = NULL,
  isPairedEnd = TRUE,
  allowMultiOverlap = FALSE,
  countMultiMappingReads = TRUE,
  fraction = TRUE,
  minMQS = 0,
  saf_list = NULL,
  gtf = NULL,
  threads = 1,
  verbose = FALSE
)
```

Arguments

SummarizedCounts	An object of SummarizedCounts.
isPairedEnd	A logical(1), whether the RNA-seq data is paired-end.
allowMultiOverlap	A logical(1), indicating if a read is allowed to be assigned to more than one feature (or meta-feature) if it is found to overlap with more than one feature (or meta-feature). FALSE by default. A read (or read pair) will be assigned to the feature (or meta-feature) that has the largest number of overlapping bases, if the read (or read pair) overlaps with multiple features (or meta-features).
countMultiMappingReads	A logical(1), indicating if multi-mapping reads/fragments should be counted, TRUE by default. 'NH' tag is used to located multi-mapping reads in the input BAM/SAM files.
fraction	A logical(1) indicating if fractional counts are produced for multi-mapping reads and/or multi-overlapping reads. FALSE by default.
minMQS	An integer(1), giving the minimum mapping quality score a read must satisfy in order to be counted. For paired-end reads, at least one end should satisfy this criteria. 0 by default.
saf_list	A list of data frames containing annotation in the SAF format, such as the output of the `get_saf()` function.

gtf	A character(1), specifying a path to a GTF file.
threads	An integer(1), number of threads for Rsubread::featureCounts() calling.
verbose	A logical(1) vector, indicating if verbose information for debugging will be generated. This may include information such as unmatched chromosomes/contigs between reads and annotation.

Value

An object of SummarizedCounts with the feature_counts field populated by modified output from the Rsubread::featureCounts() function for each type of genomic features.

Examples

```
tmp_dir <- tempdir()
in_dir <- system.file("extdata", package = "CleanUpRNAseq")
gtf.gz <- dir(in_dir, ".gtf.gz$", full.name = TRUE)
gtf <- file.path(tmp_dir, gsub("\\.gz", "", basename(gtf.gz)))
R.utils::gunzip(gtf.gz, destname= gtf,
                overwrite = TRUE, remove = FALSE)

in_dir <- system.file("extdata", package = "CleanUpRNAseq")
BAM_file <- dir(in_dir, ".bam$", full.name = TRUE)
salmon_quant_file <- dir(in_dir, ".sf$", full.name = TRUE)
sample_name = gsub(".+/(.+?).srt.bam", "\\1", BAM_file)
salmon_quant_file_opposite_strand <- salmon_quant_file
col_data <- data.frame(sample_name = sample_name,
                       BAM_file = BAM_file,
                       salmon_quant_file = salmon_quant_file,
                       salmon_quant_file_opposite_strand =
                           salmon_quant_file_opposite_strand,
                       group = c("CD1N", "CD1P"))

sc <- create_summarizedcounts(lib_strand = 0, colData = col_data)

bam_file <- system.file("extdata", "K084CD7PCD1N.srt.bam",
    package = "CleanUpRNAseq"
)

tmp_dir <- tempdir()
gtf <- system.file("extdata", "example.gtf.gz",
                   package = "CleanUpRNAseq")
hs_ensdb_sqlite <-
    ensembldb::ensDbFromGtf(
        gtf = gtf,
        outfile = file.path(tmp_dir, "EnsDb.hs.v110.sqlite"),
        organism = "Homo_Sapiens",
        genomeVersion = "GRCh38",
        version = 110
    )
saf_list <- get_saf(
    ensdb_sqlite = hs_ensdb_sqlite,
    bamfile = bam_file,
```

```
    mitochondrial_genome = "MT"
)

counts_list <- summarize_reads(
    SummarizedCounts = sc,
    saf_list = saf_list,
    gtf = gtf
)
```

SummarizedCounts *SummarizedCounts Object*

Description

A class for storing and retrieving summarized RNA-seq data.

Format

An R6 class

Constructors

Create an object of SummarizedCounts by setting `lib_strand` and `col_data`:

```
x <- SummarizedCounts$new(lib_strand = 0,
                          col_data = "colData in a tab-delimited txt file")
```

Alternatively, users can call the wrapper function `create_summarizedcounts()` to construct an object of SummarizedCounts.

Setting and getting fields

The library stranded information, `colData`, and corrected expression data of a SummarizedCounts object, `x`, can be simply accessed as follows: "'r x$lib_strand x$col_data x$global_correction x$gc_correction x$ir_correction x$stranded_correction

```
The field and sub-fields of featureCounts-summarized RNA-seq data can be
set and get as follows. The feature_counts list contains the following
elements: "gene", "exon", "intergenic_region", "intronic_region", "rRNA",
"mitochondrion", "gtf", "chloroplast". For genomic features, gene, exon,
intron, rRNA, mitochondrion, and chloroplast, only the `stat` elements of
the [Rsubread::featureCounts()] oupt is stored; for intergenic region,
the `stat`, `annotation` and `counts` elements are stored; for GTF-based
summarization, the `stat` and `counts` elements are stored.

```r
```
```

```
## the whole feature_counts list
x$set_feature_counts(feature_counts_list)
x$get_feature_counts()

## gene-coding regions: intron + exons
x$get_gene_stat ()

## exons
x$get_exon_stat()

## intergenic region summaries
x$get_ir_stat()
x$get_ir_counts()
x$get_ir_anno()

## intronic regions
x$get_intron_stat()

## rRNA-coding regions
x$get_rRNA_stat()

## mitochondrion
x$get_mt_stat()

## chloroplast
x$get_ct_stat()

## GTF meta-gene, exons as features
x$get_gtf_stat()
x$get_gtf_counts()
```

Adding Salmon quantification data (setting the library type information to the opposite type) to the SummarizedCounts object. For library type information, see https://salmon.readthedocs.io/en/latest/library_type.html.

```
x$set_salmon_quant(tximport_list)
x$get_salmon_quant()
x$get_salmon_counts()
x$get_salmon_abundance()
x$get_salmon_length()
```

Adding Salmon quantification data (setting the library type information to the opposite type) to the SummarizedCounts object. For library type information, see https://salmon.readthedocs.io/en/latest/library_type.html.

```
x$set_salmon_quant_opposite(tximport_list)
x$get_salmon_quant_opposite()
x$get_salmon_opposite_counts()
x$get_salmon_opposite_abundance()
```

Adding corrected expression data to the SummarizedCounts object:

```
x$set_global_correction(corrected_expr)
x$set_gc_correction(corrected_expr)
x$set_ir_correction(corrected_expr)
x$set_stranded_correction(salmon_correction)
```

Public fields

`lib_strand (integer(1))`
: Library strandedness, which can be 0, 1, or 2. See `create_summarizedcounts()`.

`col_data (data.frame)`
: colData for RNA-seq data, see `create_summarizedcounts()`.

`feature_counts (list())` FeatureCounts summaries for different types of genomic features.

`salmon_quant (list())`
: Salmon quant using the actual library strandedness,imported by tximport.

`salmon_quant_opposite (list())`
: Salmon quant using strandedness info opposite to actual library strandedness, imported by tximport.

`global_correction (matrix())`
: Count matrix corrected for gDNA contamination by using the "Global" method.

`gc_correction (matrix())`
: Count matrix corrected for gDNA contamination by using the "GC%" method.

`ir_correction (matrix())`
: Count matrix corrected for gDNA contamination by using the "IR%" method.

`stranded_correction (matrix())`
: Count matrix corrected for gDNA contamination by using the method dedicated to stranded RNA-seq data.

Methods

Public methods:

- `SummarizedCounts$new()`
- `SummarizedCounts$add_ir_rate()`
- `SummarizedCounts$set_feature_counts()`
- `SummarizedCounts$set_salmon_quant()`
- `SummarizedCounts$set_salmon_quant_opposite()`
- `SummarizedCounts$set_global_correction()`
- `SummarizedCounts$set_gc_correction()`
- `SummarizedCounts$set_ir_correction()`
- `SummarizedCounts$set_stranded_correction()`
- `SummarizedCounts$get_feature_counts()`
- `SummarizedCounts$get_gene_stat()`
- `SummarizedCounts$get_exon_stat()`
- `SummarizedCounts$get_ir_stat()`
- `SummarizedCounts$get_ir_counts()`

- SummarizedCounts$get_ir_anno()
- SummarizedCounts$get_intron_stat()
- SummarizedCounts$get_rRNA_stat()
- SummarizedCounts$get_mt_stat()
- SummarizedCounts$get_ct_stat()
- SummarizedCounts$get_gtf_stat()
- SummarizedCounts$get_gtf_counts()
- SummarizedCounts$get_salmon_quant()
- SummarizedCounts$get_salmon_counts()
- SummarizedCounts$get_salmon_abundance()
- SummarizedCounts$get_salmon_length()
- SummarizedCounts$get_salmon_quant_opposite()
- SummarizedCounts$get_salmon_opposite_counts()
- SummarizedCounts$get_salmon_opposite_abundance()
- SummarizedCounts$clone()

Method `new()`**:** Creates a new instance of this R6 class.
Note that this object is typically constructed via a wrapper function, `create_summarizedcounts()`.

Usage:

```
SummarizedCounts$new(lib_strand, col_data)
```

Arguments:

`lib_strand` (`integer(1)`) The library's strandedness. It has three possible values:

- 0: unstranded, the default;
- 1: stranded, read 1 (or single-end read) comes from the forward strand;
- 2: reversely stranded, read 1 (or single-end read) comes from the reverse strand For more details, See https://sailfish.readthedocs.io/en/master/library_type.html. `colData`

`col_data` (`data.frame()`) A data frame with rows corresponding to samples. For unstranded RNA-seq data, it at least contains the following columns: `sample_name`, `BAM_file`, `group`, `salmon_quant_file`, and `batch` if the data were generated in more than one batches. For stranded RNA-seq data, an extra column, `salmon_quant_file_opposite_strand`, should be included.

Method `add_ir_rate()`**:** Add IR% to the col_data by merging data frames

Usage:

```
SummarizedCounts$add_ir_rate(IR_rate)
```

Arguments:

`IR_rate` (`data.frame()`)
A data frame with sample_name as rownames, and a single column `IR_rate`, stroing the percentage of reads mapping to intergenic regions.

Returns: An object of SummarizedCounts.

Method `set_feature_counts()`**:** Set feature_counts

Usage:

```
SummarizedCounts$set_feature_counts(feature_counts_list)
```

Arguments:

`feature_counts_list (data.frame())`
: The feature_counts list contains the following elements: "gene", "exon", "intergenic_region", "intronic_region", "rRNA", "mitochondrion", "gtf", "chloroplast". For genomic features, gene, exon, intron, rRNA, mitochondrion, and chloroplast, only the `stat` elements of the `Rsubread::featureCounts()` oupt is stored; for intergenic region, the `stat`, `annotation` and `counts` elements are stored; for GTF-based summarization, the `stat` and `counts` elements are stored.

Returns: An object of SummarizedCounts.

Method `set_salmon_quant()`**:** set salmon_quant

Usage:

```
SummarizedCounts$set_salmon_quant(tximport_list)
```

Arguments:

`tximport_list (list())`
: A three-element `tximport()` list aggregated from Salmon quantification results genrated by using the correct library strandedness information, containing `counts`, `abundance`, and `length`.

Returns: An object of SummarizedCounts.

Method `set_salmon_quant_opposite()`**:** set salmon_quant_opposite

Usage:

```
SummarizedCounts$set_salmon_quant_opposite(tximport_list)
```

Arguments:

`tximport_list (list())`
: A three-element `tximport()` list aggregated from Salmon quantification results genrated by using the opposite library strandedness information, containing `counts`, `abundance`, and `length`.

Returns: An object of SummarizedCounts.

Method `set_global_correction()`**:** Set express matrix by using the "Global" method

Usage:

```
SummarizedCounts$set_global_correction(corrected_expr)
```

Arguments:

`corrected_expr (matrix())`
: A count matrix containing expression corrected by the "Global" method.

Returns: An object of SummarizedCounts.

Method `set_gc_correction()`**:** Set express matrix by using the "GC%" method

Usage:

```
SummarizedCounts$set_gc_correction(corrected_expr)
```

Arguments:

`corrected_expr (matrix())`
A count matrix containing expression corrected by the "GC%" method.

Returns: An object of SummarizedCounts.

Method `set_ir_correction()`**:** Set express matrix by using the "IR%" method

Usage:

```
SummarizedCounts$set_ir_correction(corrected_expr)
```

Arguments:

`corrected_expr (matrix())`
A matrix containing expression corrected by the "Global" method in the form of log2(count per million).

Returns: An object of SummarizedCounts.

Method `set_stranded_correction()`**:** Set express matrix by using the method dedicated to stranded RNA-seq data

Usage:

```
SummarizedCounts$set_stranded_correction(corrected_expr)
```

Arguments:

`corrected_expr (matrix())`
A count matrix containing expression corrected by the method dedicated to stranded RNA-seq data.

Returns: An object of SummarizedCounts.

Method `get_feature_counts()`**:** Get featureCounts output as a whole list

Usage:

```
SummarizedCounts$get_feature_counts()
```

Returns: A list containing the following elements: "gene", "exon", "intergenic_region", "intronic_region", "rRNA", "mitochondrion", "gtf", "chloroplast". For genomic features, gene, exon, intron, rRNA, mitochondrion, and chloroplast, only the `stat` elements of the `Rsubread::featureCounts()` output is stored; for intergenic region, the `stat`, `annotation` and `counts` elements are stored; for GTF-based summarization, the `stat` and `counts` elements are stored.

Method `get_gene_stat()`**:** Get featureCounts `stat` for the `gene-coding regions`

Usage:

```
SummarizedCounts$get_gene_stat()
```

Returns: only the `stat` elements of the `Rsubread::featureCounts()` output for the `gene-coding regions`.

Method `get_exon_stat()`**:** Get featureCounts `stat` for the `exon regions`

Usage:

```
SummarizedCounts$get_exon_stat()
```

Returns: only the `stat` elements of the `Rsubread::featureCounts()` output for the `exon regions`

Method `get_ir_stat()`**:** Get featureCounts `stat` for the `intergenic regions`

Usage:

```
SummarizedCounts$get_ir_stat()
```

Returns: only the stat elements of the Rsubread::featureCounts() output for the intergenic regions.

Method `get_ir_counts()`**:** Get featureCounts counts for the intergenic regions

Usage:

```
SummarizedCounts$get_ir_counts()
```

Returns: only the counts elements of the Rsubread::featureCounts() output for the intergenic regions.

Method `get_ir_anno()`**:** Get featureCounts annotation for the intergenic regions

Usage:

```
SummarizedCounts$get_ir_anno()
```

Returns: only the annotation elements of the Rsubread::featureCounts() output for the intergenic regions.

Method `get_intron_stat()`**:** Get featureCounts stat for the intronic regions

Usage:

```
SummarizedCounts$get_intron_stat()
```

Returns: only the stat elements of the Rsubread::featureCounts() output for the intronic regions.

Method `get_rRNA_stat()`**:** Get featureCounts stat for the rRNA-encoding regions

Usage:

```
SummarizedCounts$get_rRNA_stat()
```

Returns: only the stat elements of the Rsubread::featureCounts() output for the rRNA-encoding regions.

Method `get_mt_stat()`**:** Get featureCounts stat for the mitochondiral genome

Usage:

```
SummarizedCounts$get_mt_stat()
```

Returns: only the stat elements of the Rsubread::featureCounts() output for the mitochondiral genome.

Method `get_ct_stat()`**:** Get featureCounts stat for the chloroplast genome

Usage:

```
SummarizedCounts$get_ct_stat()
```

Returns: only the stat elements of the Rsubread::featureCounts() output for the chloroplast genome.

Method `get_gtf_stat()`**:** Get featureCounts stat for the GTF metagene

Usage:

```
SummarizedCounts$get_gtf_stat()
```

Returns: only the stat elements of the Rsubread::featureCounts() output for the GTF metagenes.

Method `get_gtf_counts()`**:** Get featureCounts counts for the GTF metagene

Usage:

```
SummarizedCounts$get_gtf_counts()
```

Returns: only the counts elements of the Rsubread::featureCounts() output for the GTF metagenes.

Method `get_salmon_quant()`: Get `tximport()` output as a whole list, including the three matrices `counts`, `abundance`, and `length`

Usage:

```
SummarizedCounts$get_salmon_quant()
```

Returns: A list of three elements, `counts`, `abundance`, and `length` of the `tximport::tximport()` output.

Method `get_salmon_counts()`: Get `tximport()` output as a whole list, only including the `counts` matrix

Usage:

```
SummarizedCounts$get_salmon_counts()
```

Returns: only the `counts` element of the `tximport::tximport()` output.

Method `get_salmon_abundance()`: Get `tximport()` output as a whole list, only including the `abundance` matrix

Usage:

```
SummarizedCounts$get_salmon_abundance()
```

Returns: only the `abundance` element of the `tximport::tximport()` output.

Method `get_salmon_length()`: Get `tximport()` output as a whole list, only including the `length` matrix

Usage:

```
SummarizedCounts$get_salmon_length()
```

Returns: only the `length` element of the `tximport::tximport()` output.

Method `get_salmon_quant_opposite()`: Get `tximport()` output as a whole list, including the three matrices `counts`, `abundance`, and `length`

Usage:

```
SummarizedCounts$get_salmon_quant_opposite()
```

Returns: A list of three elements, `counts`, `abundance`, and `length` of the `tximport::tximport()` output.

Method `get_salmon_opposite_counts()`: Get `tximport()` output as a whole list, only including the `count` matrix

Usage:

```
SummarizedCounts$get_salmon_opposite_counts()
```

Returns: only the `counts` element of the `tximport::tximport()` output.

Method `get_salmon_opposite_abundance()`: Get `tximport()` output as a whole list, only including the `abundance` matrix

Usage:

```
SummarizedCounts$get_salmon_opposite_abundance()
```

Returns: only the `abundance` element of the `tximport::tximport()` output.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SummarizedCounts$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index

∗ **datasets**
    feature_counts_list, 12
    gene_GC, 12
    intergenic_GC, 15
    salmon_quant, 23

BSgenome::BSgenome, *3, 4*, *26*

calc_gene_gc, 2
calc_gene_gc(), *6*, *12*
calc_region_gc, 4
calc_region_gc(), *6*, *12*, *15*, *23*
correct_GC, 6
correct_global, 7
correct_IR, 8
correct_stranded, 10
create_summarizedcounts, 11
create_summarizedcounts(), *29*, *31, 32*

ensembldb::EnsDb, *3*, *14*, *24*

feature_counts_list, 12

gene_GC, 12
GenomicRanges::GRanges, *4*
GenomicRanges::GRangesList, *4*
get_region_stat, 13
get_region_stat(), *21*
get_saf, 14
get_saf(), *4*, *27*
grDevices::pdf(), *19*
gtable::gtable(), *20*

intergenic_GC, 15

limma::lmFit(), *9*

plot_assignment_stat, 16
plot_expr_distr, 17
plot_gene_content, 18
plot_pca_heatmap, 20
plot_read_distr, 21
plot_read_distr(), *8*
plot_sample_corr, 22

qsmooth::qsmooth(), *17*

R6, *32*
Rsubread::featureCounts(), *12*, *16*, *27, 28*, *33–35*

salmon_quant, 23
salmon_tximport, 24
style_BSgenome, 25
summarize_reads, 27
SummarizedCounts, *6, 7*, *9–11*, *13*, *16–18*, *20*, *22*, *24*, *27, 28*, 29, *29*, *32–34*

tximport(), *33*, *36*
tximport::tximport(), *10*, *24*, *36*