

Package ‘CRMetrics’

June 2, 2026

Title Cell Ranger Output Filtering and Metrics Visualization

Version 0.3.3

Description Sample and cell filtering as well as visualisation of output metrics from 'Cell Ranger' by Grace X.Y. Zheng et al. (2017) <[doi:10.1038/ncomms14049](https://doi.org/10.1038/ncomms14049)>. 'CRMetrics' allows for easy plotting of output metrics across multiple samples as well as comparative plots including statistical assessments of these. 'CRMetrics' allows for easy removal of ambient RNA using 'SoupX' by Matthew D Young and Sam Behjati (2020) <[doi:10.1093/gigascience/giaa151](https://doi.org/10.1093/gigascience/giaa151)> or 'CellBender' by Stephen J Fleming et al. (2022) <[doi:10.1101/791699](https://doi.org/10.1101/791699)>. Furthermore, it is possible to preprocess data using 'Pagoda2' by Nikolas Barkas et al. (2021) <<https://github.com/kharchenko1ab/pagoda2>> or 'Seurat' by Yuhan Hao et al. (2021) <[doi:10.1016/j.cell.2021.04.048](https://doi.org/10.1016/j.cell.2021.04.048)> followed by embedding of cells using 'Conos' by Nikolas Barkas et al. (2019) <[doi:10.1038/s41592-019-0466-z](https://doi.org/10.1038/s41592-019-0466-z)>. Finally, doublets can be detected using 'scrublet' by Samuel L. Wolock et al. (2019) <[doi:10.1016/j.cels.2018.11.005](https://doi.org/10.1016/j.cels.2018.11.005)> or 'Doublet Detection' by Gayoso et al. (2020) <[doi:10.5281/zenodo.2678041](https://doi.org/10.5281/zenodo.2678041)>. In the end, cells are filtered based on user input for use in downstream applications.

License GPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Imports cowplot, dplyr, ggbeeswarm, ggplot2, ggpmisc, ggpubr, ggrepel, magrittr, Matrix, methods, R6, scales, sccore, sparseMatrixStats, stats, tibble, tidy, utils

Suggests conos, data.table, knitr, markdown, pagoda2, reticulate, rhdf5, rmarkdown, Seurat, SoupX, testthat (>= 3.0.0)

URL <https://github.com/khodosevichlab/CRMetrics>

BugReports <https://github.com/khodosevichlab/CRMetrics/issues>

Maintainer Rasmus Rydbirk <rnydbirk@bmb.sdu.dk>

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Rasmus Rydbirk [aut, cre],
 Fabienne Kick [aut],
 Henrietta Holze [aut],
 Xian Xin [ctb],
 Laura Wolbeck [aut]

Repository CRAN

Date/Publication 2026-06-02 06:50:11 UTC

Contents

CRMetrics	2
read10xH5	42

Index	43
--------------	-----------

CRMetrics	<i>CRMetrics class object</i>
-----------	-------------------------------

Description

Functions to analyze 10x Genomics Cell Ranger and Parse Biosciences split-pipe count data. To initialize a new object, 'data.path' or 'cms' is needed. 'metadata' is also recommended, but not required.

Public fields

`metadata` data.frame or character Path to metadata file or name of metadata data.frame object. Metadata must contain a column named 'sample' containing sample names that must match folder names in 'data.path' (default = NULL)

`data.path` character Path(s) to Cell Ranger or Parse's split-pipe count data, one directory per sample. If multiple paths, do `c("path1", "path2")` (default = NULL)

`cms` list List with count matrices from Cell Ranger or Parse's split-pipe (default = NULL)

`cms.preprocessed` list List with pagoda2- or seurat-preprocessed count matrices after `$doPreprocessing()` (default = NULL)

`cms.raw` list List with raw, unfiltered count matrices, i.e., including all cell barcodes detected, also empty ones (default = NULL)

`summary.metrics` data.frame Summary metrics from Cell Ranger or Parse's split-pipe pipeline (default = NULL)

`detailed.metrics` data.frame Detailed metrics, i.e., total number of genes and UMIs per cell (default = NULL)

`comp.group` character A group present in the metadata to compare the metrics by, can be added with `addComparison` (default = NULL)

`verbose` logical Print messages or not (default = TRUE)

`theme` ggplot2 theme (default: `theme_bw()`)

`pal` Plotting palette (default = NULL)

`n.cores` numeric Number of cores for calculations (default = 1) Initialize a CRMetrics object

Methods

Public methods:

- `CRMetrics$new()`
- `CRMetrics$addDetailedMetrics()`
- `CRMetrics$plotBarcodeRankPlot()`
- `CRMetrics$getExpectedCells()`
- `CRMetrics$addComparison()`
- `CRMetrics$plotSamples()`
- `CRMetrics$plotSummaryMetrics()`
- `CRMetrics$plotDetailedMetrics()`
- `CRMetrics$plotEmbedding()`
- `CRMetrics$plotDepth()`
- `CRMetrics$plotRawDepth()`
- `CRMetrics$plotMitoFraction()`
- `CRMetrics$detectDoublets()`
- `CRMetrics$doPreprocessing()`
- `CRMetrics$createEmbedding()`
- `CRMetrics$filterCms()`
- `CRMetrics$selectMetrics()`
- `CRMetrics$plotFilteredCells()`
- `CRMetrics$getDepth()`
- `CRMetrics$getMitoFraction()`
- `CRMetrics$prepareCellbender()`
- `CRMetrics$saveCellbenderScript()`
- `CRMetrics$addCms()`
- `CRMetrics$plotCbTraining()`
- `CRMetrics$plotCbCellProbs()`
- `CRMetrics$plotCbAmbExp()`
- `CRMetrics$plotCbAmbGenes()`
- `CRMetrics$addSummaryFromCms()`
- `CRMetrics$runSoupX()`
- `CRMetrics$plotSoupX()`
- `CRMetrics$plotCbCells()`
- `CRMetrics$addDoublets()`
- `CRMetrics$clone()`

`CRMetrics$new()`: To initialize new object, 'data.path' or 'cms' and 'technology' is needed. 'metadata' is also recommended, but not required.

Usage:

```
CRMetrics$new(  
  data.path = NULL,  
  technology = c("10x", "10xflex", "10xmultiome", "parse"),
```

```

metadata = NULL,
cms = NULL,
samples = NULL,
unique.names = TRUE,
sep.cells = "!!",
comp.group = NULL,
verbose = TRUE,
theme = theme_bw(),
n.cores = 1,
sep.meta = ",",
raw.meta = FALSE,
pal = NULL
)

```

Arguments:

`data.path` character Path to directory with count data, one directory per sample (default = NULL).

`technology` character applied single cell technology (default = c("10x", "10xflex", "10xmultiome", "parse"))

`metadata` data.frame or character Path to metadata file (comma-separated) or name of metadata dataframe object. Metadata must contain a column named 'sample' containing sample names that must match folder names in 'data.path' (default = NULL)

`cms` list List with count matrices (default = NULL)

`samples` character Sample names. Only relevant if cms is provided (default = NULL)

`unique.names` logical Create unique cell names. Only relevant if cms is provided (default = TRUE)

`sep.cells` character Sample-cell separator. Only relevant if cms is provided and unique.names=TRUE (default = "!!")

`comp.group` character A grouping variable present in the metadata column names, can be added with addComparison (default = NULL)

`verbose` logical Print messages or not (default = TRUE)

`theme` ggplot2 theme (default: theme_bw())

`n.cores` integer Number of cores for the calculations (default = self\$n.cores)

`sep.meta` character Separator for metadata file (default = ",")

`raw.meta` logical Keep metadata in its raw format. If FALSE, classes will be converted using "type.convert" (default = FALSE)

`pal` character Plotting palette (default = NULL)

Returns: CRMetrics object

Examples:

```
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
```

`CRMetrics$addDetailedMetrics()`: Function to read in detailed metrics (total UMI count and gene count per cell). This is not done upon object initialization for speed.

Usage:

```

CRMetrics$addDetailedMetrics(
  cms = self$cms,
  min.transcripts.per.cell = 100,
  n.cores = self$n.cores,
  verbose = self$verbose
)

```

Arguments:

cms list List of (sparse) count matrices (default = self\$cms)
min.transcripts.per.cell numeric Minimal number of transcripts per cell (default = 100)
n.cores integer Number of cores for the calculations (default = self\$n.cores).
verbose logical Print messages or not (default = self\$verbose).

Returns: Count matrices

Examples:

```

# Simulate data
testdata.cms <- lapply(seq_len(2), \ (x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \ (x) paste0("gene",x)),
    sapply(seq_len(1e3), \ (x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Run function
crm$addDetailedMetrics()

```

CRMetrics\$plotBarcodeRankPlot(): Read in raw count matrices and save UMI counts to object in "cellbender" slot, then create barcode rank plots for quality control

Usage:

```

CRMetrics$plotBarcodeRankPlot(
  shrinkage = 100,
  show.expected.cells = TRUE,
  cms.raw = self$cms.raw,
  technology = self$technology,
  umi.counts = self$umi.counts,
  data.path = self$data.path,
  samples = self$metadata$sample,
  verbose = self$verbose,
  n.cores = self$n.cores,
  unique.names = FALSE,
  sep = "!!"
)

```

Arguments:

shrinkage integer Select every nth UMI count per cell for plotting. Improves plotting speed drastically. To plot all cells, set to 1 (default = 100)

show.expected.cells logical Plot line depicting called cells by Parse or Cell Ranger pipeline (default = TRUE)

cms.raw list Raw count matrices from HDF5 Cell Ranger or Parse outputs (default = self\$cms.raw)

technology character applied single cell technology (default = c("10x", "10xflex", "10xmulti-ome", "parse"))

umi.counts list UMI counts calculated as column sums of raw count matrices from HDF5 Cell Ranger or Parse outputs (default: stored list)

data.path character Path to Cell Ranger or Parse outputs (default = self\$data.path)

samples character Sample names to include (default = self\$metadata\$sample)

verbose logical Show progress (default: stored vector)

n.cores integer Number of cores (default: stored vector)

unique.names logical Create unique cell names (default = FALSE)

sep character Separator for creating unique cell names (default = "!!!")

Returns: ggplot2 object and bash script

Examples:

```
crm <- CRMetrics$new(data.path = "/path/to/count/data")
crm$prepareCellbender()
```

`CRMetrics$getExpectedCells()`: Extract the expected number of cells per sample based on the Cell Ranger or Parse's split-pipe summary metrics.

Usage:

```
CRMetrics$getExpectedCells(samples = self$metadata$sample)
```

Arguments:

samples character Sample names to include (default = self\$metadata\$sample)

Returns: A numeric vector

Examples:

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Get summary
crm$addSummaryFromCms()

# Get no. cells
crm$getExpectedCells()
```

CRMetrics\$addComparison(): Add comparison group for statistical testing.

Usage:

```
CRMetrics$addComparison(comp.group, metadata = self$metadata)
```

Arguments:

comp.group character Comparison metric (default = self\$comp.group).

metadata data.frame Metadata for samples (default = self\$metadata).

Returns: Vector

Examples:

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add metadata
crm$metadata <- data.frame(sex = c("male", "female"))

# Add comparison group
crm$addComparison(comp.group = "sex")
```

CRMetrics\$plotSamples(): Plot the number of samples per group based on metadata.

Usage:

```
CRMetrics$plotSamples(
  comp.group = self$comp.group,
  h.adj = 0.05,
  exact = FALSE,
  metadata = self$metadata,
  second.comp.group = NULL,
  pal = self$pal
)
```

Arguments:

comp.group character Metadata column used as the primary grouping variable. Must match a column name of metadata (default = self\$comp.group).

h.adj numeric Relative vertical offset for displaying the statistical test p-value in the plot, expressed as a % of the maximum y-value (default = 0.05).

exact logical Whether to display exact p-values instead of rounded values (default = FALSE).

metadata data.frame Metadata for samples (default = self\$metadata).

second.comp.group character Optional second metadata column used for grouping and coloring bars in the plot. Must match a column name of metadata (default = NULL).

pal character Plotting palette (default = self\$pal)

Returns: ggplot2 object

Examples:

```

samples <- c("sample1", "sample2")

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})
names(testdata.cms) <- samples

# Create metadata
metadata <- data.frame(sample = samples,
  sex = c("male", "female"),
  condition = c("a", "b"))

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, metadata = metadata, n.cores = 1)

# Plot
crm$plotSamples(comp.group = "sex", second.comp.group = "condition")

```

CRMetrics\$plotSummaryMetrics(): Plot all summary stats or a selected list.

Usage:

```

CRMetrics$plotSummaryMetrics(
  comp.group = self$comp.group,
  second.comp.group = NULL,
  metrics = NULL,
  h.adj = 0.05,
  plot.stat = TRUE,
  stat.test = c("non-parametric", "parametric"),
  exact = FALSE,
  metadata = self$metadata,
  summary.metrics = self$summary.metrics,
  plot.geom = "bar",
  se = FALSE,
  group.reg.lines = FALSE,
  secondary.testing = TRUE,
  pal = self$pal
)

```

Arguments:

comp.group character Defines the groups to compare (categorical) or the variable for regression analysis (continuous). Must match a column name of metadata (default = self\$comp.group).

`second.comp.group` character Optional metadata column used for grouping, colouring or stratifying the plot. Usually categorical. Used for the metric "samples per group" and for grouped regression when "comp.group" is continuous Does not affect statistical testing (default = NULL).

`metrics` character Vector of metrics to plot. If NULL, all available metrics are plotted (default = NULL).

`h.adj` numeric Relative vertical offset for displaying the statistical test p-value in the plot, expressed as % of the maximum y-value (default = 0.05)

`plot.stat` logical Whether to show statistical test results in the plot. Automatically set to FALSE if `comp.group` is "sample" or continuous (default = TRUE)

`stat.test` character Statistical test to compare group means. Can either be "non-parametric" or "parametric" (default = "non-parametric").

`exact` logical Whether to display exact p-values instead of rounded values (default = FALSE).

`metadata` data.frame Metadata for samples (default = `self$metadata`).

`summary.metrics` data.frame Data frame of precomputed summary metrics for the samples (default = `self$summary.metrics`).

`plot.geom` character Geometric used to plot the data. Options include "bar", "point", "violin", "histogram" (default = "bar").

`se` logical For regression lines, whether to display standard errors (default = FALSE)

`group.reg.lines` logical For regression lines, if FALSE show a single line, if TRUE show one line per group defined by `second.comp.group` (default = FALSE)

`secondary.testing` logical Whether to show post hoc testing which is performed in case of multiple-group comparisons (default = TRUE)

`pal` character Plotting palette (default = `self$pal`)

Returns: ggplot2 object

Examples:

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add summary metrics
crm$addSummaryFromCms()

crm$plotSummaryMetrics(plot.geom = "point")
```

`CRMetrics$plotDetailedMetrics()`: Plot detailed metrics from the `detailed.metrics` object

Usage:

```
CRMetrics$plotDetailedMetrics(
  comp.group = self$comp.group,
  detailed.metrics = self$detailed.metrics,
  metadata = self$metadata,
  metrics = NULL,
  plot.geom = "violin",
  hline = TRUE,
  pal = self$pal
)
```

Arguments:

`comp.group` character Comparison metric (default = `self$comp.group`).

`detailed.metrics` data.frame Object containing the count matrices (default = `self$detailed.metrics`).

`metadata` data.frame Metadata for samples (default = `self$metadata`).

`metrics` character Metrics to plot. NULL plots both plots (default = NULL).

`plot.geom` character How to plot the data (default = "violin").

`hline` logical Whether to show median as horizontal line (default = TRUE)

`pal` character Plotting palette (default = `self$pal`)

`data.path` character Path to Cell Ranger count data (default = `self$data.path`).

Returns: ggplot2 object

Examples:

```
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add detailed metrics
crm$addDetailedMetrics()

# Plot
crm$plotDetailedMetrics()
```

`CRMetrics$plotEmbedding()`: Plot cells in an embedding using Conos, with coloring based on depth (= total UMI count per cell), doublet status, or mitochondrial gene fraction. Users can either specify cutoff thresholds to highlight filtered cells or plot the raw continuous values on the UMAP.

Usage:

```
CRMetrics$plotEmbedding(
  doublet.method = NULL,
  doublet.scores = FALSE,
```

```

doublet.score.threshold = NULL,
depth = FALSE,
depth.cutoff = NULL,
depth.cutoff.upper = NULL,
mito.frac = FALSE,
mito.cutoff = NULL,
species = c("human", "mouse"),
size = 0.3,
sep = "!!",
pal = NULL,
...
)

```

Arguments:

`doublet.method` character Colour cells based on doublet calling result of respective doublet detection method (default = NULL).

`doublet.scores` logical Colour cells by doublet score. Applied doublet method must be supplied in parameter 'doublet.method' (default = FALSE).

`doublet.score.threshold` numeric Set manual threshold for doublets. If NULL and `doublet.scores = TRUE`, continuous scores are plotted. Applied doublet method must be supplied in parameter 'doublet.method' (default = NULL).

`depth` logical Colour cells by depth. If TRUE and no cutoffs are supplied, continuous depth values are plotted (default = FALSE).

`depth.cutoff` numeric Lower depth cutoff. Can be a single value or a named numeric vector specifying cutoffs per sample (default = NULL).

`depth.cutoff.upper` numeric Upper depth cutoff. Can be a single value or a named numeric vector specifying cutoffs per sample (default = NULL).

`mito.frac` logical Plot mitochondrial fraction. If TRUE and `mito.cutoff` is NULL, continuous values are plotted (default = FALSE).

`mito.cutoff` numeric Mitochondrial fraction cutoff (default = NULL).

`species` character Species to calculate the mitochondrial fraction for (default = c("human","mouse")).

`size` numeric Dot size (default = 0.3)

`sep` character Separator for creating unique cell names (default = "!!")

`pal` character Plotting palette (default = NULL)

... Plotting parameters passed to `score::embeddingPlot`.

Returns: ggplot2 object

Examples:

```

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
sapply(seq_len(1e3), \x) paste0("cell",x))
return(out)
}
}
}

```

```

})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

crm$plotEmbedding()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

```

`CRMetrics$plotDepth()`: Plot per-sample density histograms of cell depths (=total UMI counts per cell). The plot highlights cells below or above a specified UMI cutoff with distinct colors, helping to assess data quality and filtering thresholds.

Usage:

```

CRMetrics$plotDepth(
  cutoff = 1000,
  samples = self$metadata$sample,
  sep = "!!",
  keep.col = "#E7CDC2",
  filter.col = "#A65141"
)

```

Arguments:

`cutoff` numeric Depth cutoff (lower). Can be a single value or a named numeric vector specifying cutoffs per sample (default = 1e3).
`samples` character Sample names to include for plotting (default = `$metadata$sample`).
`sep` character Separator for creating unique cell names (default = "!!")
`keep.col` character Color for density of cells that are kept (default = "#E7CDC2")
`filter.col` Character Color for density of cells to be filtered (default = "#A65141")

Returns: ggplot2 object

Examples:

```

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
sapply(seq_len(1e3), \x) paste0("cell",x)))
return(out)
}
}
}

```

```

})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot
crm$plotDepth()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

```

`CRMetrics$plotRawDepth()`: Plot per-sample histograms of total UMI counts per cell (= cell depth), allowing visualization of sequencing depth distributions across samples.

Usage:

```

CRMetrics$plotRawDepth(
  cutoff = 1000,
  samples = self$metadata$sample,
  sep = "!!",
  binwidth = 10,
  xupper = 20000
)

```

Arguments:

`cutoff` numeric Depth cutoff (lower) used to distinguish filtered and retained cells. Can be a single value or a named numeric vector specifying cutoffs per sample (Default = 1e3).
`samples` character Vector of sample names to include in the plot (default = `self$metadata$sample`).
`sep` character Separator for creating unique cell names (default = "!!")
`binwidth` numeric Width of the histogram bins (default = 10).
`xupper` numeric Optional upper x-axis limit (default = 2e4).

Returns: A combined ggplot object showing UMI count histograms per sample.

`CRMetrics$plotMitoFraction()`: Plot the mitochondrial fraction in histogram.

Usage:

```

CRMetrics$plotMitoFraction(
  cutoff = 0.05,
  species = c("human", "mouse"),
  samples = self$metadata$sample,
  sep = "!!",
  keep.col = "#E7CDC2",
  filter.col = "#A65141"
)

```

Arguments:

cutoff numeric The mito. fraction cutoff to color the embedding (default = 0.05)
 species character Species to calculate the mitochondrial fraction for (default = "human")
 samples character Sample names to include for plotting (default = \$metadata\$sample)
 sep character Separator for creating unique cell names (default = "!!")
 keep.col character Color for density of cells that are kept (default = "#E7CDC2")
 filter.col Character Color for density of cells to be filtered (default = "#A65141")

Returns: ggplot2 object

Examples:

```
if (requireNamespace("pagoda2", quietly = TRUE)) {
  if (requireNamespace("conos", quietly = TRUE)) {
    # Simulate data
    testdata.cms <- lapply(seq_len(2), \(x) {
      out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
      out[out < 0] <- 1
      dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
        sapply(seq_len(1e3), \(x) paste0("cell",x)))
      return(out)
    })

    # Initialize
    crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

    # Create embedding
    crm$doPreprocessing()
    crm$createEmbedding()

    # Plot
    crm$plotMitoFraction()
  } else {
    message("Package 'conos' not available.")
  }
} else {
  message("Package 'pagoda2' not available.")
}
```

CRMetrics\$detectDoublets(): Run doublet detection methods.

Usage:

```
CRMetrics$detectDoublets(
  method = c("scrublet", "doubletdetection", "scdblfinder"),
  cms = self$cms,
  samples = self$metadata$sample,
  env = "r-reticulate",
  conda.path = system("whereis conda"),
  n.cores = self$n.cores,
  verbose = self$verbose,
```

```

    args = list(),
    export = FALSE,
    data.path = self$data.path
  )

```

Arguments:

`method` character Which method to use, `scrublet`, `doubletdetection` or `scdblfinder` (default="scrublet").

`cms` list List containing the count matrices (default=self\$cms).

`samples` character Vector of sample names. If NULL, samples are extracted from cms (default = self\$metadata\$sample)

`env` character Environment to run python in (default="r-reticulate").

`conda.path` character Path to conda environment (default=system("whereis conda")).

`n.cores` integer Number of cores to use (default = self\$n.cores)

`verbose` logical Print messages or not (default = self\$verbose)

`args` list A list with additional arguments for `DoubletDetection`, `scrublet` or `scDb1Finder`. Please check the respective manuals.

`export` boolean Export count matrices as transposed .mtx files and save python script to run doublet detection outside R (default = FALSE)

`data.path` character Path to write data, only relevant if `export = TRUE`. Last character must be / (default = self\$data.path)

Returns: data.frame

Examples:

```

# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
    sapply(seq_len(1e3), \x) paste0("cell",x))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Detect doublets
crm$detectDoublets(method = "scrublet",
  conda.path = "/opt/software/miniconda/4.12.0/condabin/conda")

```

`CRMetrics$doPreprocessing()`: Perform conos preprocessing.

Usage:

```

CRMetrics$doPreprocessing(
  cms = self$cms,
  preprocess = c("pagoda2", "seurat"),
  min.transcripts.per.cell = 100,

```

```

    verbose = self$verbose,
    n.cores = self$n.cores,
    get.largevis = FALSE,
    tsne = FALSE,
    make.geneknn = FALSE,
    cluster = FALSE,
    ...
)

```

Arguments:

`cms` list List containing the count matrices (default = `self$cms`).

`preprocess` character Method to use for preprocessing (default = `c("pagoda2", "seurat")`).

`min.transcripts.per.cell` numeric Minimal transcripts per cell (default = 100)

`verbose` logical Print messages or not (default = `self$verbose`).

`n.cores` integer Number of cores for the calculations (default = `self$n.cores`).

`get.largevis` logical For Pagoda2, create largeVis embedding (default = FALSE)

`tsne` logical Create tSNE embedding (default = FALSE)

`make.geneknn` logical For Pagoda2, estimate gene kNN (default = FALSE)

`cluster` logical For Seurat, estimate clusters (default = FALSE)

... Additional arguments for `Pagoda2::basicP2Proc` or `conos::basicSeuratProc`

Returns: Conos object

Examples:

```

if (requireNamespace("pagoda2", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
sapply(seq_len(1e3), \x) paste0("cell",x)))
return(out)
}}

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Perform preprocessing
crm$doPreprocessing(preprocess = "pagoda2")
} else {
message("Package 'pagoda2' not available.")
}

```

`CRMetrics$createEmbedding()`: Create Conos embedding.

Usage:

```

CRMetrics$createEmbedding(
  cms = self$cms.preprocessed,
  verbose = self$verbose,

```

```

    n.cores = self$n.cores,
    arg.buildGraph = list(),
    arg.findCommunities = list(),
    arg.embedGraph = list(method = "UMAP")
  )

```

Arguments:

`cms` list List containing the preprocessed count matrices (default = `self$cms.preprocessed`).

`verbose` logical Print messages or not (default = `self$verbose`).

`n.cores` integer Number of cores for the calculations (default = `self$n.cores`).

`arg.buildGraph` list A list with additional arguments for the `buildGraph` function in `Conos` (default = `list()`)

`arg.findCommunities` list A list with additional arguments for the `findCommunities` function in `Conos` (default = `list()`)

`arg.embedGraph` list A list with additional arguments for the `embedGraph` function in `Conos` (default = `list(method = "UMAP")`)

Returns: `Conos` object

Examples:

```

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

```

`CRMetrics$filterCms()`: Filter cells based on depth, mitochondrial fraction and doublets from the count matrix.

Usage:

```

CRMetrics$filterCms(
  depth.cutoff = NULL,

```

```

depth.cutoff.upper = NULL,
mito.cutoff = NULL,
doublet.method = NULL,
doublet.score.threshold = NULL,
species = c("human", "mouse"),
samples.to.exclude = NULL,
verbose = self$verbose,
sep = "!!",
raw = FALSE
)

```

Arguments:

`depth.cutoff` numeric Lower depth (total UMI counts per cell) cutoff (default = NULL).
`depth.cutoff.upper` numeric Upper depth (total UMI counts per cell) cutoff (default = NULL).
`mito.cutoff` numeric Mitochondrial fraction cutoff (default = NULL).
`doublet.method` character Doublet detection method to use (default = NULL).
`doublet.score.threshold` Manually set threshold for doublets across all samples. Cells with a doublet score above the threshold are declared doublets (default = NULL).
`species` character Species to calculate the mitochondrial fraction for (default = "human").
`samples.to.exclude` character Sample names to exclude (default = NULL)
`verbose` logical Show progress (default = self\$verbose)
`sep` character Separator for creating unique cell names (default = "!!")
`raw` boolean Filter on raw, unfiltered count matrices. Usually not intended (default = FALSE)

Returns: list of filtered count matrices

Examples:

```

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
sapply(seq_len(1e3), \x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetric$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Filter CMs
crm$filterCms(depth.cutoff = 1e3, mito.cutoff = 0.05)
} else {

```

```

message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

```

CRMetrics\$selectMetrics(): Select metrics from summary.metrics

Usage:

```
CRMetrics$selectMetrics(ids = NULL)
```

Arguments:

ids character Metric id to select (default = NULL).

Returns: vector

Examples:

```

# Simulate data
testdata.cms <- lapply(seq_len(2), \ (x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \ (x) paste0("gene",x)),
    sapply(seq_len(1e3), \ (x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Select metrics
crm$selectMetrics()
selection.metrics <- crm$selectMetrics(c(1:4))

```

CRMetrics\$plotFilteredCells(): Plot filtered cells in an embedding, in a bar plot, on a tile or export the data frame

Usage:

```

CRMetrics$plotFilteredCells(
  type = c("embedding", "bar", "tile", "export"),
  depth = FALSE,
  depth.cutoff = NULL,
  depth.cutoff.upper = NULL,
  doublet.method = NULL,
  doublet.score.threshold = NULL,
  mito.frac = FALSE,
  mito.cutoff = NULL,
  species = c("human", "mouse"),
  size = 0.3,
  sep = "!!",
  cols = c("grey80", "red", "blue", "green", "yellow", "black", "pink", "purple"),
  ...
)

```

Arguments:

type character The type of plot to use: embedding, bar, tile or export (default = c("embedding", "bar", "tile", "export")).
depth logical Plot the depth (= total amount of UMI counts per cell) or not (default = FALSE).
depth.cutoff numeric Lower depth cutoff, either a single number or a vector with cutoff per sample and with sample IDs as names (default = NULL).
depth.cutoff.upper numeric Upper depth cutoff, either a single number or a vector with cutoff per sample and with sample IDs as names (default = NULL).
doublet.method character Method to detect doublets (default = NULL).
doublet.score.threshold Manually set threshold for doublets across all samples. Cells with a doublet score above the threshold are declared doublets. (default = NULL)
mito.frac logical Plot the mitochondrial fraction or not (default = FALSE).
mito.cutoff numeric Mitochondrial fraction cutoff, either a single number or a vector with cutoff per sample and with sample IDs as names (default = NULL).
species character Species to calculate the mitochondrial fraction for (default = c("human", "mouse")).
size numeric Dot size (default = 0.3)
sep character Separator for creating unique cell names (default = "!!")
cols character Colors used for plotting (default = c("grey80", "red", "blue", "green", "yellow", "black", "pink", "purple"))
 ... Plotting parameters passed to `score::embeddingPlot`.

Returns: ggplot2 object or data frame

Examples:

```

if (requireNamespace("pagoda2", quietly = TRUE)) {
  if (requireNamespace("conos", quietly = TRUE)) {
    # Simulate data
    testdata.cms <- lapply(seq_len(2), \x) {
      out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
      out[out < 0] <- 1
      dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene", x)),
        sapply(seq_len(1e3), \x) paste0("cell", x))
      return(out)
    }

    # Initialize
    crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

    # Create embedding
    crm$doPreprocessing()
    crm$createEmbedding()

    # Plot and extract result
    crm$plotFilteredCells(type = "embedding")
    filtered.cells <- crm$plotFilteredCells(type = "export")
  } else {
    message("Package 'conos' not available.")
  }
} else {
  message("Package 'pagoda2' not available.")
}

```

CRMetrics\$getDepth(): Extract sum of UMI counts per cell (= cell depth)

Usage:

```
CRMetrics$getDepth(cms = self$cms)
```

Arguments:

cms list List of (sparse) count matrices (default = self\$cms)

Returns: data frame

Examples:

```
if (requireNamespace("pagoda2", quietly = TRUE)) {
  if (requireNamespace("conos", quietly = TRUE)) {
    # Simulate data
    testdata.cms <- lapply(seq_len(2), \(x) {
      out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
      out[out < 0] <- 1
      dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
        sapply(seq_len(1e3), \(x) paste0("cell",x)))
      return(out)
    })

    # Initialize
    crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

    # Create embedding
    crm$doPreprocessing()
    crm$createEmbedding()

    # Get depth
    crm$getDepth()
  } else {
    message("Package 'conos' not available.")
  }
} else {
  message("Package 'pagoda2' not available.")
}
```

CRMetrics\$getMitoFraction(): Calculate the fraction of mitochondrial genes.

Usage:

```
CRMetrics$getMitoFraction(species = c("human", "mouse"), cms = self$cms)
```

Arguments:

species character Species to calculate the mitochondrial fraction for (default = "human").

cms list List of (sparse) count matrices (default = self\$cms)

Returns: data frame

Examples:

```
if (requireNamespace("pagoda2", quietly = TRUE)) {
  if (requireNamespace("conos", quietly = TRUE)) {
```

```

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Get mito. fraction
crm$getMitoFraction(species = c("human", "mouse"))
} else {
  message("Package 'conos' not available.")
}
} else {
  message("Package 'pagoda2' not available.")
}

```

`CRMetrics$prepareCellbender()`: Generates input files for CellBender from Parse's split-pipe output.

Usage:

```

CRMetrics$prepareCellbender(
  technology = self$technology,
  data.path = self$data.path,
  samples = self$metadata$sample,
  verbose = self$verbose,
  n.cores = self$n.cores,
  unique.names = FALSE,
  sep = "!!"
)

```

Arguments:

`technology` character applied single cell technology (default = `c("10x", "10xflex", "10xmulti-ome", "parse")`)

`data.path` character Path to Cell Ranger outputs (default = `self$data.path`)

`samples` character Sample names to include (default = `self$metadata$sample`)

`verbose` logical Show progress (default: stored vector)

`n.cores` integer Number of cores (default: stored vector)

`unique.names` logical Create unique cell names (default = `FALSE`)

`sep` character Separator for creating unique cell names (default = `"!!"`)

Returns: Saves files to disc.

Examples:

```
crm <- CRMetrics$new(data.path = "/path/to/count/data")
crm$prepareCellbender()
```

CRMetrics\$saveCellbenderScript(): Create CellBender script

Usage:

```
CRMetrics$saveCellbenderScript(
  file = "cellbender_script.sh",
  technology = self$technology,
  fpr = 0.01,
  epochs = 150,
  use.gpu = TRUE,
  expected.cells = NULL,
  total.droplets = NULL,
  data.path = self$data.path,
  samples = self$metadata$sample,
  args = NULL
)
```

Arguments:

`file` character File name for CellBender script. Will be stored in `data.path` (default: "cellbender_script.sh")

`technology` character Applied single cell technology (default = `self$technology`).

`fpr` numeric False positive rate for CellBender (default = 0.01)

`epochs` integer Number of epochs for CellBender (default = 150)

`use.gpu` logical Use CUDA capable GPU (default = TRUE)

`expected.cells` named numeric vector By default, the `-expected-cells` argument is omitted in the generated script. Otherwise, a named vector of expected cell numbers with sample IDs as names. Sample IDs must match those in `summary.metrics` (default = NULL).

`total.droplets` named numeric vector By default, the `-total-droplets-included` argument is omitted in the generated script. Otherwise, a named vector of total droplets included with sample IDs as names. Sample IDs must match those in `summary.metrics` (default = NULL).

`data.path` character Path to Cell Ranger or Parse's split-pipe outputs (default = `self$data.path`)

`samples` character Sample names to include (default = `self$metadata$sample`)

`args` character (optional) Additional parameters for CellBender

Returns: bash script

Examples:

```
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
```

CRMetrics\$addCms(): Add a list of count matrices to the CRMetrics object.

Usage:

```
CRMetrics$addCms(
  cms = NULL,
  data.path = self$data.path,
```

```

    technology = self$technology,
    samples = self$metadata$sample,
    cellbender = FALSE,
    raw = FALSE,
    symbol = TRUE,
    unique.names = TRUE,
    sep = "!!",
    add.metadata = TRUE,
    n.cores = self$n.cores,
    verbose = self$verbose
  )

```

Arguments:

`cms` list List of (sparse) count matrices (default = NULL)

`data.path` character Path to Cell Ranger or Parse's split-pipe count data (default = `self$data.path`).

`technology` character Applied single cell technology (default = `self$technology`).

`samples` character Vector of sample names. If NULL, samples are extracted from `cms` (default = `self$metadata$sample`)

`cellbender` logical Add CellBender filtered count matrices in HDF5 format. Requires that "cellbender" is in the names of the files (default = FALSE)

`raw` logical Add raw count matrices from Cell Ranger or Parse's split-pipe output. Cannot be combined with `cellbender=TRUE`. (default = FALSE)

`symbol` character The type of gene IDs to use, SYMBOL (TRUE) or ENSEMBLE (default = TRUE)

`unique.names` logical Make cell names unique based on `sep` parameter (default = TRUE)

`sep` character Separator used to create unique cell names (default = "!!")

`add.metadata` boolean Add metadata from `cms` or not (default = TRUE)

`n.cores` integer Number of cores to use (default = `self$n.cores`)

`verbose` boolean Print progress (default = `self$verbose`)

Returns: Add list of (sparse) count matrices to R6 class object

Examples:

```

crm <- CRMetrics$new(data.path = "/path/to/count/data/")

# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
    sapply(seq_len(1e3), \x) paste0("cell",x))
  return(out)
})

crm$addCms(cms = testdata.cms)

```

`CRMetrics$plotCbTraining()`: Plot the results from the CellBender estimations

Usage:

```

CRMetrics$plotCbTraining(
  data.path = self$data.path,
  samples = self$metadata$sample,
  technology = self$technology,
  pal = self$pal
)

```

Arguments:

`data.path` character Path to Cell Ranger outputs (default = `self$data.path`)
`samples` character Sample names to include (default = `self$metadata$sample`)
`technology` character applied single cell technology (default = `c("10x", "10xflex", "10xmulti-ome", "parse")`)
`pal` character Plotting palette (default = `self$pal`)

Returns: A ggplot2 object

Examples:

```

crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbTraining()

```

`CRMetrics$plotCbCellProbs()`: Plot the CellBender assigned cell probabilities

Usage:

```

CRMetrics$plotCbCellProbs(
  data.path = self$data.path,
  samples = self$metadata$sample,
  technology = self$technology,
  low.col = "gray",
  high.col = "red"
)

```

Arguments:

`data.path` character Path to Cell Ranger outputs (default = `self$data.path`)
`samples` character Sample names to include (default = `self$metadata$sample`)
`technology` character applied single cell technology (default = `c("10x", "10xflex", "10xmulti-ome", "parse")`)
`low.col` character Color for low probabilities (default = "gray")
`high.col` character Color for high probabilities (default = "red")

Returns: A ggplot2 object

Examples:

```

crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run the CellBender script
crm$plotCbCellProbs()

```

`CRMetrics$plotCbAmbExp()`: Plot the estimated ambient gene expression per sample from CellBender calculations

Usage:

```
CRMetrics$plotCbAmbExp(
  cutoff = 0.005,
  data.path = self$data.path,
  samples = self$metadata$sample,
  technology = self$technology
)
```

Arguments:

`cutoff` numeric Horizontal line included in the plot to indicate highly expressed ambient genes (default = 0.005)

`data.path` character Path to Cell Ranger outputs (default = `self$data.path`)

`samples` character Sample names to include (default = `self$metadata$sample`)

`technology` character applied single cell technology (default = `c("10x", "10xflex", "10xmultiome", "parse")`)

Returns: A `ggplot2` object

Examples:

```
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbExp()
```

`CRMetrics$plotCbAmbGenes()`: Plot the most abundant estimated ambient genes from the CellBender calculations

Usage:

```
CRMetrics$plotCbAmbGenes(
  cutoff = 0.005,
  data.path = self$data.path,
  samples = self$metadata$sample,
  technology = self$technology,
  pal = self$pal
)
```

Arguments:

`cutoff` numeric Cutoff of ambient gene expression to use to extract ambient genes per sample

`data.path` character Path to Cell Ranger outputs (default = `self$data.path`)

`samples` character Sample names to include (default = `self$metadata$sample`)

`technology` character applied single cell technology (default = `c("10x", "10xflex", "10xmultiome", "parse")`)

`pal` character Plotting palette (default = `self$pal`)

Returns: A `ggplot2` object

Examples:

```

crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbGenes()

```

CRMetrics\$addSummaryFromCms(): Add summary metrics from a list of count matrices

Usage:

```

CRMetrics$addSummaryFromCms(
  cms = self$cms,
  n.cores = self$n.cores,
  verbose = self$verbose
)

```

Arguments:

cms list A list of filtered count matrices (default = self\$cms)
n.cores integer Number of cores to use (default = self\$n.cores)
verbose logical Show progress (default = self\$verbose)

Returns: data.frame

Examples:

```

# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
    sapply(seq_len(1e3), \x) paste0("cell",x))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add summary
crm$addSummaryFromCms()

```

CRMetrics\$runSoupX(): Run SoupX ambient RNA estimation and correction

Usage:

```

CRMetrics$runSoupX(
  data.path = self$data.path,
  samples = self$metadata$sample,
  n.cores = self$n.cores,
  verbose = self$verbose,
  arg.load10X = list(),
  arg.autoEstCont = list(),
  arg.adjustCounts = list()
)

```

Arguments:

`data.path` character Path to Cell Ranger outputs (default = `self$data.path`)
`samples` character Sample names to include (default = `self$metadata$sample`)
`n.cores` numeric Number of cores (default = `self$n.cores`)
`verbose` logical Show progress (default = `self$verbose`)
`arg.load10X` list A list with additional parameters for `SoupX::load10X` (default = `list()`)
`arg.autoEstCont` list A list with additional parameters for `SoupX::autoEstCont` (default = `list()`)
`arg.adjustCounts` list A list with additional parameters for `SoupX::adjustCounts` (default = `list()`)

Returns: List containing a list with corrected counts, and a data.frame containing plotting estimations

Examples:

```
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()
```

`CRMetrics$plotSoupX()`: Plot the results from the SoupX estimations

Usage:

```
CRMetrics$plotSoupX(plot.df = self$soupx$plot.df)
```

Arguments:

`plot.df` data.frame SoupX estimations (default = `self$soupx$plot.df`)

Returns: A ggplot2 object

Examples:

```
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()
crm$plotSoupX()
```

`CRMetrics$plotCbCells()`: Plot CellBender cell estimations against the estimated cell numbers from Cell Ranger

Usage:

```
CRMetrics$plotCbCells(
  data.path = self$data.path,
  samples = self$metadata$sample,
  technology = self$technology,
  pal = self$pal
)
```

Arguments:

`data.path` character Path to Cell Ranger outputs (default = `self$data.path`)
`samples` character Sample names to include (default = `self$metadata$sample`)
`technology` character applied single cell technology (default = `c("10x", "10xflex", "10xmulti-ome", "parse")`) description
`pal` character Plotting palette (default = `self$pal`)

Returns: A ggplot2 object

Examples:

```

crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbCells()

```

CRMetrics\$addDoublets(): Add doublet results created from exported Python script

Usage:

```

CRMetrics$addDoublets(
  method = c("scrublet", "doubletdetection"),
  data.path = self$data.path,
  samples = self$metadata$sample,
  cms = self$cms,
  verbose = self$verbose
)

```

Arguments:

method character Which method to use, either scrublet or doubletdetection (default is both).

data.path character Path to Cell Ranger outputs (default = self\$data.path)

samples character Sample names to include (default = self\$metadata\$sample)

cms list List containing the count matrices (default = self\$cms).

verbose boolean Print progress (default = self\$verbose)

Returns: List of doublet results

Examples:

```

crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$detectDoublets(export = TRUE)
## Run Python script
crm$addDoublets()

```

CRMetrics\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```

## -----
## Method `CRMetrics$new()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")

## End(Not run)

```

```

## -----
## Method `CRMetrics$addDetailedMetrics()`
## -----

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Run function
crm$addDetailedMetrics()

## -----
## Method `CRMetrics$plotBarcodeRankPlot()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data")
crm$prepareCellbender()

## End(Not run)

## -----
## Method `CRMetrics$getExpectedCells()`
## -----

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Get summary
crm$addSummaryFromCms()

# Get no. cells
crm$getExpectedCells()

## -----

```

```

## Method `CRMetrics$addComparison`
## -----

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add metadata
crm$metadata <- data.frame(sex = c("male", "female"))

# Add comparison group
crm$addComparison(comp.group = "sex")

## -----
## Method `CRMetrics$plotSamples`
## -----

samples <- c("sample1", "sample2")

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})
names(testdata.cms) <- samples

# Create metadata
metadata <- data.frame(sample = samples,
  sex = c("male", "female"),
  condition = c("a", "b"))

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, metadata = metadata, n.cores = 1)

# Plot
crm$plotSamples(comp.group = "sex", second.comp.group = "condition")

## -----
## Method `CRMetrics$plotSummaryMetrics`
## -----

```

```

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetric$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add summary metrics
crm$addSummaryFromCms()

crm$plotSummaryMetrics(plot.geom = "point")

## -----
## Method `CRMetric$plotDetailedMetrics()`
## -----

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
    sapply(seq_len(1e3), \(x) paste0("cell",x)))
  return(out)
})

# Initialize
crm <- CRMetric$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add detailed metrics
crm$addDetailedMetrics()

# Plot
crm$plotDetailedMetrics()

## -----
## Method `CRMetric$plotEmbedding()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
  if (requireNamespace("conos", quietly = TRUE)) {
    # Simulate data
    testdata.cms <- lapply(seq_len(2), \(x) {
      out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
      out[out < 0] <- 1

```

```

dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

crm$plotEmbedding()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

## -----
## Method `CRMetrics$plotDepth()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot
crm$plotDepth()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}

```

```

## -----
## Method `CRMetrics$plotMitoFraction()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot
crm$plotMitoFraction()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

## -----
## Method `CRMetrics$detectDoublets()`
## -----

## Not run:
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Detect doublets

```

```

crm$detectDoublets(method = "scrublet",
conda.path = "/opt/software/miniconda/4.12.0/condabin/conda")

## End(Not run)

## -----
## Method `CRMetrics$doPreprocessing()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Perform preprocessing
crm$doPreprocessing(preprocess = "pagoda2")
} else {
message("Package 'pagoda2' not available.")
}

## -----
## Method `CRMetrics$createEmbedding()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()
} else {

```

```

message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

## -----
## Method `CRMetrics$filterCms()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Filter CMs
crm$filterCms(depth.cutoff = 1e3, mito.cutoff = 0.05)
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

## -----
## Method `CRMetrics$selectMetrics()`
## -----

# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

```

```

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Select metrics
crm$selectMetrics()
selection.metrics <- crm$selectMetrics(c(1:4))

## -----
## Method `CRMetrics$plotFilteredCells()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
  if (requireNamespace("conos", quietly = TRUE)) {
    # Simulate data
    testdata.cms <- lapply(seq_len(2), \(x) {
      out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
      out[out < 0] <- 1
      dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
        sapply(seq_len(1e3), \(x) paste0("cell",x)))
      return(out)
    })

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Plot and extract result
crm$plotFilteredCells(type = "embedding")
filtered.cells <- crm$plotFilteredCells(type = "export")
} else {
  message("Package 'conos' not available.")
}
} else {
  message("Package 'pagoda2' not available.")
}

## -----
## Method `CRMetrics$getDepth()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
  if (requireNamespace("conos", quietly = TRUE)) {
    # Simulate data
    testdata.cms <- lapply(seq_len(2), \(x) {
      out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
      out[out < 0] <- 1

```

```

dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Get depth
crm$getDepth()
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}

## -----
## Method `CRMetrics$getMitoFraction()`
## -----

if (requireNamespace("pagoda2", quietly = TRUE)) {
if (requireNamespace("conos", quietly = TRUE)) {
# Simulate data
testdata.cms <- lapply(seq_len(2), \(x) {
out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
out[out < 0] <- 1
dimnames(out) <- list(sapply(seq_len(2e3), \(x) paste0("gene",x)),
sapply(seq_len(1e3), \(x) paste0("cell",x)))
return(out)
})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Create embedding
crm$doPreprocessing()
crm$createEmbedding()

# Get mito. fraction
crm$getMitoFraction(species = c("human", "mouse"))
} else {
message("Package 'conos' not available.")
}
} else {
message("Package 'pagoda2' not available.")
}
}

```

```

## -----
## Method `CRMetric$prepareCellbender()`
## -----

## Not run:
crm <- CRMetric$new(data.path = "/path/to/count/data")
crm$prepareCellbender()

## End(Not run)

## -----
## Method `CRMetric$saveCellbenderScript()`
## -----

## Not run:
crm <- CRMetric$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()

## End(Not run)

## -----
## Method `CRMetric$addCms()`
## -----

## Not run:
crm <- CRMetric$new(data.path = "/path/to/count/data/")

# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
    sapply(seq_len(1e3), \x) paste0("cell",x))
  return(out)
})

crm$addCms(cms = testdata.cms)

## End(Not run)

## -----
## Method `CRMetric$plotCbTraining()`
## -----

## Not run:
crm <- CRMetric$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbTraining()

```

```

## End(Not run)

## -----
## Method `CRMetrics$plotCbCellProbs()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run the CellBender script
crm$plotCbCellProbs()

## End(Not run)

## -----
## Method `CRMetrics$plotCbAmbExp()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbExp()

## End(Not run)

## -----
## Method `CRMetrics$plotCbAmbGenes()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbAmbGenes()

## End(Not run)

## -----
## Method `CRMetrics$addSummaryFromCms()`
## -----

# Simulate data
testdata.cms <- lapply(seq_len(2), \x) {
  out <- Matrix::rsparsematrix(2e3, 1e3, 0.1)
  out[out < 0] <- 1
  dimnames(out) <- list(sapply(seq_len(2e3), \x) paste0("gene",x)),
    sapply(seq_len(1e3), \x) paste0("cell",x))
  return(out)
}

```

```

})

# Initialize
crm <- CRMetrics$new(cms = testdata.cms, samples = c("sample1", "sample2"), n.cores = 1)

# Add summary
crm$addSummaryFromCms()

## -----
## Method `CRMetrics$runSoupX()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()

## End(Not run)

## -----
## Method `CRMetrics$plotSoupX()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$runSoupX()
crm$plotSoupX()

## End(Not run)

## -----
## Method `CRMetrics$plotCbCells()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$prepareCellbender()
crm$saveCellbenderScript()
## Run CellBender script
crm$plotCbCells()

## End(Not run)

## -----
## Method `CRMetrics$addDoublets()`
## -----

## Not run:
crm <- CRMetrics$new(data.path = "/path/to/count/data/")
crm$detectDoublets(export = TRUE)
## Run Python script
crm$addDoublets()

## End(Not run)

```

read10xH5 *Read 10x or cellbenders HDF5 files*

Description

Read 10x or cellbenders HDF5 files

Usage

```
read10xH5(
  data.path,
  samples = NULL,
  type = c("raw", "filtered", "cellbender", "cellbender_filtered"),
  technology = c("10x", "10xmultiome", "flex"),
  symbol = TRUE,
  sep = "!!",
  n.cores = 1,
  verbose = TRUE,
  unique.names = FALSE
)
```

Arguments

data.path	character
samples	character vector, select specific samples for processing (default = NULL)
type	name of H5 file to search for, "raw" and "filtered" are Cell Ranger count outputs, "cellbender" is output from CellBender after running script from saveCellbenderScript
technology	character Used single-cell technology (c("10x", "10xmultiome", "flex"))
symbol	logical Use gene SYMBOLs (TRUE) or ENSEMBL IDs (FALSE) (default = TRUE)
sep	character Separator for creating unique cell names from sample IDs and cell IDs (default = "!!")
n.cores	integer Number of cores (default = 1)
verbose	logical Print progress (default = TRUE)
unique.names	logical Create unique cell IDs (default = FALSE)

Value

list with sparse count matrices

Examples

```
## Not run:
cms.h5 <- read10xH5(data.path = "/path/to/count/data")

## End(Not run)
```

Index

CRMetrics, [2](#)

read10xH5, [42](#)