Package: SpaceTrooper (via r-universe)

October 14, 2026

Type Package

Title SpaceTrooper performs Quality Control analysis of Image-Based spatial

Version 1.3.3

Description SpaceTrooper performs Quality Control analysis using data driven GLM models of Image-Based spatial data, providing exploration plots, QC metrics computation, outlier detection. It implements a GLM strategy for the detection of low quality cells in imaging-based spatial data (Transcriptomics and Proteomics). It additionally implements several plots for the visualization of imaging based polygons through the ggplot2 package.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.5.0), SpatialExperiment

Imports DropletUtils, S4Vectors, SummarizedExperiment, arrow, data.table, dplyr, e1071, ggplot2, ggpubr, robustbase, scater, scuttle, sf, sfheaders, cowplot, glmnet, rhdf5, methods, rlang, SpatialExperimentIO

Suggests knitr, rmarkdown, BiocStyle, testthat (>= 3.0.0), withr, viridis

biocViews Software, Transcriptomics, GeneExpression, QualityControl, Spatial, SingleCell, DataImport, ImmunoOncology

RoxygenNote 7.3.3

VignetteBuilder knitr

BugReports https://github.com/drighelli/SpaceTrooper/issues

URL https://github.com/drighelli/SpaceTrooper

Config/testthat/edition 3

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libmagick++-dev gsfonts libicu-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev zlib1g-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-10-09 14:25:17 UTC

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

RemoteRef HEAD

RemoteSha 95265069ca8de5eeb03526f29f8debae9dde4d13

Contents

.getActiveGeometryName . . . 3
.renameGeometry . . . 3
.setActiveGeometry . . . 4
addPolygonsToSPE . . . 5
applyQScoreModel . . . 5
checkOutliers . . . 6
computeAreaFromPolygons . . . 7
computeAspectRatioFromPolygons . . . 8
computeCenterFromPolygons . . . 8
computeLambda . . . 9
computeMissingMetricsMerfish . . . 10
computeMissingMetricsXenium . . . 11
computeOutliersQScore . . . 12
computeQScore . . . 13
computeQScoreFlags . . . 15
computeSpatialOutlier . . . 15
computeThresholdFlags . . . 17
computeTrainDF . . . 18
getFencesOutlier . . . 19
getModelFormula . . . 20
plotCellsFovs . . . 20
plotCentroids . . . 22
plotMetricHist . . . 24
plotPolygons . . . 25
plotQScoreTerms . . . 26
plotZoomFovsMap . . . 27
qcFlagPlots . . . 29
readAndAddPolygonsToSPE . . . 30
readCosmxSPE . . . 31
readMerfishSPE . . . 32
readPolygons . . . 34
readPolygonsCosmx . . . 36
readPolygonsMerfish . . . 37
readPolygonsXenium . . . 38
readXeniumSPE . . . 39
SpaceTrooper-deprecated . . . 40
spatialPerCellQC . . . 42
trainModel . . . 43

updateCosmxProteinSPE 44
updateCosmxSPE 46
updateXeniumSPE 47

Index **49**

`.getActiveGeometryName` *.getActiveGeometryName*

Description

.getActiveGeometryName

Usage

```
.getActiveGeometryName(sf)
```

Arguments

`sf`	an sf object

Value

character with the name of the active geometry

Examples

```
example(readPolygonsCosmx)
.getActiveGeometryName(polygons)
```

`.renameGeometry` *.renameGeometry*

Description

renames the ‘from‘ to ‘to‘ geometry of the ‘sf‘ object. If ‘activate‘ is ‘TRUE‘ it set as the active geometry the new geometry name. Default behaviour is to check if the renamed geometry is already active and leave it as active with the new name.

Usage

```
.renameGeometry(sf, from, to, activate = FALSE)
```

Arguments

sf	an sf object with the ‘from‘ geometry
from	character indicating the name of the geometry to change
to	character indicating the new name of the geometry
activate	logical indicating if the renamed geometry has to be activated

Value

an sf object

Examples

```
example(readPolygonsCosmx)
.renameGeometry(polygons, "global", "global1")
```

`.setActiveGeometry` *.setActiveGeometry*

Description

.setActiveGeometry

Usage

```
.setActiveGeometry(sf, name)
```

Arguments

sf	an sf object
name	character for the geometry to activate

Value

an sf object

Examples

```
example(readPolygonsCosmx)
.setActiveGeometry(polygons, "local")
```

addPolygonsToSPE *addPolygonsToSPE*

Description

This function adds polygon data to a 'SpatialExperiment' object.

Usage

```
addPolygonsToSPE(spe, polygons, polygonsCol = "polygons")
```

Arguments

spe	A 'SpatialExperiment' object to which polygons will be added.
polygons	An 'sf' object containing the polygon data.
polygonsCol	character indicating the name of the polygons column to add into the colData (default is 'polygons').

Value

The 'SpatialExperiment' object with polygons added to the 'colData'.

Examples

```
example(readCosmxSPE)
polygons <- readPolygonsCosmx(metadata(spe)$polygons)
spe <- addPolygonsToSPE(spe, polygons)
spe$polygons
```

applyQScoreModel *applyQScoreModel*

Description

Applies a previously trained Quality Score model to a new SpatialExperiment object. See details for important considerations when applying a model to a different dataset.

Usage

```
applyQScoreModel(spe, qsModel, scoreName = "QScore")
```

Arguments

spe	A 'SpatialExperiment' object with QC metrics already computed.
qsModel	A Quality Score model object, usually stored in 'metadata(spe)$QScore_model'.
scoreName	Name of the output column in 'colData'.

Details

The authors do not recommend applying a Quality Score (QS) model trained on one dataset to a different dataset, but this could be useful in some cases, for example to transfer a model trained on a dataset sourced from a tissue slide sample and then applied to other slides from the same experiment. When transferring the model from a reference dataset of the same technology of the query dataset, there is no need to set a common formula. However, when transferring from a different technology, reference QS computation should be performed providing a formula that excludes untransferable or unwanted variables (e.g. computing QS on a reference CosMx dataset omitting the border effect term to transfer the QS model to a Xenium query dataset. This is done by providing a custom formula in computeQScore, see link computeQScore). The function will check for the presence of required model variables in the new dataset and will handle missing or extra variables accordingly.

Value

A 'SpatialExperiment' object with added Quality Score in 'colData'.

Examples

```
example(readCosmxSPE)

## Compute per-cell quality control metrics
spe <- spatialPerCellQC(spe)

## Split the object only for example purposes
set.seed(1998)
idx <- sample(seq_len(ncol(spe)), floor(ncol(spe) / 2))
spe_train <- spe[, idx]
spe_test <- spe[, -idx]

## Train the Quality Control (QC) score model on one dataset
spe_train <- computeQScore(spe_train)
qs_model <- metadata(spe_train)$QScore_model

## Apply the trained model to another dataset
spe_test <- applyQScoreModel(
    spe=spe_test,
    qsModel=qs_model,
    scoreName="QScore_transferred"
)

summary(spe_test$QScore_transferred)
```

`checkOutliers` *checkOutliers*

Description

Checks if computed outliers meet the minimum numerical requirement, being at least 0.1% of total cells for each metric to be used in Quality Score formula. If the requirement is not met, the variable is removed from the formula.

Usage

```
checkOutliers(spe, verbose = FALSE)
```

Arguments

spe	A 'SpatialExperiment' object with spatial omics data.
verbose	Logical. If 'TRUE', prints how many outliers were found for each metric.

Details

The function checks if computed outliers for each metric meet the minimum number to get the metric included in the Quality Score formula. If verbose is TRUE, it also prints how many outliers were found for each metric.

Value

The 'SpatialExperiment' object with added metric variables that will be used for Quality Score computation stored in the object' metadata'.

Examples

```
example(computeOutliersQScore)
spe <- checkOutliers(spe, verbose = TRUE)
metadata(spe)$formula_variables
```

---

computeAreaFromPolygons

computeAreaFromPolygons

Description

This function computes the area from polygon data.

Usage

```
computeAreaFromPolygons(polygons)
```

Arguments

polygons	An 'sf' object containing polygon data.

Value

A 'numeric' vector with the area information.

Examples

```
example(readPolygonsMerfish)
area <- computeAreaFromPolygons(polygons)
area
```

computeAspectRatioFromPolygons
computeAspectRatioFromPolygons

Description

This function computes the aspect ratio (width / height) from polygon data.

Usage

```
computeAspectRatioFromPolygons(polygons)
```

Arguments

polygons	An ‘sf‘ object containing polygon data.

Value

A ‘numeric‘ vector with the aspect ratio information.

Examples

```
example(readPolygonsMerfish)
ar <- computeAspectRatioFromPolygons(polygons)
ar
```

computeCenterFromPolygons
computeCenterFromPolygons

Description

This function computes the center coordinates on x and y axis from polygon data and adds it to the ‘colData‘. It is necessary only for Merfish.

Usage

```
computeCenterFromPolygons(polygons, coldata)
```

Arguments

polygons	An ‘sf‘ object containing polygon data.
coldata	A ‘DataFrame‘ containing the ‘colData‘ to which center coordinates information will be added.

Value

A 'DataFrame' with the added center information.

Examples

```
example(readPolygonsMerfish)
coldata <- computeCenterFromPolygons(polygons, colData(spe))
colData(spe) <- coldata
```

`computeLambda` *computeLambda*

Description

Computes optimal ridge regularization parameter λ via k-fold cross-validation.

`computeLambda` performs ridge (L2) logistic regression with k-fold cross-validation to identify the optimal regularization parameter λ for Quality Score (QS) model training.

Usage

```
computeLambda(trainDF, modelFormula)
```

Arguments

`trainDF`	'data.frame' A data frame for QS model training that must include: Predictor columns: All columns referenced in the formula returned by 'getModelFormula()'. 'QScore_train' or historical 'qcscore_train', containing the binary (0/1) response to be modeled.
`modelFormula`	A one-sided formula or character string describing the supported predictor terms.

Details

Complete cases for the formula variables and response are retained. The function constructs a model matrix and passes it to an internal matrix-based implementation that runs ridge logistic-regression cross-validation using 'cv.glmnet' with 'alpha = 0'.

Value

'numeric' The value of λ (i.e., 'lambda.min') from 'cv.glmnet' that minimizes the cross-validation error.

Examples

```
example(computeTrainDF)
modform <- getModelFormula(names(metadata(spe)$formula_variables))
best_lambda <- computeLambda(df_train, modform)
print(best_lambda)
```

computeMissingMetricsMerfish

computeMissingMetricsMerfish

Description

'computeMissingMetricsMerfish()' takes cell metadata and boundary polygons, calculates per-cell area and aspect-ratio, and optionally appends the raw polygon geometries.

Usage

```
computeMissingMetricsMerfish(
  polFile,
  coldata,
  boundariesType = c("parquet", "HDF5"),
  keepPolygons = FALSE,
  polygonsCol = "polygons",
  useVolume = TRUE
)
```

Arguments

polFile	character for the path to the polygon file. Tipically a parquet file or a folder of HDF5 files in the 'metadata(spe)$polygons'.
coldata	'DataFrame' or 'data.frame' Cell metadata with at least a 'cell_id' column.
boundariesType	'character(1)' One of '"HDF5"' or '"parquet"'—passed on to 'readPolygonsMerfish()'.
keepPolygons	'logical(1)' If 'TRUE', cbinds the raw polygon 'sf' columns onto 'coldata'.
polygonsCol	character indicating the name of the polygons column to add into the colData (default is 'polygons').
useVolume	'logical(1)' it assigns the area from the "volume" column. If the column is not present it computes the area from the polygons. Default: 'TRUE'.

Details

'computeMissingMetricsMerfish()' reads polygon geometries via 'readPolygonsMerfish(polFile, type=boundariesType)' where 'polFile' is a Parquet file (parquet) or a folder of HDF5 files (HDF5). It expects 'coldata' to contain at least 'cell_id'.

Behavior: - If 'useVolume=TRUE' and 'coldata' contains a 'volume' column, the returned 'Area_um' is set to 'volume'. Otherwise 'Area_um' is computed from the polygons using 'computeAreaFromPolygons()'. - 'AspectRatio' is computed from polygons using 'computeAspectRatioFromPolygons()'. - If 'keepPolygons=TRUE', geometries are attached to the returned 'DataFrame' under the name given by 'polygonsCol'.

Important: polygons must align (in order or by matching logic) with the rows of 'coldata'; otherwise area/aspect values may be assigned incorrectly.

Value

A ‘DataFrame‘ (or ‘data.frame‘) with: - all columns of ‘coldata‘ - ‘Area_um‘: area/volume of each cell’s polygon - ‘AspectRatio‘: width/height aspect ratio - (optionally) the polygon geometries

Examples

```
example(readMerfishSPE)
cd <- computeMissingMetricsMerfish(metadata(spe)$polygons, colData(spe),
    boundariesType="parquet")
colData(spe) <- cd
cd
```

`computeMissingMetricsXenium` *computeMissingMetricsXenium*

Description

Compute Missing Metrics for Xenium Data

This function computes missing metrics, such as the aspect ratio, from polygon data in a Xenium dataset and optionally appends the polygon data to the resulting ‘colData‘.

Usage

```
computeMissingMetricsXenium(
  polFile,
  colData,
  keepPolygons = FALSE,
  polygonsCol = "polygons"
)
```

Arguments

`polFile`	A character string specifying the file path to the polygon data.
`colData`	A ‘DataFrame‘ containing the ‘colData‘ for the Xenium dataset.
`keepPolygons`	A logical value indicating whether to keep the polygon data in the resulting ‘colData‘. Default is ‘FALSE‘.
`polygonsCol`	character indicating the name of the polygons column to add into the colData (default is ‘polygons‘).

Details

The function reads the polygon data from the specified file, computes the aspect ratio for each polygon, and merges these metrics with the provided ‘colData‘. Optionally, the polygon data can be kept in the returned ‘colData‘.

Value

A 'DataFrame' containing the updated 'colData' with computed metrics. If 'keepPolygons' is 'TRUE', the polygon data is also included.

Examples

```
example(readXeniumSPE)
colData(spe) <- computeMissingMetricsXenium(metadata(spe)$polygons,
    colData(spe), keepPolygons=TRUE)
```

`computeOutliersQScore` *computeOutliersQScore*

Description

Computes outlier cells for each metric that can be used in Quality Score formula for SpatialExperiment.

This function calculates outlier cells for each variable specified in 'metricList' for a 'SpatialExperiment'. log2SignalDensity must be present in the 'colData' of the 'SpatialExperiment' object as a minimum requirement. The user can choose which metrics to include among the following: Area_um, log2Ctrl_total_ratio, log2AspectRatio. For Xenium and Merfish datasets, even if outliers for log2AspectRatio are detected, neither the metric will be included in the Quality Score formula nor nor the outliers will be used for training.

Usage

```
computeOutliersQScore(
  spe,
  metricList = c("log2SignalDensity", "Area_um", "log2AspectRatio",
    "log2Ctrl_total_ratio")
)
```

Arguments

`spe`	A 'SpatialExperiment' object with spatial omics data.
`metricList`	A character vector specifying the metrics to include in the Quality Score formula. Default is 'c("log2SignalDensity", "Area_um", "log2AspectRatio", "log2Ctrl_total_ratio")'.

Details

The function computes outliers for each specified metric after automatically choosing the appropriate method according to the skewness of the distribution. Internally the function:

1. Calls `.checkSkw()` to choose the proper outlier detection method according to the variable skewness;
2. Calls `computeSpatialOutlier()` on each included metric to get fences;
3. Labels cells as "LOW"/"HIGH" outliers or "NO".

Value

The 'SpatialExperiment' object with added outlier variables in 'colData' and the temporary metrics considered for Quality Score computation stored in the object 'metadata'.

Examples

```
example(readCosmxSPE)
spe <- spatialPerCellQC(spe)
spe <- computeOutliersQScore(spe)
table(spe$log2SignalDensity_outlier_train)
```

`computeQScore` *computeQScore*

Description

Computes Quality Score (QS) and automatically define weights of the QS model terms through glm training. This function computes QS with a formula defined on the metrics "log2SignalDensity", "Area_um", "log2AspectRatio", "log2Ctrl_total_ratio", as computed from the 'spatialPerCellQC' function. It automatically computes the number of outliers for each available metric, which are needed for the glm training. See Details for further information.

Usage

```
computeQScore(spe, bestLambda = NULL, modelFormula = NULL, verbose = FALSE)
```

Arguments

`spe`	A 'SpatialExperiment' object with spatial omics data.
`bestLambda`	the best lambda typically computed using 'computeLambda'.
`modelFormula`	A one-sided formula or character string. If 'NULL', the default formula is generated from available supported metrics and their outliers. If supplied, its supported terms and interactions are fitted without rewriting. See Details.
`verbose`	logical for having a verbose output. Default is FALSE.

Details

For all the technologies, the QS formula can contain only the following terms at most:

Quality Score ~ signal density - size - border effect - background signal

Where signal density is the total counts/protein intensities-to-size ratio, border effect is represented by the aspect ratio (i.e. width/height of the cell) and is considered only for cells with a distance from the FOV border less than 50 px and only for CosMx datasets. Background signal is represented by the ratio of control probe counts to total counts (protein intensities) and cell size is the area for CosMx and Xenium, while it is the volume for Merfish. For each couple of variables interaction terms are computed.

Note that the function is responsible for automatically including/excluding metrics in the formula based on their availability in the ‘colData‘ of the ‘SpatialExperiment‘ object.

For the default formula, inclusion of metrics also depends on the number of available outliers. If the number of outliers for a metric is less than 0.1% of the dataset, that metric is excluded from the default QS formula.

- Model fitting: ridge (L2) logistic regression is fitted (via ‘glmnet‘) on the balanced training set. The function uses ‘trainModel()‘ for fitting and ‘computeLambda()‘ to select lambda unless ‘best-Lambda‘ is supplied.

- Lambda details: because of the randomness in the training set selection, results may vary. The easiest way to ensure reproducibility is to let lambda be computed internally, just set a seed with ‘set.seed()‘ before running ‘computeQScore‘. Otherwise, a fixed value of lambda previously computed with ‘computeLambda‘ preceeded by ‘computeTrainDF‘ and ‘getModelFormula‘ can be set.

- Model formula details: the most complete default formula is: ‘~(log2SignalDensity + Area_um + I(abs(log2AspectRatio) * as.numeric(dist_border < 50)) + log2Ctrl_total_ratio)^2‘. A user-supplied formula is fitted as supplied: additive structures, selected interactions, ‘:‘ and ‘*‘ are preserved. The formula must include ‘log2SignalDensity‘ and may use any subset of the other supported predictors: ‘Area_um‘, ‘log2AspectRatio‘, and ‘log2Ctrl_total_ratio‘. Arbitrary predictors and transformations are rejected. The supported CosMx-only border expression is ‘I(abs(log2AspectRatio) * as.numeric(dist_border < 50))‘.

When a custom ‘modelFormula‘ is supplied, the supported base QC metrics in that formula are also used to compute outliers, define good and bad training cells, and construct ‘QScore_train‘. Interactions do not create additional training metrics. In the supported border expression, ‘dist_border‘ is an auxiliary variable and ‘log2AspectRatio‘ is the corresponding training QC metric. Supported metrics omitted from the custom formula do not contribute to training-label construction. When ‘modelFormula=NULL‘, the established default metric-selection and training-label workflow is unchanged.

The computed model output is stored in ‘metadata(spe)$QScore_model‘. For inspection and model coefficient transfer between datasets see also `applyQScoreModel`. However, transferring models between datasets is not recommended. As described in the paper, QS model training is dataset-specific and does not generalize well across datasets. Moreover, this step is computationally efficient. Please, refer to the paper for details on scalability and coefficient transfer.

Value

The ‘SpatialExperiment‘ object with added Quality Score in ‘colData‘.

Examples

```
example(spatialPerCellQC)
set.seed(1998)
spe <- computeQScore(spe)
summary(spe$QScore)
```

computeQScoreFlags *computeQScoreFlags*

Description

Computes flagged cells based on a manually chosen threshold on Quality Score (QS).

This function computes flagged cells based on a manually chosen threshold on QS stored in 'SpatialExperiment' object.

Usage

```
computeQScoreFlags(spe, qsThreshold = 0.5, useQSQuantiles = FALSE)
```

Arguments

spe	A 'SpatialExperiment' object with spatial omics data.
qsThreshold	Numeric threshold or quantile for QS. Default is '0.5'.
useQSQuantiles	Logical; if 'TRUE', treat 'qsThreshold' as a percentile.

Value

The 'SpatialExperiment' object with added low-quality flagged cells as TRUE/FALSE in 'colData'.

Examples

```
example(computeQScore)
spe <- computeQScoreFlags(spe)
table(spe$low_QScore)
# if fixed filters are defined we have an additional column
spe <- computeThresholdFlags(spe)
spe <- computeQScoreFlags(spe)
table(spe$low_threshold_QScore)
```

computeSpatialOutlier *computeSpatialOutlier*

Description

Computes outliers for quantitative cell metadata based on a statistical test: medcouple ('mc' method argument) or the Median Absolute Deviation (MAD, 'scuttle' method argument), when the metric distribution is respectively asymmetric or symmetric.

Usage

```
computeSpatialOutlier(
  spe,
  computeBy = NULL,
  method = c("mc", "scuttle", "both"),
  mcDoScale = FALSE,
  scuttleType = c("both", "lower", "higher")
)
```

Arguments

spe	a SpatialExperiment object with quantitative cell metrics in the ‘colData‘.
computeBy	character indicating a ‘colData‘ column name to compute the outliers on.
method	one of ‘mc‘, ‘scuttle‘, ‘both‘. Use ‘mc‘ for medcouple, ‘scuttle‘ for MAD as computed in ‘scuttle‘, ‘both‘ for computing both of them.
mcDoScale	logical indicating if the metric values should be scaled to compute medcouple (default is FALSE, as suggested by the original medcouple authors). See mc for further readings.
scuttleType	One of ‘"both"‘, ‘"lower"‘, ‘"higher"‘ for scuttle method.

Details

The medcouple method is a measure for the skeweness of univariate distribution as described in Hubert M. et al. (2008). In particular, the computed medcouple value must be in a range between -0.6 and 0.6 to compute adjusted boxplots and perform the outlier detection. For MAD method we just wrap the isOutlier function in the scuttle package. Please see McCarthy DJ et al (2017) for further details.

Value

a SpatialExperiment object with additional column(s) (named as the column name indicated in ‘computeBy‘ followed by the outlier_sc/mc). Resulting columns are ‘outlier.filter‘ logical class objects, allowing to store the thresholds as attributes of the columns. Use attr(,"thresholds") to retrieve them.

Examples

```
example(spatialPerCellQC)
spe <- computeSpatialOutlier(spe, computeBy="log2SignalDensity", method="both")
table(spe$log2SignalDensity_outlier_mc)
table(spe$log2SignalDensity_outlier_sc)
```

computeThresholdFlags *computeThresholdFlags*

Description

Computes flagged cells using fixed thresholds for SpatialExperiment.

This function calculates flagged cells only for total counts (protein intensities) and control-to-total probe counts ratio using fixed thresholds for a 'SpatialExperiment' object.

Usage

```
computeThresholdFlags(spe, totalThreshold = 0, ctrlTotRatioThreshold = 0.1)
```

Arguments

spe	A 'SpatialExperiment' object with spatial omics data.
totalThreshold	A numeric value for the threshold of total counts to identify cells with low counts. Default is '0'.
ctrlTotRatioThreshold	A numeric value for the threshold of control-to-total ratio to flag cells over a certain threshold. Default is '0.1'.

Details

The function flags cells basing on zero counts and control-to-total ratio to identify junk cells. It also combines these two flags into a single flag metric named 'threshold_flags'.

Value

The 'SpatialExperiment' object with added flagged cells as TRUE/FALSE in 'colData'.

Examples

```
example(readCosmxSPE)
spe <- spatialPerCellQC(spe)
spe <- computeThresholdFlags(spe)
table(spe$threshold_flags)
```

computeTrainDF *computeTrainDF*

Description

Builds a balanced training data frame from a SpatialExperiment.

computeTrainDF takes a SpatialExperiment object and assembles a balanced training set of "good" vs "bad" cells for subsequent model fitting.

Usage

```
computeTrainDF(colData, formulaVars, tech, verbose = FALSE)
```

Arguments

colData	A per-cell metadata table. Typically 'as.data.frame(colData(spe))'. Must include at least: 'cell_id', raw metric columns named as in 'formulaVars' (e.g. 'log2SignalDensity', 'Area_um', 'log2Ctrl_total_ratio', optionally 'log2AspectRatio' and 'dist_border'), and the corresponding outlier-label columns referenced by 'formulaVars'.
formulaVars	A named character vector mapping variable name to its outlier label column name, e.g. 'c(log2SignalDensity="log2SignalDensity_outlier_train", ...)'.
tech	Character string with the acquisition technology. Used to enable CosMx-specific inclusion of border effect. Expected values include '"Nanostring_CosMx"' or '"Nanostring_CosMx_Protein"'.
verbose	'logical(1)' (default FALSE) If TRUE, prints the number of "bad" and "good" cells selected.

Details

The function builds a training set using the variables specified in the 'metadata' of the 'SpatialExperiment' object.

Value

A data.frame with one row per cell, including: QScore_train and its compatibility alias qcscore_train (0/1) indicating "bad" vs "good", relevant colData columns used for modeling. Deduplicates and down-samples "good" cells to match the number of "bad" cells.

Examples

```
example(spatialPerCellQC)
spe <- computeOutliersQScore(spe)
spe <- checkOutliers(spe)
df_train <- computeTrainDF(colData(spe), metadata(spe)$formula_variables,
    metadata(spe)$technology)
table(df_train$QScore_train)
```

getFencesOutlier *getFencesOutlier*

Description

Retrieve Threshold (Fence) Values from a SpatialExperiment Object

This function extracts the threshold values, also known as fences, from a specified column in the ‘colData‘ of a ‘SpatialExperiment‘ object.

Usage

```
getFencesOutlier(
  spe,
  fencesOf,
  highLow = c("both", "lower", "higher"),
  decimalRound = NULL
)
```

Arguments

spe	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
fencesOf	A character string specifying the name of the column in ‘colData(spe)‘ from which to extract the fence values. This column should contain an ‘outlier.filter‘ object (see ‘computeSpatialOutlier“‘).
highLow	character indicating which fence to get if "higher", "lower" or "both" (default is "both").
decimalRound	An optional integer specifying the number of decimal places to which the fence values should be rounded. If ‘NULL‘, no rounding is applied. Default is ‘NULL‘.

Value

A numeric vector containing the lower and upper threshold values extracted from the specified column.

Examples

```
example(computeSpatialOutlier)
getFencesOutlier(spe, fencesOf="log2SignalDensity_outlier_mc")
```

getModelFormula *getModelFormula*

Description

Returns the right-hand side of a model formula string based on a vector of metric names.

Usage

```
getModelFormula(formulaVars, verbose = FALSE, ...)
```

Arguments

formulaVars	A named character vector mapping metric names to outlier columns, as historically stored in ‘metadata(spe)$formula_variables‘. An unnamed character vector of metric names is also accepted.
verbose	Logical. If ‘TRUE‘, prints the generated formula.
...	May contain the named replacement ‘metricList‘. If both ‘formulaVars‘ and ‘metricList‘ are supplied, ‘metricList‘ takes precedence with a warning.

Value

‘character‘ A one-sided formula as a string (e.g. "~ log2SignalDensity + ...").

Examples

```
example(checkOutliers)
getModelFormula(metadata(spe)$formula_variables)
```

plotCellsFovs *plotCellsFovs*

Description

Plot cell centroids in FoVs, creating a map of the whole experiment, where cells are plotted as points and FoV boundaries and numbers are overlaid.

Usage

```
plotCellsFovs(
  spe,
  sampleId = unique(spe$sample_id),
  pointCol = "firebrick",
  numbersCol = "black",
  alphaNumbers = 0.8,
```

```
  fovDim = metadata(spe)$fov_dim,
  size = 0.05,
  alpha = 0.8,
  scaleBar = TRUE,
  micronConvFact = 0.12,
  pointSize = NULL,
  pointAlpha = NULL,
  numberSize = 1,
  numbersAlpha = NULL
)
```

Arguments

spe	A 'SpatialExperiment' object with 'fov' in 'colData'.
sampleId	Character string identifying which sample to plot. Default: 'unique(spe$sample_id)'.
pointCol	Color for the cell centroids. Default: '"firebrick"'.
numbersCol	Color for the FoV labels. Default: '"black"'.
alphaNumbers	Deprecated alias for 'numbersAlpha'.
fovDim	numeric with two named dimensions xdim, ydim. (Default is metadata(spe)$fov_dim)
size	Deprecated alias for 'pointSize'.
alpha	Deprecated alias for 'pointAlpha'.
scaleBar	A logical value indicating whether to add a scale bar to the plot. (Default is 'TRUE')
micronConvFact	Numeric conversion factor from pixels to microns. Default is '0.12'.
pointSize	Numeric point size for the cell centroids. If supplied, takes precedence over 'size'.
pointAlpha	Numeric transparency for the cell centroids. If supplied, takes precedence over 'alpha'.
numberSize	Numeric size for the FoV labels. Default: '1'.
numbersAlpha	Numeric transparency for FoV labels. If supplied, takes precedence over 'alphaNumbers'.

Details

The function expects spe (a SpatialExperiment) to include field-of-view metadata in metadata(spe): - 'metadata(spe)$fov_positions': a matrix or data.frame (or list with named elements) containing at minimum 'x_global_px', 'y_global_px', and 'fov'. Values 'x_global_px'/'y_global_px' are in pixels and represent the origin (bottom-left) of each FoV. - 'metadata(spe)$fov_dim' (or the 'fovDim' argument): a named numeric with 'xdim' and 'ydim' giving FoV width/height in pixels.

For each FoV the rectangle is drawn as: - 'xmin' = 'x_global_px', - 'xmax' = 'x_global_px' + 'xdim', - 'ymin' = 'y_global_px', - 'ymax' = 'y_global_px' + 'ydim'.

'sampleId': if 'NULL' no plot title is added; if a vector of length > 1 is provided the first element is used. Prefer passing a single sample identifier string. 'micronConvFact' is the conversion factor from pixels to micrometers (µm/pixel). It is forwarded to '.plotScaleBar()' and controls how the

scale bar is labeled (useful for technologies where coordinates are in pixels but the scale should display μm). The function uses the coordinates returned by ‘spatialCoords(spe)‘ and the names from ‘spatialCoordsNames(spe)‘. It performs basic input checks but assumes the described metadata structures are present; if they are missing or malformed the function will raise an informative error.

Value

A ‘ggplot‘ object showing cell centroids and FoV boundaries.

Examples

```
example(readCosmxSPE)
g <- plotCellsFovs(spe)
print(g)
```

`plotCentroids` *plotCentroids*

Description

Plot Spatial Coordinates for a SpatialExperiment Object This function generates a ggplot of spatial coordinates from a ‘SpatialExperiment‘ object, optionally coloring the points by a specified column in ‘colData‘.

Usage

```
plotCentroids(
  spe,
  colourBy = NULL,
  colourLog = FALSE,
  sampleId = unique(spe$sample_id),
  isNegativeProbe = FALSE,
  palette = NULL,
  pointCol = "darkmagenta",
  size = 0.05,
  alpha = 0.8,
  aspectRatio = 1,
  scaleBar = TRUE,
  micronConvFact = 0.12
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
`colourBy`	An optional character string specifying the column in ‘colData(spe)‘ to use for coloring the points. If ‘NULL‘, all points will be colored the same.
`colourLog`	Logical to log-transform the data to enhance visualization (Default is FALSE).

sampleId	A character string specifying the sample identifier to be used as the plot title. (Default is the unique sample ID from 'spe')
isNegativeProbe	A logical value indicating whether to apply a custom color gradient for negative probe data. (Default is 'FALSE')
palette	A vector of colors to be used as a custom palette. For categorical data, this should be a vector of colors with the same length as the number of levels in 'colourBy'. For continuous data, this should be a vector of colors used to create a gradient.
pointCol	A character string specifying the color of the points when 'colourBy' is 'NULL'. (Default is '"darkmagenta"')
size	A numeric value specifying the size of the points. (Default is '0.05')
alpha	A numeric value specifying the transparency level of the points. (Default is '0.8')
aspectRatio	A numeric value specifying the aspect ratio of the plot. (Default is '1')
scaleBar	A logical value indicating whether to add a scale bar to the plot. (Default is 'TRUE')
micronConvFact	Numeric conversion factor from pixels to microns. DEFAULT is '0.12'.

Details

This function plots cell centroids from a 'SpatialExperiment' object using coordinates returned by 'spatialCoords(spe)' and the coordinate names from 'spatialCoordsNames(spe)' (the function expects at least two spatial coordinate dimensions). If 'colourBy' is 'NULL' all points are drawn using 'pointCol'; otherwise the function colors points using the specified 'colData' column or a provided palette.

Requirements and inputs: - 'spe' must be a 'SpatialExperiment' with valid spatial coordinates. - If 'colourBy' is provided and is the name of a 'colData' column, that column must exist and be suitable for plotting (factor/logical for discrete palettes, numeric for continuous palettes). - 'palette' may be either (a) a vector of color values (used directly) or (b) the name of a column in 'colData(spe)' from which a palette will be built via 'createPaletteFromColData()'. The function detects which mode to use at runtime.

Value

A 'ggplot' object representing the spatial coordinates plot of polygon centroids.

Examples

```
example(readCosmxSPE)
g <- plotCentroids(spe, colourBy="Mean.DAPI")
print(g)
```

plotMetricHist *plotMetricHist*

Description

Plot a Histogram for a Given Metric in a SpatialExperiment Object

This function generates a histogram for a specified metric in a ‘SpatialExperiment‘ object.

Usage

```
plotMetricHist(
  spe,
  metric,
  fillColor = "#c0c8cf",
  useFences = NULL,
  fencesColors = c(lower = "purple4", higher = "tomato"),
  bins = 30,
  binWidth = NULL
)
```

Arguments

spe	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
metric	A character string specifying the name of the metric (column in ‘colData(spe)‘) to plot.
fillColor	A character string specifying the fill color of the histogram bars. (Default is ‘"#69b3a2"‘)
useFences	A character string specifying the name of the column in ‘colData(spe)‘ that contains the fence thresholds (typically from an outlier filter). If ‘NULL‘, no fences will be plotted. (Default is ‘NULL‘)
fencesColors	A named character vector specifying the colors to use for the lower and higher fences. The names should be ‘"lower"‘ and ‘"higher"‘ . (Default is ‘c("lower"="purple4", "higher"="tomato")‘)
bins	An integer specifying the number of bins to use in the histogram. (Default is ‘30‘)
binWidth	A numeric value specifying the width of the bins. If ‘NULL‘, the bin width will be automatically determined based on the ‘bins‘ parameter. (Default is ‘NULL‘)

Value

A ‘ggplot‘ object representing the histogram of the specified metric.

Examples

```
example(readCosmxSPE)
g <- plotMetricHist(spe, metric="Mean.DAPI")
print(g)
```

plotPolygons *plotPolygons*

Description

Plot polygons from a 'SpatialExperiment' object using ggplot2.

Usage

```
plotPolygons(
  spe,
  colourBy = "darkgrey",
  colourLog = FALSE,
  polyColumn = "polygons.global",
  sampleId = unique(spe$sample_id),
  bgColor = "white",
  fillAlpha = 1,
  palette = NULL,
  borderCol = NA,
  borderAlpha = 1,
  borderLineWidth = 0.1,
  drawBorders = TRUE,
  scaleBar = TRUE,
  micronConvFact = 0.12
)
```

Arguments

spe	A 'SpatialExperiment' object with polygon data as an 'sf' object.
colourBy	A column in 'colData(spe)' for coloring the polygons or a string color in colors(). (Default is "darkgrey")
colourLog	Logical to log-transform the data to enhance visualization (Default is FALSE).
polyColumn	character for the name of the column where the polygons sf are stored (default is "polygons.global")
sampleId	Sample ID for plot title. Default is the unique sample ID.
bgColor	character indicating color for the background (default is "white")
fillAlpha	Transparency level for polygon fill. Default is '1'.
palette	Colors to use if 'colourBy' is a factor. Default is 'NULL'.
borderCol	Color of polygon borders. Default is '"black"'.
borderAlpha	Transparency level for borders. Default is '1'.
borderLineWidth	Width of polygon borders. Default is '0.1'.
drawBorders	Logical; whether to draw borders. Default is 'TRUE'.

scaleBar	A logical value indicating whether to add a scale bar to the plot. (Default is 'TRUE')
micronConvFact	Numeric conversion factor from pixels to microns. DEFAULT is '0.12'.

Details

Renders polygon geometries stored in 'colData' of a 'SpatialExperiment' object using 'ggplot2::geom_sf()' and provides options for fill, borders and an optional scale bar.

Input expectations: - 'spe' must be a 'SpatialExperiment' and 'polyColumn' must name a column in 'colData(spe)' that contains polygon geometries ('sfc'/'sf' or a list of geometry objects) suitable for 'geom_sf()'. - 'colourBy' supports two modes: (a) the name of a 'colData' column to map fill to data values, or (b) a literal colour name (e.g. '"darkgrey"') used as a constant fill.

Colour / palette behaviour: - If 'colourBy' refers to a 'colData' column and 'colourLog = TRUE', the function computes 'log1p()' of that column in the local plotting data frame (it does not mutate 'spe'). - 'palette' may be a vector of colour values (used for discrete scales). Numeric 'colourBy' values use a continuous Viridis-like scale by default.

Scale bar and units: - If 'scaleBar = TRUE' the function calls '.plotScaleBar()' to add a length scale. 'micronConvFact' is the conversion factor (μm per pixel) used by '.plotScaleBar()' to label the bar in micrometres when appropriate.

Robustness notes: - If 'colourBy' is neither a 'colData' column nor a valid colour name the function issues a warning and falls back to the default fill.

Value

A 'ggplot' object representing the polygon plot of the spatial data.

Examples

```
example(readAndAddPolygonsToSPE)
plotPolygons(spe, colourBy="Mean.DAPI")
```

`plotQScoreTerms` *plotQScoreTerms*

Description

Plots the individual terms that combine into the quality score formula, allowing assessment of each term's impact on the final score.

Usage

```
plotQScoreTerms(
  spe,
  sampleId = unique(spe$sample_id),
  size = 0.05,
  alpha = 0.8,
```

```
  aspectRatio = 1,
  custom = FALSE
)
```

Arguments

spe	A 'SpatialExperiment' object with 'quality_score' and term columns in 'colData'.
sampleId	Character string for plot title. Must match values in the 'fov' column of 'colData(spe)'. Default: 'unique(spe$sample_id)'.
size	Numeric point size for the scatter plots. Default: '0.05'.
alpha	Numeric transparency for the scatter plots. Default: '0.2'.
aspectRatio	Numeric aspect ratio of the plots. Default: '1'.
custom	Logical; if 'TRUE', use custom polygon-derived metrics.

Value

A combined plot (via 'cowplot::plot_grid') showing spatial maps of each QS term.

Examples

```
example(readAndAddPolygonsToSPE)
example(spatialPerCellQC)
p <- plotQScoreTerms(spe)
print(p)
```

---

plotZoomFovsMap *plotZoomFovsMap*

Description

Plot Zoomed-in FOVs with Map and Polygons

This function generates a plot that shows a map of all fields of view (FOVs) within a 'SpatialExperiment' object, alongside a zoomed-in view of the specified FOVs with an overlay of polygons and optional coloring.

Usage

```
plotZoomFovsMap(
  spe,
  fovs = NULL,
  title = NULL,
  mapPointCol = "darkmagenta",
  mapNumbersCol = "black",
  mapAlphaNumbers = 0.8,
  csize = 0.05,
```

```
  calpha = 0.8,
  scaleBars = NULL,
  scaleBarMap = TRUE,
  scaleBarPol = TRUE,
  ...,
  mapPointSize = NULL,
  mapPointAlpha = NULL,
  fovNumbersCol = NULL,
  fovNumberSize = 1,
  fovNumbersAlpha = NULL
)
```

Arguments

spe	A 'SpatialExperiment' object containing spatial transcriptomics data.
fovs	A character vector specifying the FOVs to be zoomed in and plotted. Must match values in the 'fov' column of 'colData(spe)'.
title	An optional character string specifying the title of the final plot. If 'NULL', no title is added. Default is 'NULL'.
mapPointCol	A character string specifying the color of the points in the map. Default is '"darkmagenta"'.
mapNumbersCol	Deprecated alias for 'fovNumbersCol'.
mapAlphaNumbers	Deprecated alias for 'fovNumbersAlpha'.
csize	Deprecated alias for 'mapPointSize'.
calpha	Deprecated alias for 'mapPointAlpha'.
scaleBars	Logical or NULL. Default is 'NULL'. Master switch controlling the presence of scale bars in both panels. If TRUE, scale bars are shown in both the map and polygon panels. If FALSE, scale bars are hidden in both panels. If NULL, individual settings defined by scaleBarMap and scaleBarPol are used.
scaleBarMap, scaleBarPol	Logical. Default is TRUE. Control the presence of the scale bar in the map (cells/FOV overview) and polygon (segmentation) panels, respectively. These parameters are only used when scaleBars is NULL; otherwise they are overridden by scaleBars.
...	Additional arguments passed to 'plotPolygons'.
mapPointSize	Numeric size for points in the map. If supplied, takes precedence over 'csize'.
mapPointAlpha	Numeric transparency for points in the map. If supplied, takes precedence over 'calpha'.
fovNumbersCol	Color for FoV labels. If supplied, takes precedence over 'mapNumbersCol'.
fovNumberSize	Numeric size for the FoV labels. Default: '1'.
fovNumbersAlpha	Transparency for FoV labels. If supplied, takes precedence over 'mapAlphaNumbers'.

Details

The function first filters the ‘SpatialExperiment‘ object to the specified FOVs, generates a plot of the cells for the entire map, then creates a detailed polygon plot of the selected FOVs, and finally combines these into a single side-by-side visualization. If ‘title‘ is not ‘NULL‘, it adds a title to the combined plot.

Value

A combined plot showing a map of all FOVs with zoomed-in views of the specified FOVs and their associated polygons.

Examples

```
example(readAndAddPolygonsToSPE)
plotZoomFovsMap(spe, fovs=16, title="FOV 16")
```

`qcFlagPlots` *qcFlagPlots*

Description

Plots the flagged cells identified with first filter, based on control count on total count ratio, area in um and DAPI signal.

This function generates a plot that shows selected (FOVs) within a ‘SpatialExperiment‘ object, with cells flagged in different colors over a light or dark layout chosen by the user.

Usage

```
qcFlagPlots(
  spe,
  fov = unique(spe$fov),
  theme = c("light", "dark"),
  custom = FALSE
)
```

Arguments

`spe`	A ‘SpatialExperiment‘ object containing spatial transcriptomics data.
`fov`	An integer or numeric vector specifying the FOVs to be plotted Must match values in the ‘fov‘ column of ‘colData(spe)‘.
`theme`	A character string among "light" or "dark".
`custom`	A boolean value. If TRUE, custom polygons derived metrics will be used.

Value

A panel with multiple plots showing flagged cells for different variables.

Examples

```
example(readAndAddPolygonsToSPE)
spe <- spatialPerCellQC(spe)
spe <- computeThresholdFlags(spe)
p <- qcFlagPlots(spe, fov=16, theme="dark")
print(p)
```

`readAndAddPolygonsToSPE` *readAndAddPolygonsToSPE*

Description

Read and Add Polygons to a SpatialExperiment Object

This function reads polygon boundary data based on the technology associated with the provided SpatialExperiment (SPE) object and adds the polygons to the SPE.

Usage

```
readAndAddPolygonsToSPE(
  spe,
  polygonsCol = "polygons",
  keepMultiPol = TRUE,
  boundariesType = c("csv", "HDF5", "parquet")
)
```

Arguments

`spe`	A `SpatialExperiment` object. The object should contain metadata with the field `"technology"`, specifying the technology used (e.g., "Nanostring_CosMx", "Vizgen_MERFISH", or "10X_Xenium").
`polygonsCol`	character indicating the name of the polygons column to add into the colData (default is 'polygons').
`keepMultiPol`	Logical. If `TRUE`, multi-polygon features will be kept when reading the boundary data. Defaults to `TRUE`.
`boundariesType`	Character. Specifies the type of boundary file format to read. Options are `"HDF5"` or `"parquet"`. Defaults to `"HDF5"`.

Details

The function first checks the technology specified in the SPE's metadata. Based on the technology, it reads the appropriate polygon data using the corresponding reading function: - For "Nanostring_CosMx" or "Nanostring_CosMx_Protein", it uses 'readPolygonsCosmx()'. - For "Vizgen_MERFISH", it uses 'readPolygonsMerfish()'. - For "10X_Xenium", it uses 'readPolygonsXenium()'. After reading the polygons, it adds them to the SPE using 'addPolygonsToSPE()'.

Value

A SpatialExperiment object with the added polygon data.

Examples

```
example(readCosmxSPE)
spe <- readAndAddPolygonsToSPE(spe)
colData(spe)
```

---

readCosmxSPE *readCosmxSPE*

Description

Read and Construct a SpatialExperiment Object from CosMx Data

This function reads in data from Nanostring CosMx files and constructs a 'SpatialExperiment' object, optionally including polygons data.

Usage

```
readCosmxSPE(
  dirName,
  sampleName = "sample01",
  coordNames = c("CenterX_global_px", "CenterY_global_px"),
  countMatFPattern = "exprMat_file.csv",
  metadataFPattern = "metadata_file.csv",
  polygonsFPattern = "polygons.csv",
  fovPosFPattern = "fov_positions_file.csv",
  fovdims = c(xdim = 4256, ydim = 4256),
  keepPolygons = FALSE,
  polygonsCol = "polygons"
)
```

Arguments

dirName	A character string specifying the directory containing the CosMx data files.
sampleName	A character string specifying the sample name. Default is '"sample01"'.
coordNames	A character vector specifying the names of the spatial coordinate columns in the data. Default is 'c("CenterX_global_px", "CenterY_global_px")'.
countMatFPattern	A character string specifying the pattern to match the count matrix file. Default is '"exprMat_file.csv"'.
metadataFPattern	A character string specifying the pattern to match the metadata file. Default is '"metadata_file.csv"'.

polygonsFPattern
: A character string specifying the pattern to match the polygons file. Default is ‘"polygons.csv"‘.

fovPosFPattern
: A character string specifying the pattern to match the FOV positions file. Default is ‘"fov_positions_file.csv"‘.

fovdims
: A named numeric vector specifying the dimensions of the FOV in pixels. Default is ‘c(xdim=4256, ydim=4256)‘.

keepPolygons
: Logical indicating if the polygons need to be loaded into memory or not (Default is ‘FALSE‘).

polygonsCol
: Character name of the column in colData to store polygons.

Details

The function firstly relies on readCosmxSXE to read in the specified files for count matrices, metadata, and FOV positions constructing a ‘SpatialExperiment‘ object. Then it harmonizes the object to have the same metadata as for the other technologies, setting the colData names as required in further QC analysis.

Optionally, polygons data can be read and added to the object by seeting the ‘keepPolygons‘ argument to ‘TRUE‘, otherwise it only stores the polygons file path into the object metadata.

readCosmxProteinSPE is a wrapper of readCosmxSPE, it changes the technology metadata in Nanostring_CosMx_Protein.

Value

A ‘SpatialExperiment‘ object containing the read CosMx data, including count matrices, metadata, and optionally polygons.

Author(s)

Dario Righelli, Benedetta Banzi

Examples

```
cospath <- system.file(file.path("extdata", "CosMx_DBKero_Tiny"),
   package="SpaceTrooper")
spe <- readCosmxSPE(cospath, sampleName="DBKero_Tiny")
```

---

readMerfishSPE *readMerfishSPE*

Description

‘readMerfishSPE()‘ imports MERFISH/Merscore outputs (counts, metadata, and optionally cell boundary polygons) from a directory and builds a SpatialExperiment object.

Usage

```
readMerfishSPE(
  dirName,
  sampleName = "sample01",
  computeMissingMetrics = TRUE,
  keepPolygons = FALSE,
  boundariesType = c("parquet", "HDF5"),
  countmatFPattern = "cell_by_gene.csv",
  metadataFPattern = "cell_metadata.csv",
  polygonsFPattern = "cell_boundaries.parquet",
  coordNames = c("center_x", "center_y"),
  polygonsCol = "polygons",
  useVolume = TRUE
)
```

Arguments

dirName	'character(1)' Path to a folder containing MERFISH output files.
sampleName	'character(1)' Identifier to assign to the 'sample_id' field in the returned object. Default: '"sample01"'.
computeMissingMetrics	'logical(1)' If 'TRUE', compute area and aspect-ratio metrics from the cell boundary polygons. Default: 'TRUE'. In particular for area, if a "volume" column is present in the colData, it will be used as area value, otherwise area will be computed from polygons. This is relevant for MERFISH as the volume is computed on the entire 3D cell available data, while polygons are 2D sections.
keepPolygons	'logical(1)' If 'TRUE', attach raw polygon geometries as extra columns in 'colData'. Default: 'FALSE'.
boundariesType	'character(1)' One of '"parquet"' or '"HDF5"'. If '"parquet"', reads a single Parquet file of boundaries; if '"HDF5"', uses a folder of HDF5 polygon files.
countmatFPattern	'character(1)' Regex passed to 'list.files()' to find the count matrix CSV. Default: '"cell_by_gene.csv"'.
metadataFPattern	'character(1)' Pattern to find the cell metadata CSV. Default: '"cell_metadata.csv"'.
polygonsFPattern	'character(1)' Pattern to find the cell boundaries file. Default: '"cell_boundaries.parquet"'.
coordNames	'character(2)' Names of the columns in 'colData' that store X/Y spatial coordinates. Default: 'c("center_x", "center_y")'.
polygonsCol	character indicating the name of the polygons column to add into the colData (default is 'polygons').
useVolume	'logical(1)' If 'TRUE', prefer a "volume" column in the metadata for area (when present). Default: 'TRUE'.

Details

The function searches ‘dirName‘ for three resources using the provided filename patterns: the count matrix (‘countmatFPattern‘), the cell metadata (‘metadataFPattern‘), and the polygon resource (‘polygonsFPattern‘). The count matrix and metadata are expected to contain a ‘cell_id‘ column (aliases ‘V1‘, ‘cell‘, or ‘EntityID‘ are renamed). If ‘computeMissingMetrics‘ is TRUE, ‘computeMissingMetricsMerfish()‘ is invoked and receives ‘useVolume‘.

The returned ‘SpatialExperiment‘ contains: - ‘assays$counts‘: gene × cell count matrix (rows = genes, cols = ‘cell_id‘); - ‘colData‘: per-cell metadata (may be reordered by the function); - spatial coordinates named by ‘coordNames‘; - ‘metadata$polygons‘: path(s) matched by ‘polygonsFPattern‘; - ‘metadata$technology‘: ‘"Vizgen_MERFISH"‘.

Side effects: the function may reorder ‘colData‘ columns, attach polygon file paths to ‘metadata(spe)$polygons‘, and (when requested) append ‘Area_um‘ and ‘AspectRatio‘ to ‘colData‘.

Value

A ‘SpatialExperiment‘ object with: - ‘assays$counts‘: gene × cell count matrix - ‘colData‘: per-cell metadata (including computed metrics) - spatial coordinates named by ‘coordNames‘ - ‘metadata$polygons‘: path to the boundaries file - ‘metadata$technology‘: ‘"Vizgen_MERFISH"‘.

Author(s)

Dario Righelli, Benedetta Banzi

Examples

```
path <- system.file("extdata", "Merfish_Tiny",
                    package = "SpaceTrooper")
spe <- readMerfishSPE(
  dirName = path,
  sampleName = "Patient2",
  keepPolygons = TRUE,
  boundariesType = "parquet"
)
spe
```

`readPolygons` *readPolygons*

Description

Read and Validate Polygons from a File

This function reads polygon data from a specified file, validates the polygons, and returns them as an ‘sf‘ object. It supports multiple file formats and can handle both global and local coordinates.

Usage

```
readPolygons(
  polygonsFile,
  type = c("csv", "parquet", "h5"),
  x = c("x_global_px", "vertex_x"),
  y = c("y_global_px", "vertex_y"),
  xloc = "x_local_px",
  yloc = "y_local_px",
  keepMultiPol = TRUE,
  verbose = FALSE
)
```

Arguments

polygonsFile	A character string specifying the path to the polygon file.
type	A character string specifying the file type. Supported types are ‘"csv"‘, ‘"parquet"‘, and ‘"h5"‘. Default is ‘"csv"‘.
x	A character vector specifying the column names for the x-coordinates in the polygon data. Default is ‘c("x_global_px", "vertex_x")‘.
y	A character vector specifying the column names for the y-coordinates in the polygon data. Default is ‘c("y_global_px", "vertex_y")‘.
xloc	A character string specifying the column name for the local x-coordinates. Default is ‘"x_local_px"‘.
yloc	A character string specifying the column name for the local y-coordinates. Default is ‘"y_local_px"‘.
keepMultiPol	A logical value indicating whether to keep multipolygons during validation. Default is ‘TRUE‘.
verbose	A logical value indicating whether to print additional information during processing. Default is ‘FALSE‘.

Details

The function reads polygon data from the specified file and formats. It validates the polygons and handles both global and local coordinates if provided. If the file type is ‘"h5"‘, the function currently does not handle the data, as this part of the code is not implemented.

Value

An ‘sf‘ object with the loaded and validated polygons.

Examples

```
example(readCosmxSPE)
polygons <- readPolygons(metadata(spe)$polygons)
polygons
```

readPolygonsCosmx *readPolygonsCosmx*

Description

This function reads polygon data specific to CosMx technology.

Usage

```
readPolygonsCosmx(
  polygonsFile,
  type = c("csv", "parquet"),
  x = "x_global_px",
  y = "y_global_px",
  xloc = "x_local_px",
  yloc = "y_local_px",
  keepMultiPol = TRUE,
  verbose = FALSE
)
```

Arguments

polygonsFile	A character string specifying the file path to the polygon data.
type	A character string specifying the file type ("csv" or "parquet").
x	A character string specifying the x-coordinate column.
y	A character string specifying the y-coordinate column.
xloc	A character string specifying the local x-coordinate column.
yloc	A character string specifying the local y-coordinate column.
keepMultiPol	A logical value indicating whether to keep multipolygons.
verbose	A logical value indicating whether to print additional information.

Value

An ‘sf‘ object containing the CosMx polygon data.

Examples

```
example(readCosmxSPE)
polygons <- readPolygonsCosmx(metadata(spe)$polygons)
polygons
```

readPolygonsMerfish *readPolygonsMerfish*

Description

This function reads polygon data specific to MERFISH technology.

Usage

```
readPolygonsMerfish(
  polygons,
  type = c("parquet", "HDF5"),
  keepMultiPol = TRUE,
  hdf5pattern = "hdf5",
  zLev = 3L,
  zcolumn = "ZIndex",
  geometry = "Geometry",
  verbose = FALSE
)
```

Arguments

polygons	A character string specifying the folder containing the polygon data files in case of HDF5, or a path to a parquet file (see 'type').
type	A character string specifying the file type("HDF5" or "parquet"). Default is parquet.
keepMultiPol	A logical value indicating whether to keep multipolygons.
hdf5pattern	A character string specifying the pattern to match HDF5 files.
zLev	An integer specifying the Z level to filter the data. Default is '3L'.
zcolumn	A character string specifying the column name for the Z index.
geometry	A character string specifying the geometry column name.
verbose	A logical value indicating whether to print additional information.

Value

An 'sf' object containing the MERFISH polygon data.

Examples

```
example(readMerfishSPE)
polygons <- readPolygonsMerfish(metadata(spe)$polygons, type="parquet")
polygons
```

readPolygonsXenium *readPolygonsXenium*

Description

This function reads polygon data specific to Xenium technology.

Usage

```
readPolygonsXenium(
  polygonsFile,
  type = c("parquet", "csv"),
  x = "vertex_x",
  y = "vertex_y",
  keepMultiPol = TRUE,
  verbose = FALSE
)
```

Arguments

polygonsFile	A character string specifying the file path to the polygon data.
type	A character string specifying the file type ("parquet" or "csv"). Default is parquet.
x	A character string specifying the x-coordinate column.
y	A character string specifying the y-coordinate column.
keepMultiPol	A logical value indicating whether to keep multipolygons.
verbose	A logical value indicating whether to print additional information.

Value

An ‘sf‘ object containing the Xenium polygon data.

Examples

```
example(readXeniumSPE)
polygons <- readPolygonsXenium(metadata(spe)$polygons, type="parquet")
polygons
```

`readXeniumSPE` *Load data from a 10x Genomics Xenium experiment*

Description

Creates a [‘SpatialExperiment‘] from an unzipped Xenium Output Bundle directory containing spatial gene expression data.

Usage

```
readXeniumSPE(
  dirName,
  sampleName = "sample01",
  type = c("HDF5", "sparse"),
  coordNames = c("x_centroid", "y_centroid"),
  boundariesType = c("parquet", "csv"),
  computeMissingMetrics = TRUE,
  keepPolygons = FALSE,
  countsFilePattern = "cell_feature_matrix",
  metadataFPattern = "cells.csv.gz",
  polygonsFPattern = "cell_boundaries",
  polygonsCol = "polygons",
  txPattern = "transcripts",
  addFOVs = FALSE
)
```

Arguments

`dirName`	‘character(1)‘ Path to a Xenium Output Bundle directory.
`sampleName`	‘character(1)‘ Sample identifier to assign to ‘sample_id‘. Default: ‘"sample01"‘.
`type`	‘character(1)‘ One of ‘"HDF5"‘ or ‘"sparse"‘; method to read the feature matrix.
`coordNames`	‘character(2)‘ Names of X/Y spatial coordinate columns. Default: ‘c("x_centroid", "y_centroid")‘.
`boundariesType`	‘character(1)‘ One of ‘"parquet"‘ or ‘"csv"‘; format of the polygon file.
`computeMissingMetrics`	‘logical(1)‘ If ‘TRUE‘, compute area and aspect-ratio from boundary polygons.
`keepPolygons`	‘logical(1)‘ If ‘TRUE‘, append raw polygon geometries to ‘colData‘.
`countsFilePattern`	‘character(1)‘ Pattern to locate the feature matrix file. Default: ‘"cell_feature_matrix"‘.
`metadataFPattern`	‘character(1)‘ Pattern to locate the cell metadata file. Default: ‘"cells"‘.
`polygonsFPattern`	‘character(1)‘ Pattern to locate the cell boundaries file. Default: ‘"cell_boundaries"‘.
`polygonsCol`	‘character(1)‘ Name of the polygons column to add to ‘colData‘. Default: ‘"polygons"‘.

txPattern	‘character(1)‘ Pattern (base filename, without extension) to locate the transcript file (usually a ‘.parquet‘ file) from which to extract Field-Of-View (FOV) information for each cell. Default: ‘"transcripts"‘.
addFOVs	‘logical(1)‘ If ‘TRUE‘, extract Field-Of-View (FOV) information from the transcript file (as located by ‘txPattern‘) and append it to cell metadata (‘colData‘). Default: ‘FALSE‘.

Details

readXeniumSPE

Expects the unzipped bundle to contain an ‘outs/‘ folder with: - ‘cell_feature_matrix.h5‘ or ‘cell_feature_matrix/‘ - ‘cells.csv.gz‘

Value

A [‘SpatialExperiment‘] object with assays, ‘colData‘, spatial coordinates, and ‘metadata$polygons‘ & ‘metadata$technology‘.

Author(s)

Dario Righelli, Benedetta Banzi

Examples

```
xepath <- system.file(
  "extdata", "Xenium_small", package = "SpaceTrooper"
)
(spe <- readXeniumSPE(
  dirName = xepath,
  keepPolygons = TRUE
))
```

SpaceTrooper-deprecated

Deprecated functions in SpaceTrooper

Description

These functions are retained for compatibility with published SpaceTrooper APIs. They issue a deprecation warning and direct users to the canonical Quality Score functions.

Usage

```
computeQCScore(spe, bestLambda = NULL, modelFormula = NULL, verbose = FALSE)

computeQCScoreFlags(spe, qsThreshold = 0.5, useQSQuantiles = FALSE)

computeOutliersQCScore(
```

```
  spe,
  metricList = c("log2SignalDensity", "Area_um", "log2AspectRatio",
    "log2Ctrl_total_ratio")
)

applyQCScoreModel(spe, qcModel, scoreName = "QC_score")
```

Arguments

spe	A 'SpatialExperiment' object with QC metrics already computed.
bestLambda	Passed to computeQScore.
modelFormula	Passed to computeQScore.
verbose	Passed to computeQScore.
qsThreshold	Passed to computeQScoreFlags.
useQSQuantiles	Passed to computeQScoreFlags.
metricList	Passed to computeOutliersQScore.
qcModel	A historical QC score model object, usually stored in 'metadata(spe)$QCScore_model'.
scoreName	Name of the legacy output column in 'colData'.

Details

- 'computeQCScore()' replaces canonical 'QScore' and 'metadata(spe)$QScore_model' with historical 'QC_score' and 'metadata(spe)$QCScore_model'.
- 'computeQCScoreFlags()' consumes 'QC_score' and creates 'low_qcscore' and, when applicable, 'low_threshold_qcscore'.
- 'computeOutliersQCScore()' is the historical name for 'computeOutliersQScore()'.
- 'applyQCScoreModel()' retains the 'qcModel' argument, the default 'QC_score' output, and 'metadata(spe)$QCScore_model_applied'.

Use 'computeQScore()', 'computeQScoreFlags()', 'computeOutliersQScore()', and 'applyQScoreModel()' for canonical names.

Value

A 'SpatialExperiment' object; see computeQScore.

A 'SpatialExperiment' object; see computeQScoreFlags.

A 'SpatialExperiment' object; see computeOutliersQScore.

A 'SpatialExperiment' object with the applied score in 'colData' and the model in 'metadata(spe)$QCScore_model_applied'.

spatialPerCellQC *spatialPerCellQC*

Description

Computes quality-control metrics for each cell and adds them to ‘colData‘.

Usage

```
spatialPerCellQC(
  spe,
  micronConvFact = 0.12,
  rmZeros = TRUE,
 negProbList = c("NegPrb", "Negative", "SystemControl", "Ms IgG1", "Rb IgG", "BLANK_",
    "NegControlProbe", "NegControlCodeword", "UnassignedCodeword", "Blank"),
  use_altexps = NULL
)
```

Arguments

spe	A ‘SpatialExperiment‘ object containing spatial omics data.
micronConvFact	Numeric factor to convert pixels to microns. Default is 0.12.
rmZeros	logical for removing zero total counts (protein intensities) cells (default is TRUE).
negProbList	Character vector of patterns to identify negative probes. Defaults include: Nanostring CosMx: ‘"NegPrb"‘, ‘"Negative"‘, ‘"SystemControl"‘ Xenium: ‘"NegControlProbe"‘, ‘"NegControlCodeword"‘, ‘"UnassignedCodeword"‘ MERFISH: ‘"Blank"‘
use_altexps	logical for ‘use_altexps‘ in ‘scuttle‘ package. If TRUE uses the altexps for computing some metrics on it. Useful for interoperability with ‘SpatialExperimentIO‘. (See addPerCellQC for additional details).

Details

The function computes and appends per-cell QC metrics to ‘colData(spe)‘ and may subset the returned SpatialExperiment.

Key behaviours and expectations: - Feature detection: negative-probe patterns supplied in ‘negProbList‘ are passed to ‘scater::addPerCellQC()‘ (via ‘use_altexps‘ when requested). ‘addPerCellQC()‘ must be able to find matching control feature names (‘rownames‘ of spe) and will create ‘subsets_*_sum‘ and ‘subsets_*_detected‘ columns used below for each of control feature types.

The function appends ‘sum‘, ‘detected‘ and ‘total‘ to the SPE colData by calling internally ‘addPerCellQC()‘; these are used to compute ‘control_sum‘, ‘control_detected‘, ‘target_sum‘ and ‘target_detected.‘

- Control metrics: ‘control_sum‘ and ‘control_detected‘ are computed by summing matching ‘subsets_*‘ columns; ‘target_*‘ metrics are computed as sum(detected) - control_sum(_detected).

- Ratios and logs: ‘ctrl_total_ratio‘ (‘control_sum‘/‘sum‘) and its stabilized log2-transformed ‘log2Ctrl_total_ratio‘ are added.

- Coordinate and area handling: - For CosMx technologies (Nanostring_CosMx and Nanostring_CosMx_Protein) spatial coordinates are converted from pixels to microns using ‘micronConvFact‘ and appended to ‘colData‘ (column names have px -> um).

- For CosMx, ‘Area_um‘ is derived from an existing Area column scaled by micronConvFact^2 and ‘.computeBorderDistanceCosMx()‘ is invoked to compute ‘dist_border‘.

- For Nanostring_CosMx_Protein, a legacy ‘Area.um2‘ column (if present) is renamed to ‘Area_um‘ to standardize naming.

- For Xenium (10X_Xenium), if ‘Area_um‘ is missing the function will attempt to use ‘cell_area‘ as a fallback and issue a warning.

- Aspect ratio: if ‘AspectRatio‘ exists it is logged (‘log2AspectRatio‘); if missing a warning is emitted.

- Signal density: ‘SignalDensity‘ is computed as ‘sum‘ / ‘Area_um‘ for most technologies; for Nanostring_CosMx_Protein it coincides with ‘total‘. A log2-transformed ‘log2SignalDensity‘ is also added.

- Zero-count (protein intensity) removal: when ‘rmZeros = TRUE‘ cells with ‘sum == 0‘ are removed from the returned SpatialExperiment (message printed).

- Side effects: the function modifies ‘colData(spe)‘ (adds multiple new columns). It may add spatial coordinates into ‘colData‘ if missing, and may subset the SPE to remove zero-count cells. It issues warnings when expected inputs (e.g. area, aspect ratio, polygon-derived fields) are missing or when fallbacks are used.

Use this information to ensure the input SpatialExperiment contains the necessary assays and fields (feature names, ‘sum‘/‘detected‘/‘total‘, ‘Area‘/‘cell_area‘, log2AspectRatio, dist_border when available) so metrics are computed and assigned correctly.

Value

A ‘SpatialExperiment‘ object with added QC metrics in ‘colData‘.

Examples

```
example(readCosmxSPE)
spe <- spatialPerCellQC(spe)
```

`trainModel` *trainModel*

Description

Fits a ridge logistic regression model.

`trainModel` fits an L2-regularized (ridge) logistic regression using **glmnet**, given a design matrix and a training data frame.

Usage

```
trainModel(modelMatrix, trainDF)
```

Arguments

modelMatrix	a matrix describing the model variables, tipically created with ‘getModelFormula‘ and ‘model.matrix‘ functions.
trainDF	‘data.frame‘ A data frame containing ‘QScore_train‘ or historical ‘qcscore_train‘, coded as 0/1. If both are present, they must agree.

Value

A glmnet model object fitted with `family="binomial"`, `alpha=0` (ridge), and a sequence of λ values.

Examples

```
example(computeTrainDF)
model_formula <- getModelFormula(metadata(spe)$formula_variables)
model_matrix <- model.matrix(as.formula(model_formula), data=df_train)
fit <- trainModel(model_matrix, df_train)
coef(fit, s = 0.01)
```

---

updateCosmxProteinSPE *updateCosmxProteinSPE*

Description

Update a SpatialExperiment object corresponding to Nanostring CosMx Protein data by adding metadata identifying the technology and optionally passing through file-location parameters.

Usage

```
updateCosmxProteinSPE(
  spe,
  dirName,
  sampleName = "sample01",
  coordNames = c("CenterX_global_px", "CenterY_global_px"),
  countMatFPattern = "exprMat_file.csv",
  metadataFPattern = "metadata_file.csv",
  polygonsFPattern = "polygons.csv",
  fovPosFPattern = "fov_positions_file.csv",
  fovdims = c(xdim = 4256, ydim = 4256),
  keepPolygons = FALSE,
  polygonsCol = "polygons"
)
```

Arguments

spe	SpatialExperiment object.
dirName	Directory containing CosMx Protein data files.
sampleName	Character sample ID. Default "sample01".
coordNames	Character vector of length two indicating coordinate column names in the per-cell metadata. Default c("CenterX_global_px","CenterY_global_px").
countMatFPattern	Character pattern for the counts matrix file.
metadataFPattern	Character pattern for the single-cell metadata file.
polygonsFPattern	Character pattern for the polygon file(s).
fovPosFPattern	Character pattern for the FOV positions file.
fovdims	Named numeric vector with FOV size in pixels.
keepPolygons	Logical indicating if the polygons need to be loaded into memory or not (Default is 'FALSE').
polygonsCol	Character name of the column in colData to store polygons.

Details

This function sets metadata(spe)$technology <- "Nanostring_CosMx_Protein". It does not modify other assay or metadata components.

Value

SpatialExperiment object with updated technology metadata.

See Also

readCosmxProteinSPE, readCosmxSPE

Examples

```
protfolder <- system.file( "extdata", "S01_prot", package="SpaceTrooper")
spe <- SpatialExperimentIO::readCosmxSXE(dirName=protfolder,
    addParquetPaths=FALSE)
spe <- updateCosmxProteinSPE(spe, protfolder, sampleName="cosmx_prots")
```

updateCosmxSPE *updateCosmxSPE*

Description

Update a SpatialExperiment object derived from CosMx data by adding polygons, FOV dimensions, standardized column names, and metadata.

Usage

```
updateCosmxSPE(
  spe,
  dirName,
  sampleName = "sample01",
  polygonsFPattern = "polygons.csv",
  fovdims = c(xdim = 4256, ydim = 4256),
  keepPolygons = FALSE,
  polygonsCol = "polygons"
)
```

Arguments

spe	SpatialExperiment object.
dirName	Directory containing CosMx output files (e.g., polygon CSVs).
sampleName	Character scalar, sample identifier stored in `colData(spe)$sample_id`. Default `"sample01"`.
polygonsFPattern	Character, pattern used by `list.files()` to locate polygon files. Default `"polygons.csv"`.
fovdims	Named numeric vector with entries `xdim` and `ydim` representing the FOV dimensions in pixels.
keepPolygons	Logical indicating if the polygons need to be loaded into memory or not (Default is 'FALSE').
polygonsCol	Character name of the column in colData to store polygons.

Details

The function standardizes CosMx SPE structure by: - creating unique cell names of the form `f<fov>_c<cell_ID>`; - ensuring consistent cell identifiers and sample metadata; - recording FOV dimensions, polygon paths, and technology type in `metadata(spe)`.

Value

A SpatialExperiment object with updated metadata and column names.

See Also

readCosmxSPE, readCosmxProteinSPE

Examples

```
cospath <- system.file(file.path("extdata", "CosMx_DBKero_Tiny"),
    package="SpaceTrooper")
spe <- SpatialExperimentIO::readCosmxSXE(dirName=cospath,
    addParquetPaths=FALSE)
spe <- updateCosmxSPE(spe, dirName=cospath, sampleName="DBKero_Tiny")
```

`updateXeniumSPE` *updateXeniumSPE*

Description

Update a SpatialExperiment created from 10x Genomics Xenium outputs by wiring polygons/boundaries, computing optional QC metrics, and standardizing metadata and column names. This is a thin wrapper that delegates to the internal helper '.setupXeniumSPE()'.

Usage

```
updateXeniumSPE(
  spe,
  dirName,
  sampleName = "sample01",
  boundariesType = c("parquet", "csv"),
  computeMissingMetrics = TRUE,
  keepPolygons = FALSE,
  polygonsFPattern = "cell_boundaries",
  polygonsCol = "polygons",
  txPattern = "transcripts",
  addFOVs = FALSE
)
```

Arguments

`spe`	SpatialExperiment object to update.
`dirName`	Directory containing Xenium outputs.
`sampleName`	Sample identifier to store (default `"sample01"`).
`boundariesType`	One of `c("parquet","csv")` indicating the source format of cell boundaries.
`computeMissingMetrics`	Logical; if `TRUE`, compute metrics that are not already present from transcripts/polygons.
`keepPolygons`	Logical; if `TRUE`, keep polygons in the resulting object (e.g., in metadata or colData, depending on implementation).
`polygonsFPattern`	Character pattern used to locate polygon files when boundaries are provided as CSVs (default `"polygons.csv"`).

polygonsCol	Name of the geometry/column storing polygons when reading from parquet (default "polygons").
txPattern	Pattern (file/glob) for transcript-level files (default "transcripts").
addFOVs	Logical; if TRUE, derive and attach FOV identifiers from transcript resources.

Details

This function performs input checks and then calls '.setupXeniumSPE()', which does the heavy lifting (I/O, renaming, metadata updates, metrics).

Value

Updated SpatialExperiment object.

Examples

```
xepath <- system.file("extdata", "Xenium_small", package="SpaceTrooper")
(spe <- SpatialExperimentIO::readXeniumSXE(dirName=xepath))
spe <- updateXeniumSPE(spe, dirName=xepath, boundariesType="parquet",
    computeMissingMetrics=TRUE, keepPolygons=TRUE)
```

Index

.getActiveGeometryName, 3
.renameGeometry, 3
.setActiveGeometry, 4

addPerCellQC, *42*
addPolygonsToSPE, 5
applyQCScoreModel
(SpaceTrooper-deprecated), 40
applyQScoreModel, 5, *14*

checkOutliers, 6
computeAreaFromPolygons, 7
computeAspectRatioFromPolygons, 8
computeCenterFromPolygons, 8
computeLambda, 9
computeMissingMetricsMerfish, 10
computeMissingMetricsXenium, 11
computeOutliersQCScore
(SpaceTrooper-deprecated), 40
computeOutliersQScore, 12, *41*
computeQCScore
(SpaceTrooper-deprecated), 40
computeQCScoreFlags
(SpaceTrooper-deprecated), 40
computeQScore, 13, *41*
computeQScoreFlags, 15, *41*
computeSpatialOutlier, 15
computeThresholdFlags, 17
computeTrainDF, 18

getFencesOutlier, 19
getModelFormula, 20
glmnet, *44*

mc, *16*

plotCellsFovs, 20
plotCentroids, 22
plotMetricHist, 24
plotPolygons, 25
plotQScoreTerms, 26
plotZoomFovsMap, 27

qcFlagPlots, 29

readAndAddPolygonsToSPE, 30
readCosmxProteinSPE (readCosmxSPE), 31
readCosmxSPE, 31
readCosmxSXE, *32*
readMerfishSPE, 32
readPolygons, 34
readPolygonsCosmx, 36
readPolygonsMerfish, 37
readPolygonsXenium, 38
readXeniumSPE, 39

SpaceTrooper-deprecated, 40
spatialPerCellQC, 42

trainModel, 43

updateCosmxProteinSPE, 44
updateCosmxSPE, 46
updateXeniumSPE, 47